Fast Algorithms
for Digital Signal Processing

Richard E. Blahut

Addison-Wesley Publishing Company
Reading, Massachussetts Menlo Park, California
Wokingham, Berkshire Amsterdam Don Mills, Ontario Sydney

Р. Блейхут

БЫСТРЫЕ АЛГОРИТМЫ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Перевод с английского
И. Б. Грушко

Москва «Мир» 1989

ББК 32.811
Б68
УДК 519.725

Блейхут Р.

Б68 Быстрые алгоритмы цифровой обработки сигналов: Пер. с англ. — М.: Мир, 1989. — 448 с., ил.

ISBN 5-09-001009-2

Книга американского специалиста, посвященная актуальным прикладным задачам построения быстрых алгоритмов цифровой обработки сигналов (автор известен по его «Теории и практике кодов, контролирующих ошибки» (М.: Мир, 1986)). Для ускорения типичных для таких задач вычислений используется организация данных в виде конечных алгебраических структур (групп, колец, полей), что позволяет применить структурные теоремы алгебры и теории чисел. В двух из двенадцати глав книги содержится краткое, но строгое и систематическое изложение соответствующих разделов математики, как правило, недостаточно известных инженерам-прикладникам.

Для математиков-прикладников, программистов, инженеров — разработчиков систем обработки дискретных сигналов, студентов и аспирантов университетов.

Б $\frac{1402030000-130}{041\ (01)-89}$ 2—89

ББК 32.811

Редакция литературы по математическим наукам

ISBN 5-09-001009-2 (русск.)
ISBN 0-201-10155-6 (англ.)

Copyright © 1985 by Addison-Wesley Publishing Company, Inc.
© перевод на русский язык, «Мир», 1989

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ

Предлагаемая читателю книга известного американского специалиста в области теории информации и ее приложений Р. Блейхута посвящена построению быстрых алгоритмов цифровой обработки сигналов — вычислительных алгоритмов, повсеместно возникающих в таких приложениях, как все виды связи, радиолокация, радиоастрономия, цифровая голография, медицинская электроника и т. п. Отсутствие подобной книги остро ощущалось многими специалистами в перечисленных областях — конструкторами информационных систем различного назначения. В частности, Р. Блейхут указывает, что он почувствовал ее необходимость во время работы над своей предыдущей книгой «Теория и практика кодов, контролирующих ошибки», в которую ему пришлось включить несколько специальных глав с описанием алгоритмов быстрого вычисления преобразования Фурье, которые, конечно, никак не зависят от данного конкретного приложения. (Книга была переведена в 1985 г. на русский язык и быстро разошлась.)

Хотя в настоящую книгу вошли и алгоритмы решения теплицевых систем уравнений (возникающих в таких приложениях, как линейное предсказание, построение авторегрессионных фильтров, теория кодирования и др.), и алгоритмы быстрого поиска по древовидному графу (возникающие, например, при декодировании сверточных кодов), сердцевиной книги являются быстрые алгоритмы преобразования Фурье и вычисления цифровой свертки (линейной и циклической).

В основе таких быстрых алгоритмов лежит специальная организация массивов данных в виде конечных алгебраических структур (групп, колец, полей), что создает предпосылки для применения структурных теорем алгебры и теории чисел. Это позволяет строить практически приемлемые алгоритмы, обеспечивающие работу цифровых процессоров в реальном масштабе времени. К настоящему времени накопился широкий ассортимент различных по своей архитектуре и теоретическим предпосылкам алгоритмов, но пока инженеры-разработчики и программисты не знакомы с их теоретическим обоснованием, вопрос о широком использовании этих алгоритмов остается открытым.

Данная книга весьма удачно ликвидирует образовавшийся разрыв. С одной стороны, в ней в двух из 12 глав дается краткое, но строгое и систематическое изложение необходимых разделов алгебры и элементарной теории чисел, как правило, недостаточно известных инженерам-прикладникам. С другой стороны, в остальных 10 главах дается систематическое и исчерпывающее описание накопленных к настоящему времени быстрых алгоритмов преобразования Фурье, вычисления цифровых сверток и решения

систем теплицевых уравнений не только для привычных в инженерной практике полей комплексных и вещественных чисел, но и для полей Галуа.

Книга несомненно будет полезна широкому кругу читателей — математикам-прикладникам, программистам, инженерам-разработчикам систем обработки данных, — а также может быть рекомендована для включения в программу преподавания математики будущим инженерам-конструкторам систем обработки дискретной информации.

В. И. Сифоров

ОТ АВТОРА

Подобно тому как дети перерастают своих родителей, книги могут выйти за рамки возможностей их авторов. Предлагаемый перевод книги «Быстрые алгоритмы цифровой обработки сигналов» является второй моей книгой, вышедшей из умелых рук Инны Грушко, которой я очень благодарен, как и за перевод первой моей книги «Теория и практика кодов, контролирующих ошибки». Мне известно, что не существует «быстрых алгоритмов» переводов, и поэтому я очень ценю тщательную работу над переводом и исправление опечаток, допущенных в английском издании.

Я полагаю, что русские читатели найдут здесь много интересного. Предмет книги, алгебраический по существу, тем не менее ближе к цифровой обработке сигналов, чем к математике. Я надеюсь, что меня простят за почти полное незнание работ советских авторов в данной области.

Р. Э. Блейхут

ПРЕДИСЛОВИЕ

В настоящее время цифровая обработка сигналов переживает взрыв. Ее используют повсюду, включая радиолокацию, сейсмографию, связь, радиоастрономию и медицинскую электронику. Активно разрабатываются и находят рыночный спрос цифровые процессоры — специализированные цифровые компьютеры для обработки сигналов. Такое широкое использование порождает еще более широкий спрос на цифровые процессоры, применяемые в некоторых случаях в массовых масштабах.

Одним из путей удовлетворения этих потребностей является выбор разумно построенных алгоритмов. Вместо того чтобы повышать быстродействие процессора от одного миллиона умножений в секунду до пяти миллионов умножений в секунду, можно для некоторых задач попытаться так организовать вычисления, чтобы быстродействия в один миллион умножений в секунду оказалось достаточно. К настоящему моменту имеется хорошо разработанная теория, позволяющая подойти к решению задач с этих позиций. Она хорошо известна теоретикам, специалистам в данной области, но на практике инженеры-конструкторы часто пренебрегают ей, поскольку она пока недоступна в виде цельного учебного курса. Инженер-конструктор должен хорошо знать предмет, чтобы выбрать алгоритм, нужный в данном конкретном приложении, среди смущающего разнообразия известных быстрых алгоритмов свертки или быстрого преобразования Фурье.

Данная книга является результатом курса, прочитанного автором в Корнеллском университете и корпорации ИБМ под названием «Быстрые алгоритмы цифровой обработки сигналов». Курс был построен с расчетом на стажирующихся инженеров-электриков и аспирантов первого года обучения; его целью было воспитание инженеров, которые свободно владеют предметом. Эта же цель является первой в данной книге.

Второй целью является формирование широкого взгляда на состояние работ в области быстрых алгоритмов цифровой обработки сигналов, который сможет стимулировать новые работы в будущем. Если собрать воедино все нити, то многое становится более очевидным. Например, перенесение БПФ-алгоритмов Кули—Тьюки, Гуда—Томаса и Винограда в произвольное поле облегчило понимание взаимосвязи между многими последующими идеями.

Я думаю, что важно различать быстрый алгоритм, функцию, которую он вычисляет, и приложение, в котором он используется. Это разные элементы, и, когда их смешивают, они могут терять свою ясность. Таким образом, я настаиваю на том, чтобы отличать дискретное преобразование Фурье от быстрого преобразования

Фурье. Первое из них является преобразованием, а второе — алгоритмом для вычисления первого. Подобно этому, алгоритм Витерби не является методом вычисления последовательности максимального правдоподобия; он представляет собой алгоритм вычисления кратчайшего пути на решетке — пути, который в некоторых приложениях будет путем максимального правдоподобия, но не обязан быть им. Но и тогда, когда кратчайший путь действительно является путем максимального правдоподобия, не следует смешивать определение максимального правдоподобия с алгоритмом вычисления его пути. В соответствии с этим подходом в данной книге мало обсуждаются возможные приложения; после постановки задачи все внимание уделяется построению хорошего алгоритма ее решения. Обсуждение возможных приложений цифровой обработки сигналов следует искать в других источниках.

Идея написания этой книги возникла во время работы над более ранней книгой «Теория и практика кодов, контролирующих ошибки». Во многих главах этой книги приходилось рассматривать быстрые алгоритмы вычислений в конечных полях, хотя по существу алгоритмы не зависели от выбранного поля. Я почувствовал, что стоило бы поместить эти алгоритмы в отдельную книгу, описать их независимо от использования и дополнить многими другими важными в цифровой обработке сигналов алгоритмами. Изложение затрагивает многие области теории вычислений и теории алгоритмов. Однако в первую очередь нас интересуют инженерные задачи нахождения лучших алгоритмов цифровой обработки сигналов; асимптотический анализ играет второстепенную роль.

В настоящей книге используются разделы математики, которые могут быть незнакомы типичному читателю с инженерным образованием. Поэтому в книгу включен весь необходимый математический аппарат и строго доказываются все теоремы. Мне представляется, что если этому предмету предстоит стать самостоятельной и зрелой дисциплиной, то вся необходимая математика должна быть частью этой книги; ссылки на другие источники не могут быть адекватной заменой. Инженер не может уверенно овладеть предметом, если ему часто предлагается принять утверждение на веру или обратиться к своей математической библиотеке. Необходимая для построения быстрых алгоритмов математика содержится в гл. 2 и 5. Эти главы можно сначала прочитать бегло, но к ним следует возвращаться по мере необходимости.

Одним из недостатков, которые некоторые читатели заметят в книге, является недостаточность рекомендаций по практическому использованию алгоритмов. Такие темы, как длина машинного слова, погрешность округления и время работы алгоритма А на компьютере В, вообще не рассматриваются. Это сознательное

решение вызвано тем, что я не столь мудр, чтобы делать широкие обобщения по этим вопросам. В немногих встречавшихся мне исследованиях на эти темы всегда рассматривались узко сформулированные задачи, и я не доверяю любому общему выводу, сделанному на основе имеющихся данных. Мне кажется, что конструктору следует решать эти вопросы в контексте конкретной задачи и непосредственно изучать литературу, чтобы узнать, как поступают в аналогичных случаях другие конструкторы.

Среди рассматриваемых в настоящей книге алгоритмов многие уже широко используются в практической деятельности. Другие станут важны в будущих приложениях, когда цифровая обработка сигналов будет более разнообразной и широко применяемой. Некоторая часть, возможно, никогда не станет полезной. Я стремился дать широкий обзор; какие из методов окажутся практически важными, предстоит решить инженерам-конструкторам в следующие несколько десятилетий.

Сердцевиной книги являются описываемые в гл. 3 и 7 алгоритмы циклической свертки и описываемые в гл. 4 и 8 алгоритмы быстрого преобразования Фурье. В гл. 7 и 8 описываются многомерные аналоги алгоритмов гл. 3 и 4 соответственно, и при желании их можно читать сразу после этих глав. Изучение одномерных алгоритмов свертки и преобразования Фурье завершается лишь в контексте многомерных задач. Главы 2 и 5 представляют собой математические введения; некоторые читатели, возможно, предпочтут пользоваться ими как приложением, обращаясь к ним лишь по мере надобности. Главы 6 и 9 следует читать отдельно, следом за гл. 3 и 4. Главы 10, 11 и 12 в основном независимы; каждую из них вполне можно читать независимо от остального материала книги.

БЛАГОДАРНОСТИ

Тем, что я написал данную книгу, я больше всего обязан Шмуэлю Виноградy. Без его существенного вклада в рассматриваемую область она была бы бесформенной и намного более короткой. Он щедро уделял мне время, проясняя многие моменты и просматривая первоначальные наброски. Статьи Винограда, а также книга Нуссбаумера являются источниками большей части материала, обсуждаемого в этой книге.

Книга не достигла бы своего нынешнего уровня без неоднократных рецензирований и критики. Я особенно признателен профессору Б. В. Диксону за несколько чрезвычайно подробных рецензий, которые сильно улучшили ее качество. Я также признателен профессору Тоби Бергеру, профессору К. С. Баррасу, профессору Дж. Гибсону, профессору Дж. Г. Проакису, профессору Т. Б. Парксу, доктору Б. Райсу, профессору И. Сугияме, доктору В. Вандеркулку и профессору Дж. Вергезе за полезную критику. Книга не была бы написана без поддержки компании «Интернейшнал Бизнес Машинз». Я глубоко благодарен ИБМ за эту поддержку, а также Корнеллскому университету за предоставленную возможность апробации первого варианта рукописи с помощью курса лекций. Самую важную поддержку мне оказала моя жена Барбара, без помощи которой эта книга никогда не была бы написана.

Глава 1

ВВЕДЕНИЕ

Вычислительные алгоритмы встречаются повсеместно, и эффективные варианты таких алгоритмов весьма высоко ценятся теми, кто ими пользуется. Мы рассматриваем в основном только некоторые типы вычислений, а именно те, которые связаны с цифровой обработкой сигналов и включают такие задачи, как цифровая фильтрация, дискретное преобразование Фурье, корреляция и спектральный анализ. Наша основная цель состоит в описании современных методов цифровой реализации этих вычислений, причем нас интересует не построение весовых множителей в отводах цифрового фильтра, а организация способа их вычисления при реализации фильтра. Нас также не интересует, зачем кому-то нужно, скажем, дискретное преобразование Фурье; нас заботит только то, как можно вычислить это преобразование эффективно. Удивительно, что для решения столь узко специального вопроса разработана столь глубоко развитая теория.

1.1. Введение в быстрые алгоритмы

Любой алгоритм, подобно большинству инженерных устройств, можно описывать либо через соотношение между входом и выходом, либо детально объясняя его внутреннюю структуру. Применяя к некоторой новой задаче методы цифровой обработки сигналов, мы сталкиваемся с заданием алгоритмов через соотношение вход-выход. При заданном сигнале или записанных некоторым образом данных основное внимание сосредоточивается на том, что надо с этими данными сделать, т. е. каков должен быть выход алгоритма, если на его вход подаются те или иные данные. Примерами такого выхода служат профильтрованная версия входа или его преобразование Фурье. Такая связь входа с выходом в алгоритме математически может быть записана без детального выписывания всех шагов, необходимых для выполнения вычислений.

С этой, опирающейся на вход-выход точки зрения, сама задача построения хороших алгоритмов обработки информации может быть трудной и тонкой, но в данной книге она не рассматривается. Мы предполагаем что уже задан алгоритм типа вход-выход,

описанный в терминах фильтров, преобразований Фурье, интерполяций, прореживаний, корреляций, модуляций, гистограмм, матричных операций и им подобных. Все такие алгоритмы могут быть записаны математической формулой, и, следовательно, вычислены в прямом соответствии с этой записью. Такую реализацию алгоритма вычислений будем называть прямой.

Кто-то может быть вполне удовлетворен прямой реализацией алгоритма; в течение многих лет большинство пользователей считали такие реализации удовлетворительными, и даже сейчас некоторые из них так считают. Но с тех пор, как люди начали решать подобные задачи, другие люди начали искать более эффективные пути их решения. Именно эту историю мы хотим рассказать — историю быстрых алгоритмов. Под быстрыми алгоритмами мы понимаем детальное описание вычислительной процедуры, которая не является очевидным способом вычисления выхода по данному входу. Как правило, быстрый алгоритм жертвует концептуальной ясностью вычислений в пользу их эффективности.

Допустим, что требуется вычислить число A, равное

$$A = ac + ad + bc + bd.$$

В том виде, как оно записано, это вычисление содержит четыре умножения и три сложения. Если число A надо вычислять много раз для различных множеств данных, то мы быстро заметим, что

$$A = (a + b)(c + d)$$

представляет собой эквивалентную форму, требующую лишь одного умножения и двух сложений, так что она предпочтительнее исходной. Этот простенький пример весьма тривиален, но он на самом деле иллюстрирует большинство тем, о которых нам предстоит рассуждать. Все, что мы делаем, можно представлять себе как хитроумную расстановку скобок в вычислениях. Но для больших задач быстрые алгоритмы не удается найти простым просмотром вычислений; их построение потребует весьма развитой теории.

Нетривиальным, хотя все еще простым примером служит быстрый алгоритм вычисления произведения комплексных чисел. Произведение комплексных чисел

$$(e + if) = (a + ib) \cdot (c + id)$$

можно записать через умножения и сложения вещественных чисел:

$$e = ac - bd, \; f = ad + bc.$$

Эти формулы содержат четыре вещественных умножения и два вещественных сложения. Если умножение более трудоемкая опе-

рация, чем сложение, то более «эффективный» алгоритм дается равенствами

$$e = (a - b)\,d + a\,(c - d),$$
$$f = (a - b)\,d + b\,(c + d).$$

В такой форме алгоритм содержит три вещественных умножения и пять вещественных сложений. Если в серии умножений комплексных чисел величины c и d суть константы, то члены $c + d$ и $c - d$ также являются константами, и их можно вычислить заранее, независимо от процесса умножения. Тогда для вычисления одного произведения комплексных чисел нам понадобится три вещественных умножения и три вещественных сложения.

Мы обменяли одно умножение на одно сложение. Это может быть целесообразно, только если в конструкции процессора удастся воспользоваться этим преимуществом. Некоторые процессоры сконструированы в расчете на использование четырех вещественных умножений для вычисления произведения комплексных чисел; в этом случае преимущества улучшенного алгоритма теряются.

Предваряя дальнейшие рассмотрения, задержимся на этом примере подольше. Рассмотренное выше умножение комплексных чисел можно представить в виде

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix},$$

где вектор $[a, b]^T$ представляет комплексное число $a + jb$, матрица $\begin{bmatrix} c & -d \\ d & c \end{bmatrix}$ представляет комплексное число $c + jd$ и вектор $[e, f]^T$ представляет комплексное число $e + jf$. Такое матрично-векторное произведение является одной из форм записи произведения комплексных чисел.

Другой алгорим дается матричным равенством

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c-d & 0 & 0 \\ 0 & c+d & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix},$$

которое можно рассматривать как необычную форму разложения матрицы:

$$\begin{bmatrix} c & -d \\ d & c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} (c-d) & 0 & 0 \\ 0 & (c+d) & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}.$$

В сокращенной записи алгоритм дается равенством

$$\begin{bmatrix} e \\ f \end{bmatrix} = \mathbf{BDA} \begin{bmatrix} a \\ b \end{bmatrix},$$

где $\mathbf{A}$ — матрица размера 3×2, которую мы назовем матрицей предсложений; $\mathbf{D}$ — диагональная матрица размера 3×3, которая включает в себя все общие умножения алгоритма, и $\mathbf{B}$ — матрица размера 2×3, которую мы назовем матрицей постсложений.

Мы увидим, что многие из лучших процедур вычисления свертки и дискретного преобразования Фурье могут быть приведены к такому же матричному виду, в котором центральная матрица является диагональной, а стоящие по ее бокам матрицы содержат в качестве своих элементов только 0 и ±1. Структура таких быстрых алгоритмов сводится к серии сложений, за которой следует серия умножений, за которой следует другая серия сложений.

В качестве последнего примера этого вводного раздела рассмотрим быстрый алгоритм умножения матрицы. Пусть $\mathbf{C} = \mathbf{AB}$, где $\mathbf{A}$ и $\mathbf{B}$ — матрицы размеров $(l\times n)$ и $(n\times m)$ соответственно. Стандартное правило вычисления матрицы $\mathbf{C}$ дается правилами

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \quad i = 1, \ldots, l, \; j = 1, \ldots, m,$$

и содержит в соответствии с этой записью nlm умножений и $(n - 1)\,lm$ сложений. Мы построим алгоритм, который почти в два раза уменьшает число умножений, но увеличивает число сложений, так что общее число операций слегка возрастет.

Воспользуемся применительно к элементам матриц $\mathbf{A}$ и $\mathbf{B}$ тождеством

$$a_1 b_1 + a_2 b_2 = (a_1 + b_2)\,(a_2 + b_1) - a_1 a_2 - b_1 b_2.$$

Предположим, что n четно (в противном случае можно, не меняя произведения $\mathbf{C}$, дополнить матрицу $\mathbf{A}$ нулевыми строками, а матрицу $\mathbf{B}$ нулевыми столбцами). Применяя выписанное тождество к парам строк матрицы $\mathbf{A}$ и к парам столбцов матрицы $\mathbf{B}$, запишем

$$c_{ij} = \sum_{k=1}^{n/2} (a_{i,\,2k-1} + b_{2k,\,j})\,(a_{i,\,2k} + b_{2k-1,\,j}) - \sum_{k=1}^{n/2} a_{i,\,2k-1} a_{i,\,2k} - \sum_{k=1}^{n/2} b_{2k-1,\,j} b_{2k,\,j}, \quad \begin{matrix} i = 1, \ldots, l, \\ j = 1, \ldots, m. \end{matrix}$$

Такая форма вычислений экономит умножения, поскольку второй член зависит только от i и его не надо перевычислять для каждого j, а третий член зависит только от j и его не надо перевычислять для каждого i. Полное число умножений, необходимых для такого вычисления матрицы $\mathbf{C}$, равно $(1/2)\,nlm + (1/2)\,n\,(l + m)$, а полное число сложений равно $(3/2)\,nlm + lm + (n/2 - 1)\,(l + m)$. Для матриц большого размера это

Алгоритм	Умножений на точку	Сложений на точку
Прямое вычисление дискретного преобразования Фурье 1000×1000	8000	4000
БПФ-алгоритм Кули—Тьюки 1024×1024	40	60
Гибридный БПФ-алгоритм Кули—Тьюки/Винограда 1000×1000	40	72.8
БПФ-алгоритм Винограда 1008×1008	6.2	91.6
БПФ-алгоритм Нуссбаумера—Квендалла 1008×1008	4.1	79

Рис. 1.1. Сравнение характеристик некоторых двумерных алгоритмов преобразования Фурье.

составляет примерно половину числа умножений, необходимых в прямом алгоритме.

Последний пример является хорошим поводом для предостережения относительно точности вычислений. Несмотря на то что число умножений уменьшается, описанный алгоритм более чувствителен к погрешностям округления, если не позаботиться об этом специально. Однако можно получить почти такую же точность, как и в прямом алгоритме, если на промежуточных шагах вычислений ввести соответствующие масштабные множители. Вопрос о точности вычислений практически всегда встает при оценке быстрого алгоритма, хотя мы обычно будем им пренебрегать. Иногда при уменьшении числа операций уменьшается и погрешность вычислений, поскольку уменьшается число источников этих помех. В других алгоритмах, хотя число источников помех вычисления уменьшается, ответ может быть столь чувствительным к одному или нескольким из них, что общая погрешность вычислений увеличивается.

Бо́льшая часть книги посвящена рассмотрению всего нескольких задач: задачам вычисления линейной свертки, циклической свертки, многомерных линейной и циклической сверток, дискретного преобразования Фурье, многомерного дискретного преобразования Фурье, решению теплицевых систем уравнений и нахождению пути на решетке. Некоторые из рассматриваемых методов заслуживают более широкого применения; особенно хороши алгоритмы многомерного преобразования Фурье, если взять на себя труд разобраться в наиболее эффективных из них. Для примера на рис. 1.1 приведено сравнение некоторых алгоритмов вычисления двумерного преобразования Фурье. При приближении к концу списка повышение эффективности алгоритмов замедляется. Уменьшение числа умножений на одну точку вычислительной сетки на

выходе от шести до четырех может показаться не очень существенным после того, как уже получено уменьшение от сорока до шести. Но это недальновидная точка зрения. Это дальнейшее улучшение может вполне окупить затраты времени на разработку алгоритма при построении больших систем.

Рис. 1.1 содержит другой важный урок. Вход таблицы, обозначенный как гибридный БПФ-алгоритм Кули—Тьюки/Винограда, соответствует алгоритму вычисления двумерного (1000×1000)-точечного преобразования Фурье, в котором на каждую точку сетки на выходе требуется 40 вещественных умножений. Этот пример может помочь развеять злополучный миф о том, что дискретное преобразование Фурье применимо только тогда, когда длина блока равна степени двух. На самом деле возможность цифровой обработки сигналов не ограничивается только такими длинами блоков; для многих значений длин блоков имеются хорошие алгоритмы.

1.2. Использование быстрых алгоритмов

Сверхбольшие интегральные схемы, называемые чипами, теперь стали доступными. Чип может содержать порядка 100 000 логических элементов, и поэтому неудивительно, что теорию алгоритмов часто рассматривают как способ эффективной организации этих элементов. Иногда выбор алгоритма позволяет существенно улучшить характеристики чипа. Конечно, их можно также улучшить, увеличив размер чипа или повысив его быстродействие; эти возможности более широко известны.

Предположим, что некто придумал алгоритм вычисления преобразования Фурье, содержащий только пятую часть от числа операций, входящих в другой алгоритм вычисления преобразования Фурье. Тогда, используя этот алгоритм, можно реализовать такое же улучшение характеристик чипа, которое получается при увеличении в пять раз его размера или его быстродействия. Для реализации улучшения конструктор чипа должен, однако, перенести архитектуру алгоритма в архитектуру чипа. Непродуманная конструкция может свести на нет это преимущество, увеличивая, например, сложность индексации или поток вход-выход. Разработка оптимальных конструкций в эпоху больших интегральных схем невозможна без понимания быстрых алгоритмов, описываемых в данной книге.

С первого взгляда может показаться, что эти два направления — быстродействующие интегральные схемы и быстрые алгоритмы — конкурируют между собой. Если можно построить достаточно большие и достаточно быстродействующие чипы, то вроде бы нет ничего страшного в том, что используются неэффективные алгоритмы. В некоторых случаях такая точка зрения не вызывает сом-

нений, но заведомо имеются случаи, в которых можно высказать диаметрально противоположную точку зрения. Большие цифровые процессоры часто сами создают потребность в разработке быстрых алгоритмов, так как размерность решаемых на них задач растет. Будет ли процессорное время алгоритма, предназначенного для решения некоторой задачи, пропорционально n^2 или n^3, несущественно при n, равном 3 или 4, но при n, равном 1000, это становится критичным.

Рассматриваемые нами быстрые алгоритмы связаны с цифровой обработкой сигналов, и их приложения столь же широки, сколь и приложения самой цифровой обработки сигналов. Теперь, когда стало практичным разрабатывать изощренный алгоритм цифровой обработки сигналов, который может быть реализован конструктивно посредством одного чипа, хотелось бы иметь возможность выбрать такой алгоритм, который оптимизирует характеристики этого чипа. Но для больших чипов это невозможно сделать без достаточно развитой теории. В своем полном объеме такая теория существенно выходит за рамки включенного в данную книгу материала. Чтобы учесть все аспекты сложности, надо рассматривать и такие продвинутые разделы теории логических схем и архитектуры компьютеров, как параллельное или конвейерное исполнение.

Для оценки алгоритма обычно используют число необходимых умножений и сложений. Эти вычислительные характеристики почти исчерпывают сложность устройства на уровне алгоритма. На более низком уровне оно оценивается площадью чипа или числом логических элементов на нем и временем, необходимым для проведения вычислений. Часто в качестве критерия качества схемы используется произведение времени на площадь. Мы не будем пытаться оценивать характеристики на этом уровне, так как это выходит за рамки задач конструктора алгоритмов.

Актуальность рассматриваемых в данной книге вопросов нельзя оценить без понимания масштабов будущих приложений цифровой обработки сигналов. В настоящее время мы не можем угадать даже размеры таких систем, но достаточно просто можно предвидеть приложения, в которых объем необходимых вычислений будет на несколько порядков больше, чем тот объем, обработку которого может обеспечить современная технология.

Системы звуковой локации в последнее десятилетие стали почти полностью цифровыми. Хотя полоса частот, в которой они работают, равна всего нескольким килогерцам, эти системы выполняют десятки миллионов или сотни миллионов умножений в секунду и еще больше сложений. Такие системы уже сейчас нуждаются в мощном цифровом оборудовании, и стали обычными проекты, требующие еще более мощной цифровой техники.

Радиолокационные системы тоже становятся цифровыми, но многие важные функции по-прежнему реализуются традиционной микроволновой или аналоговой схемотехникой. Для того чтобы увидеть колоссальные потенциальные возможности использования цифровой обработки сигналов в радиолокации, достаточно отметить, что радиолокационные системы в принципе очень похожи на системы звуковой локации, отличаясь от них тем, что используемая полоса частот в 1000 или более раз больше.

Цифровая обработка сейсмической информации является главным методом разведки земных недр, в частности одним из важнейших методов поиска залежей нефти. Обработкой больших пакетов лент данных занято все процессорное время многих компьютеров, но многие вычисления остаются невыполненными.

Компьютерная томография представляет собой широко используемый ныне способ объемного синтеза изображений внутренних органов человека с помощью множественных проекций, получаемых при просвечивании рентгеновскими лучами. Разрабатываются алгоритмы, позволяющие существенно снизить дозы облучения, но требования к цифровой обработке намного превосходят все практически приемлемые в настоящее время. Изучаются и другие способы получения изображений для медицинской диагностики, в том числе с помощью ультразвука, ядерного магнитного резонанса или радиоактивных изотопов.

Неразрушающий контроль качества подукции, например отливок, возможен с помощью воссоздаваемых на компьютере изображений внутренних областей изделия по результатам эхолокации.

В принципе обработка сигналов может быть использована для улучшения качества плохих фотографий, смазанных движением камеры или расфокусировкой. Однако такая цифровая обработка требует большого объема вычислений.

Обработка на цифровом процессоре спутниковых фотографий позволяет совместить несколько изображений или выделить особенности, или скомбинировать полученную на различных длинах волн информацию, или создать синтетический стереоскопический образ. Например, в метеорологических исследованиях можно создать подвижное трехмерное изображение облачного покрова, движущегося над поверхностью земли, используя для этого последовательность спутниковых фотографий, снятых с нескольких точек.

Можно указать и другие приложения быстрых алгоритмов для цифровой обработки сигналов, но уже приведенных достаточно для того, чтобы доказать, что потребность в них существует и продолжает расти.

Каждое из описанных приложений связано с большим количеством вычислений, структура которых, однако, чрезвычайно

прозрачна и сильно упорядочена. Кроме того, если для решения некоторой задачи уже создан жесткий модуль или подпрограмма, то они и будут постоянно использоваться для этой задачи. Имеет смысл затратить усилия на разработку добротной схемы, так как хорошие операционные характеристики намного важнее стоимости разработки.

1.3. Системы счисления для проведения вычислений

Во всей книге, говоря о сложности алгоритма, мы имеем в виду число умножений и сложений, поскольку эти операции выбраны в качестве единиц измерения сложности. Кто-то может захотеть пойти дальше и, чтобы подсчитать число необходимых битовых операций, поинтересоваться тем, как построено устройство умножения. Структура умножителя и сумматора существенно зависит от вида представления данных. Хотя вопросы, связанные с системами счисления, выходят за рамки данного курса, несколько слов во введении об этом сказать следует.

Возьмем крайний случай, полагая, что основная часть вычислений состоит из умножений. Тогда сложность можно понизить, записывая данные в логарифмическом виде. Сложение при этом станет более трудоемким, но если сложений не слишком много, то общий результат улучшится. Тем не менее, как правило, мы будем полагать, что входные данные записываются в одном из естественных видов — как вещественные, как комплексные или как целые числа.

В практической реализации процессоров для цифровой обработки сигналов имеются еще более тонкие вопросы; используются формы записи числа как с плавающей, так и с фиксированной точкой. Для большинства задач цифровой обработки вполне достаточной оказывается арифметика с фиксированной точкой и в этих случаях ее и надо выбрать из соображений экономичности. Этого положения не следует придерживаться слишком строго. Всегда есть соблазн избавиться от многих трудностей проектирования, перейдя к арифметике с плавающей точкой. Однако если чип или алгоритм предназначены для одного-единственного применения — например цифровой фильтр для цифровой радиосхемы массового выпуска, — то не стоимость разработки этого чипа играет основную роль; решающую роль играют эксплуатационные характеристики изделия и стоимость его тиражирования. Деньги, затраченные на облегчение работы конструктора, нельзя потратить на улучшение характеристик.

Неотрицательное целое число j, меньшее, чем q^m, в m-символьной позиционной арифметике с фиксированной точкой по основанию q записывается в виде

$$j = j_0 + j_1 q + j_2 q^2 + \ldots + j_{m-1} q^{m-1}, \quad 0 \leqslant j_i < q.$$

Число j представляется m-последовательностью коэффициентов $(j_0, j_1, \ldots, j_{m-1})$ [1]. Имеется несколько способов учета знака числа с фиксированной точкой; эти способы записи отрицательных чисел называются соответственно прямым кодом, дополнительным кодом и обратным кодом. При любом основании q правила такого представления одни и те же. В двоичном случае, $q = 2$, дополнительный и обратный коды иногда называются соответственно 2-дополнительным и 1-дополнительным.

Наиболее прост для понимания прямой код. Для записи знака числа в нем выделяется специальная позиция, которая называется знаковой и в которую записывается 0, если число положительно, и 1, если число отрицательно. В процессе выполнения операций сложения и умножения знаковая позиция обрабатывается отдельно от значимых позиций. Часто предпочтение отдается дополнительному или обратному кодам, поскольку они проще реализуются в жестком исполнении; сумматор просто складывает два числа, не делая различия между знаковой и значащими позициями. Как прямой, так и обратный коды приводят к двум разным записям нуля — положительной и отрицательной, которые следует полагать равными. В дополнительном коде нуль записывается однозначно.

В обратном коде изменение знака числа получается заменой каждой цифры j, включая и знаковую цифру, на $q - 1 - j$. Например, взятое с обратным знаком десятичное число $+62$ (которое записывается как 062) равно в обратном коде 937, а взятое с обратным знаком двоичное число $+011$ (которое записывается как 0011) равно в обратном коде 1100. Обратный код обладает тем свойством, что для умножения любого числа на минус один достаточно каждую цифру заменить ее дополнением до $q - 1$.

В дополнительном коде изменение знака каждого числа получается прибавлением единицы к записи этого числа в обратном коде. По определению минус нуль равен нулю. При таком соглашении взятое с обратным знаком десятичное число $+62$, которое записывалось как 062, равно в дополнительном коде 938, а взятое с обратным знаком двоичное число $+011$, которое записывалось как 0011, равно в дополнительном коде 1101.

[1] В стандартной позиционной записи наиболее значимая цифра располагается не справа а слева, так что число записывается в виде $(j_{m-1}, j_{m-2}, \ldots, j_1, j_0)$. — *Прим. перев.*

1.4. Цифровая обработка сигналов

Наиболее важной задачей цифровой обработки сигналов является задача фильтрации длинной последовательности чисел, а наиболее важным устройством — цифровой фильтр. Как правило, длина последовательности данных заранее неизвестна и является столь большой, что ее можно полагать при обработке бесконечной. Числа, составляющие последовательность, являются обычно либо вещественными, либо комплексными, но нам предстоит работать и с другими видами чисел. Цифровой фильтр представляет собой устройство, формирующее новую числовую последовательность, называемую выходной последовательностью, по заданной последовательности, которая теперь называется входной последовательностью. Широко используемые фильтры можно строить из элементов, схемы которых приведены на рис. 1.2 и которые называются *разрядами регистра сдвига*, *сумматорами*, *умножителями на скаляр* и *умножителями*. Разряд регистра сдвига содержит одно число, которое поступает на его выход. В дискретные моменты времени, именуемые *тактами*, разряд регистра сдвига замещает содержащееся в нем число числом, поступающим на его вход, отбрасывая предыдущее содержимое. Как показано на рис. 1.3, *регистр сдвига* представляет собой несколько соединенных в цепь разрядов регистра сдвига. Регистр сдвига называется также *линией задержки*.

Мы будем рассматривать два наиболее важных типа регистров сдвига, известных под названием *фильтра с конечным импульсным откликом* (КИО-фильтр) и *авторегрессионного фильтра*. КИО-фильтр, как показано на рис. 1.4, представляет собой просто линию задержки с отводами, в которых выход каждого разряда умножается на фиксированную константу, а результаты складываются вместе. Как показано на рис. 1.5, авторегрессионный фильтр также является линией задержки с отводами, но с обратной связью, соединяющей выход со входом. Выходная последовательность КИО-фильтра равна линейной свертке входной последовательности и последовательности, описываемой весовыми множителями в отводах фильтра.

Линейная свертка является едва ли не самой общей вычислительной задачей цифровой обработки сигналов, и большую долю времени мы потратим на исследование вопроса ее эффективной реализации. Еще больше времени мы уделим методам эффективного вычисления циклической свертки. Это может показаться странным, так как в приложениях циклическая свертка возникает естественным образом не столь уж часто. Наше внимание к циклической свертке обусловлено тем, что для ее вычисления имеется много хороших алгоритмов. Это позволяет разработать быстрые методы вычисления очень длинных линейных сверток, связывая вместе много циклических сверток.

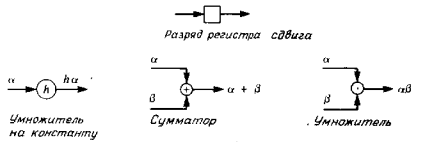


Рис. 1.2. Элементы схем.

Рис. 1.3. Регистр сдвига.

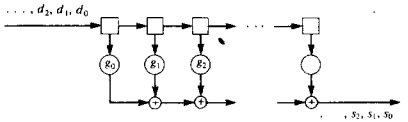


Рис. 1.4. КИО-фильтр.

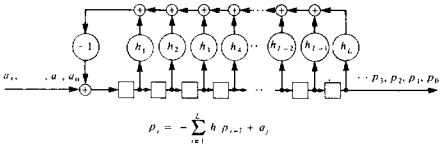


Рис. 1.5. Авторегрессивный фильтр.

Для заданных двух последовательностей — последовательности данных

$$\mathbf{d} = \{d_i,\ i = 0, \ldots, N-1\}$$

и последовательности фильтра

$$\mathbf{g} = \{g_i,\ i = 0, \ldots, L-1\},$$

где N — длина блока данных и L — длина фильтра, линейной сверткой называется новая последовательность

$$\mathbf{s} = \{s_i,\ i = 0, \ldots, L+N-2\},$$

элементы которой определяются равенствами

$$s_i = \sum_{k=0}^{N-1} g_{i-k} d_k, \quad i = 0, \ldots, L+N-2,$$

а длина выходного блока равна $L + N - 1$; последовательность $\mathbf{s}$ называется выходной или сигнальной. В записи свертки подразумевается, что $g_{i-k} = 0$ при $i - k < 0$. Так как все компоненты последовательности $\mathbf{g}$ умножаются на все компоненты последовательности $\mathbf{d}$, то прямой метод вычисления свертки содержит NL умножений.

Имеется очень разработанная теория построения КИО-фильтров, связанная с выбором длины L и весовых множителей $\{g_i\}$. Этот аспект теории конструкций фильтров мы не рассматриваем; предметом наших исследований являются быстрые алгоритмы вычисления выходной последовательности $\mathbf{s}$ по заданным последовательностям фильтра $\mathbf{g}$ и входа $\mathbf{d}$.

Со сверткой тесно связана еще одна величина, которая называется *корреляцией* и определяется равенствами

$$r_i = \sum_{i=0}^{N-1} g_{i+k} d_k, \quad i = 0, \ldots, L + N - 2,$$

где $g_{i+k} = 0$ при $i + k \geqslant L$. Корреляцию можно вычислять как свертку, если просто прочитать одну из определяющих ее последовательностей в обратном порядке. Все способы вычисления линейной свертки легко преобразуются в способы вычисления корреляции.

Свертку можно записать также через многочлены. Пусть

$$d(x) = \sum_{i=0}^{N-1} d_i x^i \text{ и } g(x) = \sum_{i=0}^{L-1} g_i x^i.$$

Тогда $s(x) = g(x)\, d(x)$, где $s(x) = \sum_{i=0}^{L+N-1} s_i x^i$. Эти равенства легко проверяются простым вычислением коэффициентов произведения $g(x)\, d(x)$. Можно, конечно, записать также равенство $s(x) = d(x)\, g(x)$, из которого становится очевидной симметричная роль последовательностей $\mathbf{g}$ и $\mathbf{d}$ в определении свертки. Таким образом, линейная свертка может быть записана в эквивалентном виде как

$$s_i = \sum_{k=0}^{L-1} g_k d_{i-k}.$$

Другой формой свертки, тесно связанной с линейной сверткой, является циклическая свертка. Для заданных двух последовательностей $\{d_i,\ i = 0, \ldots, n-1\}$ и $\{g_i,\ i = 0, \ldots, n-1\}$ с одной и той же длиной блока n циклическая свертка $\{s'_i,\ i = 0, \ldots, n-1\}$ с длиной блока n определяется равенствами

$$s'_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k, \quad i = 0, \ldots, n-1,$$

где двойные скобки означают, что вычисления над индексами производятся в арифметике по модулю n (см. разд. 2.6). Иными словами, $((n-k)) = n - k \pmod n$ и $0 \leqslant ((n-k)) < n$. Заметим, что в циклической свертке при каждом i каждое d_k умножается на значимую величину $g_{((i-k))}$. Это существенное отличие от линейной свертки, в которой d_k часто умножается на член g_{i-k}, индекс которого выходит за диапазон определения последовательности $\mathbf{g}$, так что соответствующая компонента g_{i-k} равна нулю.

Циклическую свертку можно связать с линейной следующим образом. По определению циклической свертки

$$s'_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k, \quad i = 0, \ldots, n-1.$$

Разделим эту сумму на две, выделяя в первую сумму члены, индексы которых удовлетворяют условию $i - k \geqslant 0$ (или $k \leqslant i$), а во вторую — члены, индексы которых удовлетворяют условию $i - k < 0$ (или $k > i$):

$$s'_i = \sum_{k=0}^{i} g_{i-k} d_k + \sum_{k=i+1}^{n-1} g_{n+i-k} d_k.$$

Полагая теперь в первой сумме $g_{i-k} = 0$ при $k > i$, а во второй $g_{n+i-k} = 0$ при $k < i$, можно изменить границы суммирования и получить связь циклической и линейной сверток в виде

$$s'_i = \sum_{k=0}^{n-1} g_{i-k} d_k + \sum_{k=0}^{n-1} g_{n+i-k} d_k = s_i + s_{n+i}, \quad i = 0, \ldots, n-1.$$

Скажем, что члены последовательности $\mathbf{s}$, индексы которых больше $n - 1$, «вкладываются» обратно в члены, индексы которых меньше n.

Если второй член выписанной выше суммы равен нулю, то линейную свертку можно вычислять как циклическую. Это возможно, если произведения $g_{n+i-k} d_k$ равны нулю для всех i и k. Чтобы обеспечить эти условия, можно так выбрать длину n циклической свертки, чтобы она была больше, чем $N + L - 1$ (пополняя нулями $\mathbf{g}$ и $\mathbf{d}$ до длины блока n). Тогда для вычисления линейной свертки можно пользоваться алгоритмом вычисления циклической свертки и получать при этом правильный ответ.

Циклическую свертку можно также выразить в виде произведения многочленов. Пусть

$$d(x) = \sum_{i=0}^{n-1} d_i x^i, \quad g(x) = \sum_{i=0}^{n-1} g_i x^i$$

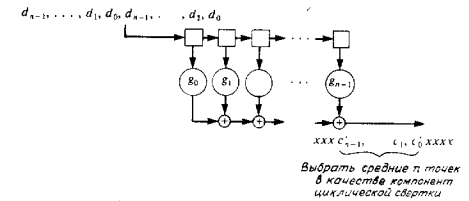


Рис. 1.6. Использование КИО-фильтра для формирования циклической свертки.

и $s(x) = g(x)\, d(x)$. Циклическая свертка вычисляется по многочлену $s(x)$ «обратным вложением» членов высшего порядка. Это можно представить в виде записи

$$s'(x) = s(x) \pmod{x^n - 1},$$

где равенство по модулю $x^n - 1$ означает, что $s'(x)$ равен остатку от деления многочлена $s(x)$ на многочлен $x^n - 1$. Таким образом,

$$s'(x) = g(x)\, d(x) \pmod{x^n - 1}.$$

Для приведения многочлена $g(x)\, d(x)$ по модулю $x^n - 1$ достаточно заменить x^n на 1, или, что эквивалентно, член x^{n+i} с положительным i на член x^i. Это соответствует формированию величин

$$s'_i = s_i + s_{n+i}, \quad i = 0, \ldots, n - 1,$$

и, следовательно, позволяет вычислить коэффициенты циклической свертки.

Так как

$$s'(x) = d(x)\, g(x) = g(x)\, d(x) \pmod{x^n - 1},$$

то ясно, что векторы $\mathbf{d}$ и $\mathbf{g}$ играют в определении свертки симметричную роль, так что циклическая свертка задается двумя эквивалентными равенствами

$$s'_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k =$$

$$= \sum_{k=0}^{n-1} d_{((i-k))} g_k, \quad i = 0, \ldots, n - 1.$$

На рис. 1.6 приведена схема КИО-фильтра, вычисляющего циклическую свертку, для чего последовательность $\mathbf{d}$ повторяется дважды. На выходе КИО-фильтра формируется последовательность из $3n - 1$ компонент, среди которых содержатся n последовательных компонент, равных компонентам циклической свертки.

Более важным для приложений является использование циклической свертки для вычисления длинной линейной свертки. В быстрых алгоритмах вычисления длинной линейной свертки входная последовательность разбивается на короткие секции, содержащие иногда несколько сотен отсчетов. Для формирования потока данных на выходе секции обрабатываются поочередно — зачастую методом циклической свертки. Такие способы называются методами с *перекрытием*; в этом названии отражается тот факт, что неперекрывающиеся секции в потоке входных данных приводят к перекрывающимся секциям в потоке выходных данных, а перекрывающиеся секции в потоке входных данных приводят к неперекрывающимся секциям в потоке выходных данных.

Работа показанного на рис. 1.5 авторегрессионного фильтра также может быть описана в терминах полиномиальной арифметики. Но если КИО-фильтр вычисляет произведение многочленов, то авторегрессионный фильтр выполняет деление многочленов. А именно, при фильтрации конечной последовательности в авторегрессионном фильтре (с нулевым начальным состоянием) на выходе фильтра формируется последовательность коэффициентов многочлена-частного, получаемого при делении многочлена, коэффициенты которого равны компонентам входной последовательности, на многочлен, коэффициенты которого задаются весовыми множителями в отводах фильтра; к моменту завершения ввода входной последовательности содержимое фильтра равно коэффициентам многочлена-остатка от такого деления. Напомним, что авторегрессионный фильтр описывается равенством

$$p_j = -\sum_{i=0}^{L} h_i p_{j-i} + a_j,$$

где a_j есть j-й символ на входе, а h_i — весовой множитель в i-м отводе фильтра. Определим многочлены

$$a(x) = \sum_{i=0}^{n-1} a_i x^i \quad \text{и} \quad h(x) = \sum_{i=0}^{L} h_i x^i$$

и запишем равенство

$$a(x) = Q(x)\, h(x) + r(x),$$

где $Q(x)$ и $r(x)$ обозначают соответственно частное и остаток в алгоритме деления многочленов. Тогда отсчет p_j на выходе фильтра равен в точности j-му коэффициенту многочлена-частного, а коэффициенты многочлена-остатка $r(x)$ будут записаны в самых левых разрядах регистра сдвига авторегрессионного фильтра после того, как на его вход будут поданы все n коэффициентов a_j делимого.

Другим очень важным в цифровой обработке сигналов видом вычисления является дискретное преобразование Фурье (с этих пор называемое в дальнейшем просто преобразованием Фурье). Пусть $\mathbf{v} = \{v_i,\ i = 0,\ \ldots,\ n-1\}$ обозначает вектор с вещественными или комплексными компонентами. Преобразованием Фурье вектора $\mathbf{v}$ называется вектор $\mathbf{V}$ длины n с комплексными компонентами, задаваемыми равенствами

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i,\quad k = 0,\ \ldots,\ n-1,$$

где $\omega = e^{-j2\pi/n}$ и $j = \sqrt{-1}$ [1]).

Иногда это определение записывается в матричном виде

$$\mathbf{V} = \mathbf{T}\mathbf{v}.$$

Если раскрыть эту запись, то она принимает вид

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ \cdot \\ \cdot \\ V_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \cdot \\ \cdot \\ v_{n-1} \end{bmatrix}.$$

Если вектор $\mathbf{V}$ равен преобразованию Фурье вектора $\mathbf{v}$, то вектор $\mathbf{v}$ можно вычислить по вектору $\mathbf{V}$ с помощью обратного преобразования Фурье, задаваемого равенством

$$v_i = \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-ik} V_k.$$

Доказательство этого факта дается следующими выкладками:

$$\sum_{k=0}^{n-1} \omega^{-ik} V_k = \sum_{k=0}^{n-1} \omega^{-ik} \sum_{l=0}^{n-1} \omega^{lk} v_l = \sum_{l=0}^{n-1} v_l \left[\sum_{k=0}^{n-1} \omega^{-k\,(l-i)} \right].$$

Если $l = i$, то сумма по k, очевидно, равна n, а если l не равно i, то сумма приводится к виду

$$\sum_{k=0}^{n-1} \left(\omega^{-(l-i)}\right)^k = \frac{1 - \omega^{-(l-i)\,n}}{1 - \omega^{-(l-i)}}.$$

[1]) Во всей книге буква j используется и для обозначения $\sqrt{-1}$ и в качестве индекса. Это не приводит, однако, ни к какой путанице.

Так как $\omega^{-n} = 1$, то справа стоит нуль. Следовательно,

$$\sum_{k=0}^{n-1} \omega^{-ik} V_k = \sum_{l=0}^{n-0} v_l\,(n\delta_{il}) = n v_i,$$

где $\delta_{il} = 1$ при $i = l$ и $\delta_{il} = 0$ в противном случае.

Между циклической сверткой и преобразованием Фурье имеется очень важная связь, которая известна как *теорема о свертке* и формулируется следующим образом. Вектор $\mathbf{e}$ равен циклической свертке векторов $\mathbf{f}$ и $\mathbf{g}$,

$$e_i = \sum_{l=0}^{n-1} f_{((i-l))} g_l, \qquad i = 0,\ \ldots,\ n-1,$$

тогда и только тогда, когда их преобразования Фурье удовлетворяют равенствам

$$E_k = F_k G_k, \qquad k = 0,\ \ldots,\ n-1.$$

Это утверждение вытекает из того, что

$$e_i = \sum_{l=0}^{n-1} f_{((i-l))} \left[\frac{1}{n} \sum_{k=0}^{n-1} \omega^{-kl} G_k \right] =$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-ik} G_k \left[\sum_{l=0}^{n-1} \omega^{(i-l)\,k} f_{((i-l))} \right] = \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-ik} G_k F_k.$$

Поскольку $\mathbf{e}$ есть обратное преобразование Фурье от $\mathbf{E}$, то мы заключаем, что $E_k = G_k F_k$.

Можно также определить двумерные преобразования Фурье, которые полезны при обработке двумерных таблиц данных, и многомерные преобразования Фурье, которые используются при обработке многомерных массивов данных. Двумерное преобразование Фурье определяется равенствами

$$V_{k',k''} = \sum_{i'=0}^{n'-1} \sum_{i''=0}^{n''-1} \omega^{i'k'} \mu^{i''k''} v_{i',i''}, \qquad \begin{matrix} k' = 0,\ \ldots,\ n'-1, \\ k'' = 0,\ \ldots,\ n''-1, \end{matrix}$$

где $\omega = e^{-j2\pi/n'}$ и $\mu = e^{-j2\pi/n''}$.

1.5. История быстрых алгоритмов обработки сигналов

Историю быстрых алгоритмов обработки сигналов принято отсчитывать с момента, когда в 1965 г. Кули и Тьюки опубликовали свой быстрый алгоритм вычисления преобразования Фурье (БПФ-алгоритм), хотя на самом деле эта история началась намного

раньше. Указанная публикация [5] появилась как раз в нужный момент и послужила катализатором применения метода цифровой обработки сигналов в новом контексте. Вскоре после опубликования этой работы Стогхем [6] заметил, что БПФ-алгоритмы могут служить удобным способом вычисления сверток. Технология цифровой обработки может непосредственно использовать БПФ, так что имеется множество приложений, и поэтому работа Кули и Тьюки стала широко известной. Лишь несколько лет спустя было осознанно, что другой БПФ-алгоритм, сильно отличающийся от алгоритма Кули—Тьюки, был разработан раньше Гудом [7] (1960) и Томасом [8] (1963). В свое время публикация БПФ-алгоритма Гуда—Томаса прошла почти незамеченной. Позже Виноград [9, 10] (1976, 1978) опубликовал свой более эффективный, хотя и более сложный, БПФ-алгоритм, который к тому же позволил значительно глубже понять, что в действительности означает процесс вычисления дискретного преобразования Фурье.

Неблагоприятным следствием популярности БПФ-алгоритма Кули—Тьюки явилось широкое распространение мнения о том, что дискретное преобразование Фурье практично применять лишь при длине блока, равной степени двух. Это привело к тому, что БПФ-алготитмы стали диктовать параметры применяемых устройств вместо того, чтобы приложения диктовали выбор подходящего алгоритма БПФ. На самом же деле хорошие БПФ-алгоритмы существуют практически для произвольной длины блока.

Различные вариации БПФ-алгоритма Кули—Тьюки появились во многих обличьях. По существу та же самая идея используется для построения многолучевой плоско-фазированной радиолокационной антенны и известна под названием матрицы Батлера [11] (1961).

Для малых длин блоков быстрые алгоритмы сверток были впервые построены Агарвалом и Кули [12] (1977) с использованием остроумных догадок, но не на основе общего метода. Общий метод построения быстрых алгоритмов свертки описал Виноград [10] (1978), доказав при этом важные теоремы несуществования лучших алгоритмов свертки для полей комплексных и вещественных чисел. Агарвал и Кули [12] также указали основанный на китайской теореме об остатках метод разбиения задачи вычисления длинных сверток на задачи вычисления коротких сверток. Их метод в сочетании с методом Винограда, применяемый для вычисления коротких сверток, дает хорошие результаты.

Самая ранняя идея того, что мы предпочитаем называть быстрыми алгоритмами, появилась намного раньше, чем БПФ-алгоритмы. Алгоритм Левинсона [13] был опубликован в 1947 г. как эффективный метод решения некоторых теплицевых систем уравнений. Несмотря на чрезвычайную важность этого алгоритма

в обработке сейсмических данных, литература, посвященная алгоритму Левинсона, многие годы не пересекалась с литературой по БПФ-алгоритмам. Как правило, в этих ранних исследованиях не делалось попыток явно разграничить алгоритм Левинсона как вычислительную процедуру и задачи фильтрации, к решению которых алгоритм применялся. Подобно этому, не всегда проводилось различие между БПФ как вычислительным средством и дискретным преобразованием Фурье, для вычисления которого используются БПФ-алгоритмы; аналогично зачастую не различались алгоритм Витерби как вычислительная процедура и задача поиска пути, к решению которой применяется алгоритм Витерби.

Задачи

1.1. Построить алгоритм 2-точечной вещественной циклической свертки

$$(s_1x + s_0) = (g_1x + g_0)\,(d_1x + d_0) \pmod{x^2 - 1},$$

содержащий два умножения и четыре сложения. Относящиеся к g_0 и g_1 вычисления не учитывать, полагая их константами, для которых все вычисления выполняются заранее раз и навсегда.

1.2. Опираясь на задачу 1.1, построить алгоритм 2-точечной комплексной свертки, содержащий только шесть вещественных умножений.

1.3. Построить алгоритм 3-точечного преобразования Фурье

$$V_k = \sum_{l=0}^{2} \omega^{lk} v_l, \qquad k = 0,\ 1,\ 2,$$

в котором имеется только два вещественных умножения.

1.4. Доказать, что не существует алгоритма умножения двух комплексных чисел, содержащего только два вещественных умножения.

1.5.а. Допустим, что имеется устройство вычисления линейной свертки двух 50-точечных последовательностей. Описать, как это устройство может быть использовано для вычисления циклической свертки двух 50-точечных последовательностей.

б. Допустим, что имеется устройство для вычисления циклической свертки двух 50-точечных последовательностей. Описать, как это устройство может быть использовано для вычисления линейной свертки двух 50-точечных последовательностей.

1.6. Доказать, что корреляцию можно вычислять как свертку, записывая одну из последовательностей в обратном порядке и, возможно, пополняя одну из последовательностей отрезком нулей.

1.7. Доказать, что любой алгоритм вычисления x^{31}, в котором используются только сложения, вычитания и умножения, должен содержать по меньшей мере семь умножений, но если допустить возможность деления, то существует алгоритм, содержащий всего шесть умножений и делений.

1.8. Один из алгоритмов вычисления произведения комплексных чисел дается равенствами

$$e = ac - bd,$$

$$f = (a + b)\,(c + d) - ac - bd.$$

Записать этот алгоритм в матричном виде, подобно тому, как это сделано в тексте главы. Каковы недостатки или преимущества этого алгоритма по сравнению с приведенным выше?

1.9. Доказать «свойство сдвига» для преобразований Фурье: если $\{v_l\} \leftrightarrow \{V_k\}$ являются парой преобразования Фурье, то парами преобразования Фурье также являются

$$\{\omega^l v_l\} \leftrightarrow \{V_{((k+1))}\} \quad \text{и} \quad \{v_{((l-1))}\} \leftrightarrow \{\omega^k V_k\}.$$

1.10. Доказать, что «циклическая корреляция» двух вещественных последовательностей **g** и **d** удовлетворяет соотношению

$$G_k D_k^* \leftrightarrow \sum_{k=0}^{1-n} g_{((l+k))} d_k,$$

где **G** и **D** соответственно преобразования Фурье последовательностей **g** и **d**.

Замечания

Хорошее изложение начала истории быстрых алгоритмов преобразования Фурье содержится в статье Кули, Левиса и Велча [1] (1967). Основы теории цифровой обработки сигналов можно найти во многих книгах; укажем две из них: Оппенгейм и Шафер [2] (1975) и Рэбинер и Голд [3] (1975).

Алгоритмы умножения комплексных чисел, содержащие три вещественных умножения, в общем стали известны с конца 1950-х, но поначалу им не придавали должного значения. Описанный нами алгоритм умножения матриц принадлежит Винограду [4] (1968).

Глава 2

ВВЕДЕНИЕ В АБСТРАКТНУЮ

Хорошие алгоритмы основаны на красивых алгебраических тождествах. Для построения таких алгоритмов необходимо знакомство с мощными структурами теории чисел и современной алгебры. Такие структуры, как множество целых чисел, кольца многочленов и поля Галуа, играют важную роль в построении алгоритмов обработки дискретных сигналов. В настоящей главе излагаются алгебраические сведения, которые необходимы для дальнейшего, но, как правило, неизвестны студентам, специализирующимся в области обработки дискретных сигналов. Сначала изучаются математические структуры групп, колец и полей. Мы увидим, что дискретное преобразование Фурье может быть определено в произвольном поле, хотя наиболее известно его определение в поле комплексных чисел. Затем будут рассмотрены известные понятия матричной алгебры и векторных пространств. Мы покажем, что их можно корректно определить для любого поля. Наконец, мы изучим кольцо целых чисел и кольца многочленов, уделяя особое внимание алгоритму Евклида и китайской теореме об остатках.

2.1. Группы

Понятие группы является математической абстракцией определенной алгебраической структуры, часто встречающейся во многих конкретных видах. Введение абстрактного понятия обусловлено тем, что легче одновременно исследовать все математические системы с общей структурой, чем изучать каждую из них по отдельности.

Определение 2.1.1. *Группой* G называется множество элементов с определенной на нем операций (обозначаемой $*$), которая удовлетворяет следующим четырем свойствам:

(1) (*Замкнутость.*) Для каждой пары элементов a и b из этого множества элемент $c = a * b$ принадлежит этому множеству.

(2) (*Ассоциативность.*) Для всех элементов a, b и c из этого множества

$$a * (b * c) = (a * b) * c.$$

*	e	g_1	g_2	g_3	g_4
e	e	g_1	g_2	g_3	g_4
g_1	g_1	g_2	g_3	g_4	e
g_2	g_2	g_3	g_4	e	g_1
g_3	g_3	g_4	e	g_1	g_2
g_4	g_4	e	g_1	g_2	g_3

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Рис. 2.1. Пример конечной группы.

(3) (*Существование единицы.*) В этом множестве существует элемент e, называемый *единичным элементом*, такой что

$$a*e = e*a = a$$

для любого элемента a рассматриваемого множества.

(4) (*Обратимость.*) Для любого a из данного множества существует (называемый *обратным* к a) некоторый элемент b из этого множества, такой что

$$a*b = b*a = e.$$

Если G содержит конечное число элементов, то она называется *конечной группой*. Число элементов в конечной группе G называется ее *порядком*. На рис. 2.1 приведен пример конечной группы. По существу здесь представлена одна и та же группа, но в разных обозначениях. В случае когда две группы описываются одной и той же структурой, хотя и в разных обозначениях, они называются *изоморфными* [1]).

Некоторые группы обладают тем дополнительным свойством, что для любых a и b из такой группы

$$a*b = b*a.$$

Это свойство называется *коммутативностью*. Группы, обладающие этим дополнительным свойством, называются *коммутативными группами*, или *абелевыми группами*. Мы всегда будем иметь дело с абелевыми группами.

В случае абелевых групп знак групповой операции обозначается плюсом и называется сложением (даже тогда, когда операция не является обычным арифметическим сложением). В этом случае единичный элемент e называется «нулем» и обозначается 0, а обратный к a элемент записывается в виде $-a$, так что

$$a + (-a) = (-a) + a = 0.$$

Иногда символ групповой операции обозначатеся точкой и называется умножением (даже тогда, когда оно не является обычным умножением). В этом случае единичный элемент называется единицей и обозначается 1, а обратный к a элемент записывается в виде a^{-1}, так что

$$a \cdot a^{-1} = a^{-1} \cdot a = 1.$$

[1]) Вообще любые две алгебраические системы с одной и той же структурой, но представленные в разных обозначениях называются *изоморфными*.

Теорема 2.1.2. *Единичный элемент в каждой группе единствен. Для каждого элемента группы обратный элемент также единствен, и* $(a^{-1})^{-1} = a$.

Доказательство. Предположим, что e и e' являются единичными элементами. Тогда $e = e*e' = e'$. Далее предположим, что b и b' являются элементами, обратными к a; тогда

$$b = b*(a*b') = (b*a)*b' = b'.$$

Наконец, $a^{-1}*a = a*a^{-1} = e$, так что a является обратным к a^{-1}. Но в силу единственности обратного элемента $(a^{-1})^{-1} = a$. □

Многие общеизвестные группы содержат бесконечное число элементов. Примерами являются: множество целых чисел с операцией сложения; множество положительных рациональных чисел с операцией умножения [1]); множество вещественных (2×2)-матриц с операций сложения. Многие другие группы содержат только конечное число элементов. Конечные группы могут быть устроены весьма хитро.

Если групповая операция применяется два или более раз к одному и тому же элементу, то можно использовать степенное обозначение. Таким образом, $a^2 = a*a$ и

$$a^k = a*a* \ \ldots \ *a,$$

где справа стоят k копий элемента a.

Циклической называется группа, в которой каждый элемент может быть записан в виде степени некоторого фиксированного элемента, называемого *образующей группы*. Каждая конечная циклическая группа имеет вид

$$G = \{a^0,\ a^1,\ a^2,\ \ldots,\ a^{q-1}\},$$

где q — порядок G, a — образующая, а a^0 — единичный элемент, а обратный к a^i элемент равен a^{q-i}. Чтобы можно было на самом деле задать группу таким способом, необходимо, чтобы выполнялось равенство $a^q = a^0$. Это вытекает из того, что в противном случае, если $a^q = a^i$ при $i \neq 0$, то $a^{q-1} = a^{i-1}$ и в противоречие с определением мы получаем меньше, чем q, элементов.

[1]) Этот пример дает удобный повод, чтобы сделать предостережение по поводу терминологии. В случае произвольной абелевой группы групповая операция обычно называется сложением, но не обязательно является обычным сложением. В данном примере она является обычным умножением.

Важнейшей циклической группой с q элементами является группа, обозначаемая $Z/(q)$, или иногда Z_q, и задаваемая множеством

$$Z/(q) = \{0,\ 1,\ 2,\ \ldots,\ q-1\}$$

со сложением по модулю q в качестве групповой операции. Например,

$$Z/(6) = \{0,\ 1,\ 2,\ \ldots,\ 5\}$$

и $3 + 4 = 1$.

Группа $Z/(q)$ может быть выбрана в качестве стандартного прототипа циклической группы с q элементами. На самом деле имеется только одна группа с q элементами; все остальные являются ее изоморфными копиями, различающимися обозначениями, но не структурой. Любую другую циклическую группу G с q элементами можно отобразить в $Z/(q)$ с заменой групповой операции в G сложением по модулю q. Любые свойства структуры в G справедливы в $Z/(q)$, и наоборот.

По заданным двум группам G' и G'' можно построить новую группу G, которая называется *произведением* групп G' и G'' и обозначается $G = G' \times G''$. Элементами G служат пары элементов (a', a''), первый из которых принадлежит G', а второй — G''. Групповая операция в произведении групп G определяется равенством

$$(a',\ a'') * (b',\ b'') = (a' * b',\ a'' * b'').$$

В этой формуле звездочка используется три раза в трех разных смыслах. В левой части равенства она обозначает групповую операцию в G, а в правой части — операции в группах G' и G'' соответственно.

Например, $Z_2 \times Z_3$ задается множеством

$$Z_2 \times Z_3 = \{(0,\ 0),\ (0,\ 1),\ (0,\ 2),\ (1,\ 0),\ (1,\ 1),\ (1,\ 2)\}.$$

Типичным элементом таблицы сложений в $Z_2 \times Z_3$ является

$$(1,\ 2) + (0,\ 2) = (1,\ 1).$$

Отметим, что группа $Z_2 \times Z_3$ сама является циклической с образующим элементом $(1,\ 1)$. Следовательно, $Z_2 \times Z_3$ изоморфна группе Z_6. Причина этого роется в том, что числа 2 и 3 не имеют общих целочисленных делителей.

Пусть G группа и H — подмножество в G. Тогда H называется *подгруппой* группы G, если H само является группой относительно ограничения групповой операции на H. В качестве примера, в множеств целых чисел (положительных, отрицательных и нуля) с операцией сложения множество четных чисел, так же как и множество чисел, кратных 3, образует подгруппу.

Один из способов построения подгруппы H конечной группы G состоит в том, чтобы, выбрав некоторый элемент h из G, формировать элементы множества H в виде последовательных степеней

этого элемента: h, h^2, h^3, h^4, ... Так как группа G конечна, то элемент последовательности обязательно начнет повторяться. Первым должен повториться сам элемент h, а непосредственно предшествующий ему элемент является единичным элементом группы, так как рассматриваемая конструкция дает циклическую группу. Множество H называется *циклической подгруппой*, порожденной элементом h. Число q элементов в подгруппе H удовлетворяет равенству $h^q = 1$ и называется *порядком* элемента h. Множество элементов h, h^2, h^3, ..., $h^q = 1$ называется *циклом* в группе G.

Для того чтобы доказать, что непустое подмножество H является подгруппой в G, достаточно только проверить, что вместе с элементами a и b из H множеству H принадлежит элемент $a * b$, и что обратный к произвольному элементу a из H также принадлежит H. Остальные требуемые групповые свойства наследуются из группы G. Если группа конечна, то, как мы увидим позже при рассмотрении циклических подгрупп, даже свойство обратимости выполняется автоматически.

Для заданных конечной группы G и подгруппы H имеется важная конструкция, иллюстрирующая определенную взаимосвязь между G и H и известная под названием *разложения группы G на смежные классы* по подгруппе H. Обозначим через h_1, h_2, h_3, ... элементы из H, причем через h_1 обозначим единичный элемент. Построим следующую таблицу: первая строка состоит из элементов подгруппы H, причем первым слева выписан единичный элемент и каждый элемент из H выписан в строке один и только один раз. Выберем произвольный элемент группы G, не содержащийся в первой строке. Назовем его g_2 и используем в качестве первого элемента второй строки. Остальные элементы второй строки теперь получаются умножением слева элементов подгруппы на этот первый элемент. Аналогично, строя стретью, четвертую и пятую строки, каждый раз выбираем не использованный на предыдущих шагах элемент группы G в качестве элементов первого столбца. Конец наступает, когда после некоторого шага оказывается, что все элементы группы записаны в некотором месте таблицы. Процесс оканчивается в силу конечности G. Таблица имеет следующий вид:

$$\begin{array}{llll} h_1 = 1 & h_2 & \ldots & h_n \\ g_2 * h_1 = g_2 & g_2 * h_2 & \ldots & g_2 * h_n \\ g_3 * h_1 = g_3 & g_3 * h_2 & \ldots & g_3 * h_n \\ & & \ldots & \\ g_m * h_1 = g_m & g_m * h_2 & \ldots & g_m * h_n \end{array}$$

Первый элемент в каждой строке называется *лидером смежного класса*. Каждая строка таблицы называется *левым смежным*

классом, а в случае абелевой группы просто *смежным классом*. Если при построении разложения группы на смежные классы использовать вместо левого правое умножение на элементы группы H, то строки называются *правыми смежными классами*. В силу указанных правил построения разложение группы на смежные классы всегда является прямоугольной таблицей, все строки которой полностью заполнены. Докажем теперь, что в этой таблице каждый элемент группы встречается точно один раз.

Теорема 2.1.3. *В разложении группы на смежные классы каждый элемент группы встречается один и только один раз.*

Доказательство. Каждый элемент появится по крайней мере один раз, так как в противном случае процесс не остановится. Покажем теперь, что каждый элемент не может появиться дважды в одной и той же строке, а затем докажем, что один и тот же элемент не может появиться в двух разных строках таблицы.

Предположим, что два элемента в одной и той же строке, скажем $g_i * h_j$ и $g_i * h_k$, равны. Тогда умножение [1]) каждого из них на g_i^{-1} дает равенство $h_j = h_k$. Это противоречит тому, что каждый элемент подгруппы выписан в первой строке только один раз.

Предположим, что два элемента из различных строк, скажем $g_l * h_j$ и $g_k * h_i$, равны, и что $k < i$. Умножение справа на h_j^{-1} приводит к равенству $g_l = g_k * h_i * h_j^{-1}$. Тогда g_l порождает k-й смежный класс, так как элемент $h_i * h_j^{-1}$ принадлежит подгруппе. Это противоречит выбранному выше правилу выбора лидеров смежных классов. □

Следствие 2.1.4. *Если H — подгруппа группы G, то число элементов в H делит число элементов в G. Таким образом,*

(Порядок H) (Число смежных классов G по H) = (Порядок G).

Доказательство. Следует непосредственно из прямоугольности таблицы разложения на смежные классы. □

Теорема 2.1.5. *Порядок конечной группы делится на порядок любого из ее элементов. Таким образом, $a^q = 1$ для любого элемента a группы G с q элементами.*

Доказательство. Группа содержит циклическую подгруппу, порожденную любым из ее элементов, и, таким образом, доказательство теоремы вытекает из следствия 2.1.4. □

2.2. Кольца

Следующей необходимой нам алгебраической структурой является кольцо. Кольцо представляет собой абстрактное множество, которое является абелевой группой и наделено дополнительной операцией.

[1]) Слева. — *Прим. перев.*

Определение 2.2.1. *Кольцом* R называется множество с двумя определенными на нем операциями: первая называется *сложением* (обозначается плюсом); *вторая* называется *умножением* (обозначается записью сомножителей рядом), причем выполняются следующие аксиомы:

1. Относительно сложения $(+)$ R является абелевой группой.
2. (*Замкнутость.*) Произведение ab принадлежит R для любых a и b из R.
3. (*Закон ассоциативности.*)
$$a\,(bc) = (ab)\,c.$$
4. (*Закон дистрибутивности.*)
$$a\,(b + c) = ab + ac,$$
$$(b + c)\,a = ba + ca.$$

Сложение в кольце всегда коммутативно, а умножение не обязательно должно быть коммутативным. *Коммутативное кольцо* — это кольцо с коммутативным умножением: $ab = ba$ для всех a и b из R. Дистрибутивный закон в определении кольца связывает операции сложения и умножения.

Важнейшими примерами колец являются кольцо Z целых чисел и кольцо $Z/(q)$ целых чисел по модулю q. Мы уже видели, что они являются группами по сложению. Так как на этих множествах имеется также умножение с необходимыми свойствами, то они являются кольцами.

Несколько хорошо известных читателю по знакомым кольцам свойств могут быть выведены из аксиом следующим образом.

Теорема 2.2.2. *Для произвольных элементов a и b из кольца R выполняются равенства*

(i) $a0 = 0a = 0$.
(ii) $a\,(-b) = (-a)\,b = -ab$.

Доказательство.
(i) $a0 = a\,(0 + 0) = a0 + a0$. Прибавляя к обеим частям равенства $-a0$, получим $0 = a0$. Аналогично доказывается вторая половина п. (i).
(ii) $0 = a0 = a\,(b - b) = ab + a\,(-b)$. Следовательно,
$$a\,(-b) = -(ab).$$
Аналогично доказывается вторая половина п. (ii). □

Относительно операции сложения кольцо содержит единицу, называемую «нулем». Относительно операции умножения кольцо не обязательно имеет единицу, но если единица есть, то она единственна. Кольцо, обладающее единицей относительно умножения, называется *кольцом с единицей*. Эта единица называется «единицей»

и обозначается символом 1. Тогда для всех a из R имеет место

$$1a = a1 = a.$$

Относительно операции сложения каждый элемент кольца имеет обратный. Относительно операции умножения обратные к данному элементу не обязательно существуют, но в кольце с единицей обратные могут существовать. Это означает, что для заданного элемента a может существовать элемент b, такой что $ab = 1$. Если это так, то b называется *правым обратным* к a. Аналогично, если существует элемент c, такой, что $ca = 1$, то c называется *левым обратным* к a.

Теорема 2.2.3. *В кольце с единицей:*

(i) *Единица единственна.*

(ii) *Если элемент a имеет и левый обратный b, и правый обратный c, то $b = c$. В этом случае элемент a называется обратимым, причем обратный элемент единствен (и обозначается через a^{-1}).*

(iii) $(a^{-1})^{-1} = a$.

Доказательство. Аргументация аналогична использованной в доказательстве теоремы 2.1.2. □

Если кольцо содержит единицу, то можно произвольное число раз сложить ее саму с собой или вычесть из себя самой, получив, таким образом, бесконечную в обе стороны последовательность

$$\ldots, -(1+1+1), -(1+1), -1, 0, 1,$$
$$1+1, 1+1+1, \ldots .$$

Эти элементы называются *целыми* кольца и иногда обозначаются просто как $0, \pm 1, \pm 2, \pm 3, \pm 4, \ldots$. Кольцо может содержать как конечное, так и бесконечное множество целых. Число целых в кольце с единицей называется его *характеристикой*. Если характеристика кольца равна конечному целому числу q, то целые этого кольца могут быть записаны в виде множества

$$\{1, 1+1, 1+1+1, \ldots\},$$

или в более простом, используемом для обычных целых чисел, обозначении

$$\{1, 2, 3, \ldots, q-1, 0\}.$$

Это подмножество является подгруппой аддитивной группы кольца; на самом деле это циклическая подгруппа, порождаемая элементом 1. Следовательно, если характеристика кольца — конечное целое число, то сложение в кольце является сложением по модулю q. Если характеристика бесконечна, то целые кольца складываются как целые числа. Следовательно, каждое кольцо

с единицей содержит подмножество, которое ведет себя относительно сложения либо как Z, либо как $Z/(q)$. В действительности оно ведет себя так же и относительно умножения, так как если α и β — конечные суммы единицы кольца, то

$$\alpha \cdot \beta = \beta + \beta + \ldots + \beta,$$

где справа стоит α копий элемента β. Так как сложение целых в R ведет себя подобно сложению в Z или в $Z/(q)$, то так же ведет себя и умножение в R.

В пределах кольца R каждый элемент α может быть возведен в целочисленную степень; α^m просто означает произведение m копий элемента α. Если кольцо содержит единицу и число целых кольца является простым, то иногда для упрощения степенных сумм оказывается полезной следующая теорема.

Теорема 2.2.4. *Пусть p простое, и пусть R — кольцо с p целыми. Тогда для любого положительного целого числа m и любых элементов α и β из R*

$$(\alpha \pm \beta)^{p^m} = \alpha^{p^m} \pm \beta^{p^m},$$

и, по непосредственному обобщению,

$$\left(\sum_i \alpha_i\right)^{p^m} = \sum_i \left(\alpha_i^{p^m}\right)$$

для любого множества элементов α_i из R.

Доказательство. Согласно биномиальному разложению,

$$(\alpha \pm \beta)^p = \sum_{i=0}^{p} \binom{p}{i} \alpha^i (\pm\beta)^{p-i},$$

где $\binom{p}{i}$ интерпретируется в R как сумма такого числа копий единицы кольца. Напомним, что в кольце с p целыми арифметика целых совпадает с арифметикой по модулю p. Следовательно, можно записать

$$(\alpha \pm \beta)^p = \sum_{i=0}^{p} \left(\left(\left(\binom{p}{i}\right)\right)\right) \alpha^i (\pm\beta)^{p-i},$$

где двойные скобки вокруг $\binom{p}{i}$ обозначают вычисление по модулю p. Теперь заметим, что

$$\binom{p}{i} = \frac{p!}{i!\,(p-i)!} = \frac{p\,(p-1)!}{i!\,(p-i)!}$$

представляет собой целое число и p простое. Следовательно, знаменатель делит $(p-1)!$ в области целых чисел, и $\binom{p}{i}$ кратно p.

Таким образом, $\binom{p}{i} = 0 \pmod p$ для всех $i = 1, 2, \ldots, p-1$. Итак, $(\alpha \pm \beta)^p = \alpha^p + (\pm\beta)^p$. Наконец, либо $p = 2$ и $-\beta = \beta$, так что $(\pm\beta)^2 = \pm\beta^2$, либо p нечетно и $(\pm\beta)^p = \pm\beta^p$. Таким образом,

$$(\alpha \pm \beta)^p = \alpha^p \pm \beta^p$$

и для $m = 1$ теорема доказана.

Возведем теперь последнее равенство в p-ю степень,

$$((\alpha \pm \beta)^p)^p = (\alpha^p \pm \beta^p)^p,$$

и опять воспользуемся утверждением теоремы при $m = 1$:

$$(\alpha \pm \beta)^{p^2} = \alpha^{p^2} \pm \beta^{p^2}.$$

Повторяя это $m - 1$ раз, получаем

$$(\alpha \pm \beta)^{p^m} = \alpha^{p^m} \pm \beta^{p^m},$$

что завершает доказательство теоремы. □

Если в кольце с единицей элемент имеет обратный, то он называется *обратимым* [1]). Множество всех обратимых элементов кольца замкнуто относительно умножения, так как, если a и b обратимы, то $c = ab$ имеет обратный элемент, равный $c^{-1} = b^{-1}a^{-1}$.

Теорема 2.2.5.

(i) *Относительно умножения в кольце множество обратимых элементов кольца образует группу.*

(ii) *Если $c = ab$ и c — обратимый элемент кольца, то a имеет правый обратный, а b — левый обратный.*

(iii) *Если $c = ab$ и a не имеет правого обратного или b не имеет левого обратного, то c не является обратимым элементом.*

Доказательство. Упражнение. □

Многие примеры колец известны. Например:

1. Множество всех вещественных чисел относительно обычных сложения и умножения образует коммутативное кольцо с единицей. Каждый ненулевой элемент кольца является обратимым.

2. Множество $\mathbb{Z}$ всех целых чисел (положительные, отрицательные и нуль) относительно обычных сложения и умножения образует коммутативное кольцо с единицей. Единственными обратимыми элементами кольца служат ± 1.

3. Множество всех $(n \times n)$-матриц с вещественными элементами относительно матричного сложения и матричного умножения образует некоммутативное кольцо с единицей. Единицей служит единичная $(n \times n)$-матрица. Обратимыми элементами кольца являются все невырожденные матрицы.

4. Множество всех $(n \times n)$-матриц, элементами которых являются целые числа, относительно матричного сложения и матричного умножения образует некоммутативное кольцо с единицей.

5. Множество всех многочленов от x с вещественными коэффициентами относительно сложения и умножения многочленов образует коммутативное кольцо с единицей. Единицей кольца является многочлен нулевой степени $p(x) = 1$.

[1]) Используемые для единицы и обратимых элементов кольца английские термины «unity» и «unit» переводятся на русский язык одним словом «единица», так что часто в литературе обратимые элементы называются единицами кольца; мы предпочитаем зафиксировать термин «единица» в определенном выше смысле.— *Прим. перев.*

2.3. Поля

Грубо говоря, абелевой группой является множество, в котором можно «складывать» и «вычитать», а кольцо — множество, в котором можно «складывать», «вычитать» и «умножать». Более сильной математической структурой, называемой полем, является множество, в котором можно «складывать», «вычитать», «умножать» и «делить».

Определение 2.3.1. *Полем* называется множество с двумя операциями — сложением и умножением, — которые удовлетворяют следующим аксиомам:

1. Множество образует абелеву группу по сложению.

2. Поле замкнуто относительно умножения и множество ненулевых элементов образует абелеву группу по умножению.

3. Дистрибутивный закон

$$(a + b)\, c = ac + bc$$

выполняется для любых a, b и c из поля.

Единичный элемент относительно сложения принято называть нулем и обозначать через 0, аддитивный обратный к элементу a обозначать $-a$, единичный элемент относительно умножения называть единицей и обозначать 1, мультипликативный обратный к элементу a обозначать a^{-1}. Под вычитанием $(a - b)$ понимается $a + (-b)$; под делением (a/b) понимается $b^{-1}a$.

Следующие примеры полей широко известны:

1. $\mathbb{R}$: множество вещественных чисел.

2. $\mathbb{C}$: множество комплексных чисел.

3. $\mathbb{Q}$: множество рациональных чисел.

Все эти поля содержат бесконечное число элементов. Имеется много других не столь широко известных полей с бесконечным числом элементов. Одно из таких полей описывается очень просто и известно как поле $\mathbb{Q}(j)$ комплексных рациональных чисел. Оно дается определением

$$\mathbb{Q}(j) = \{a + jb\},$$

где a и b — рациональные числа, а сложение и умножение определяются так же, как для комплексных чисел. При таком определении множество $\mathbb{Q}(j)$ удовлетворяет определению 2.3.1 и, следовательно, является полем.

Имеются также поля с конечным числом элементов, и мы будем ими также пользоваться. Поле с q элементами, если оно существует, называется *конечным полем* или *полем Галуа* и обозначается $GF(q)$.

Что представляет собой наименьшее поле? Оно обязано содержать нулевой элемент и единичный элемент. На самом деле этого уже достаточно для задания поля, если сложение и умножение определить таблицами

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Это поле известно как $GF(2)$. Никаких других полей с двумя элементами не существует (за исключением, конечно, изоморфных копий поля $GF(2)$).

Конечные поля можно описывать с помощью таблиц сложения и умножения. Вычитание и деление однозначно определяются таблицами сложения и умножения. Позже изучим конечные поля детально. Сейчас мы приведем еще три примера.

Поле $GF(3) = \{0, 1, 2\}$ с операциями

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Поле $GF(4) = \{0, 1, 2, 3\}$ с операциями

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Заметьте, что в $GF(4)$ умножение не есть умножение по модулю 4, а сложение не есть сложение по модулю 4.

Поле $GF(5) = \{0, 1, 2, 3, 4\}$ с операциями

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Это примеры очень малых полей. Мы найдем применения и для значительно больших полей, таких как $GF(2^{16}+1)$.

Для произвольного поля, как бесконечного, так и конечного, применимы почти все известные алгоритмы вычислений. Это происходит потому, что большинство процедур, используемых в полях вещественных и комплексных чисел, зависит только от даваемой определением 2.3.1 формальной структуры поля и не зависит от частных характеристик конкретного поля. В произвольном поле F имеется даже преобразование Фурье:

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \qquad k = 0, \ldots, n-1,$$

где ω — корень степени n из единицы в поле F, а $\mathbf{v}$ и $\mathbf{V}$ — векторы длины n над полем F. Преобразование Фурье длины n в поле F существует тогда и только тогда, когда поле содержит корень степени n из единицы. Если преобразование Фурье существует, то оно ведет себя ожидаемым образом. В частности, должно существовать и обратное преобразование Фурье, и должна выполняться теорема о свертке, так как если просмотреть доказательства этих свойств, то можно увидеть, что о поле F не делается никаких предположений, за исключением того, что оно является даваемой определением 2.3.1 формальной структурой.

Аналогично, двумерное преобразование Фурье в поле F

$$V_{k',k''} = \sum_{i'=0}^{n'-1} \sum_{i''=0}^{n''-1} \omega^{i'k'} \mu^{i''k''} v_{i',i''}, \qquad \begin{matrix} k' = 0, \ldots, n'-1, \\ k'' = 0, \ldots, n''-1, \end{matrix}$$

существует, если в поле F имеется элемент ω порядка n' и элемент μ порядка n''. Если таких элементов в поле нет, то указанное преобразование не существует.

Например, в поле $GF(5)$ порядок элемента 2 равен 4. Следовательно, в поле $GF(5)$ существует 4-точечное преобразование Фурье

$$V_k = \sum_{i=0}^{3} 2^{ik} v_i, \qquad k = 0, 1, 2, 3,$$

и двумерное (4×4)-преобразование Фурье

$$V_{k',k''} = \sum_{i'=0}^{3} \sum_{i''=0}^{3} 2^{i'k''} 2^{i''k''} v_{i',i''}.$$

Компоненты векторов $\mathbf{v}$ и $\mathbf{V}$ принадлежат полю $GF(5)$, и вся арифметика является арифметикой поля $GF(5)$. Если

$$\mathbf{v} = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix},$$

то одномерное преобразование Фурье вектора $\mathbf{v}$ равно

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \\ 1 & 3 & 4 & 2 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}.$$

Определение 2.3.2. Пусть F — поле. Подмножество в F называется *подполем*, если оно является полем относительно наследуемых операций сложения и умножения; исходное поле F в этом случае называется *расширением* подполя.

Поле рациональных чисел является подполем поля вещественных чисел, которое в свою очередь является подполем поля комплексных чисел. Поле комплексных рациональных чисел не является подполем поля вещественных чисел, но является подполем поля комплексных чисел. Легко видеть, что конечное поле GF (2) является подполем поля GF (4), так как элементы 0 и 1 складываются и умножаются в поле GF (4) так же, как в поле GF (2). Однако GF (2) не является подполем ни поля GF (3), ни поля GF (5).

Чтобы доказать, что некоторое подмножество поля является подполем, необходимо доказать только то, что оно содержит ненулевой элемент и замкнуто относительно сложения и умножения. Все остальные необходимые свойства наследуются из поля F. Обращение по сложению или умножение на элемент β входят в порождаемую элементом β циклическую группу соответственно относительно операций сложения или умножения.

Каждое поле содержит в качестве подмножества множество своих целых

$$\{\ldots, -(1+1+1), -(1+1), -1, 0, 1, 1+1, 1+1+1, \ldots\}.$$

Целые поля обычно записываются просто как $0, \pm 1, \pm 2, \pm 3, \ldots$. Поле может содержать только конечное число целых, и в этом случае такое число называется *характеристикой* поля. (Оно всегда простое.) Все элементы полей GF (3) и GF (5) являются целыми, так что характеристики полей равны соответственно 3 и 5. Так как в поле GF (4) целыми являются только 0 и 1, то характеристика этого поля равна 2. Характеристика и поля вещественных чисел, и поля комплексных чисел равна ∞. Каждое поле характеристики ∞ содержит поле рациональных чисел (или его изоморфную копию).

Если поле имеет конечную характеристику p, то целые этого поля образуют циклическую группу по сложению; следовательно, сложение целых поля представляет собой сложение по модулю p, где p — число целых поля. Сумму n копий единицы поля будем записывать в виде $((n))$, где двойные скобки обозначают вычисление по модулю p.

Конечное поле задает группу двумя путями: все элементы поля образуют группу относительно сложения и все ненулевые элементы поля образуют группу относительно умножения. На самом деле по умножению ненулевые элементы поля образуют циклическую группу. Доказать это утверждение трудно, и мы отложим доказательство до гл. 5, но пользоваться цикличностью этой структуры будем раньше. Так как по умножению множество ненулевых элементов группы образует циклическую группу, то оно порождается одним элементом.

Определение 2.3.2. *Примитивным* элементом поля Галуа GF (q) называется элемент порядка ($q-1$) относительно умножения. Он порождает мультипликативную группу поля.

Например, в поле GF (5) степени элемента 3 равны: $3^1 = 3$, $3^2 = 4$, $3^3 = 2$, $3^4 = 1$. Следовательно, 3 — примитивный элемент поля GF (5).

2.4. Векторные пространства

Для заданного поля F n-последовательность $(v_0, v_1, \ldots, v_{n-1})$ элементов поля называется *вектором* длины n над полем F. Множество всех таких векторов длины n вместе с двумя заданными на нем операциями — *сложением векторов* и *умножением на скаляр* — называется *векторным пространством* над полем F. При рассмотрении векторных пространств элементы поля F, над которыми строются векторы, называются *скалярами*. Умножение на скаляр — это операция, связывающая вектор $\mathbf{v}$ с элементом поля c по правилу

$$c(v_0, v_1, \ldots, v_{n-1}) = (cv_0, cv_1, \ldots, cv_{n-1}).$$

Векторное сложение является операцией сложения двух векторов, $\mathbf{v} = \mathbf{v}' + \mathbf{v}''$, выполняемого по следующему правилу:

$$(v_0', v_1', \ldots, v_{n-1}') + (v_0'', v_1'', \ldots, v_{n-1}'') = (v_0' + v_0'', v_1' + v_1'', \ldots, v_{n-1}' + v_{n-1}'').$$

Векторное пространство n-последовательностей представляет собой один пример векторных пространств. Векторное пространство над полем F можно определить абстрактно как множество V элементов (именуемых векторами) с двумя заданными на нем операциями. Первая операция определена на парах элементов из V, называется *сложением векторов* и обозначается плюсом. Вторая операция определена на элементе из V и элементе из F, дает в результате элемент из V и называется *умножением на скаляр*; для обозначения такой операции эти элементы записываются

рядом. Для того чтобы V было векторным пространством, эти операции должны удовлетворять следующим аксиомам.

1. V представляет собой абелеву группу относительно операции сложения векторов.

2. **(Дистрибутивный закон.)** Для любых векторов $\mathbf{v}_1$, $\mathbf{v}_2$ и любого скаляра c

$$c(\mathbf{v}_1 + \mathbf{v}_2) = c\mathbf{v}_1 + c\mathbf{v}_2.$$

3. **(Дистрибутивный закон.)** Для любого вектора $\mathbf{v}$ и любых скаляров c_1 и c_2 выполняется $1 \cdot \mathbf{v} = \mathbf{v}$ и

$$(c_1 + c_2)\,\mathbf{v} = c_1\mathbf{v} + c_2\mathbf{v}.$$

4. **(Ассоциативный закон.)** Для любого вектора $\mathbf{v}$ и любых скаляров c_1 и c_2

$$(c_1c_2)\,\mathbf{v} = c_1\,(c_2\mathbf{v}).$$

Нулевой элемент из V называется *началом координат* и обозначается через $\mathbf{0}$. Отметим, что мы использовали символ $+$ двумя различными способами: для векторного сложения и для сложения в поле. Отметим также, что мы использовали символ 0 для обозначения нулевого элемента поля и символ $\mathbf{0}$ для обозначения начала координат векторного пространства. Это неопределенность практически не приводит к недоразумениям.

Подмножество векторного пространства называется *векторным подпространством*, если оно само является векторным пространством относительно исходных операций сложения векторов и умножения вектора на скаляр. Относительно сложения векторов векторное пространство является группой, а векторное продпространство — подгруппой. Для того чтобы проверить, является ли непустое подмножество векторного пространства подпространством, необходимо проверить только замкнутость подмножества относительно сложения векторов и умножения векторов на скаляр. Замкнутость относительно умножения на скаляр гарантирует принадлежность подмножеству нулевого вектора. Все остальные требуемые свойства наследуются из исходного пространства.

Векторное пространство n-последовательностей над F обозначается через F^n. В этом пространстве сложение векторов и умножение вектора на скаляр определены покомпонентно. В F^n имеется еще одна операция, называемая *покомпонентным произведением* двух векторов. Если $\mathbf{u} = (a_0, a_1, \ldots, a_{n-1})$ и $\mathbf{v} = (b_0, b_1, \ldots, b_{n-1})$, то покомпонентное произведение определяется равенством $\mathbf{uv} = (a_0b_0, a_1b_1, \ldots, a_{n-1}b_{n-1})$. Оно представляет собой вектор, компоненты которого получаются в результате перемножения компонент векторов $\mathbf{u}$ и $\mathbf{v}$.

Скалярным произведением двух n-последовательностей из F_n называется скаляр, определяемый равенством

$$\mathbf{u} \cdot \mathbf{v} = (a_0, \ldots, a_{n-1}) \cdot (b_0, \ldots, b_{n-1}) = a_0b_0 + \ldots + a_{n-1}b_{n-1}.$$

Немедленно проверяется, что $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$, $(c\mathbf{u}) \cdot \mathbf{v} = c\,(\mathbf{u} \cdot \mathbf{v})$ и что также $\mathbf{w} \cdot (\mathbf{u} + \mathbf{v}) = (\mathbf{w} \cdot \mathbf{u}) + (\mathbf{w} \cdot \mathbf{v})$. Если скалярное произведение двух векторов равно нулю, то векторы называются *ортогональными*. Над некоторыми полями возможна ортогональность ненулевых векторов самим себе, но это невозможно над полем вещественных чисел. Вектор, ортогональный к каждому вектору из данного множества, называется ортогональным к этому множеству.

Теорема 2.4.1. *Пусть V — векторное пространство n-последовательностей над полем F, содержащее подпространство W. Множество векторов, ортогональных к W, само образует подпространство.*

Доказательство. Пусть U обозначает множество всех векторов, ортогональных к W. U не пусто, так как $\mathbf{0}$ принадлежит U. Пусть $\mathbf{w}$ — произвольный вектор из W, а $\mathbf{u}_1$ и $\mathbf{u}_2$ — некоторые векторы из U. Тогда $\mathbf{w} \cdot \mathbf{u}_1 = \mathbf{w} \cdot \mathbf{u}_2 = \mathbf{0}$ и

$$\mathbf{w} \cdot \mathbf{u}_1 + \mathbf{w} \cdot \mathbf{u}_2 = \mathbf{0} = \mathbf{w} \cdot (\mathbf{u}_1 + \mathbf{u}_2),$$

так что $\mathbf{u}_1 + \mathbf{u}_2$ также вектор из U. Далее, $\mathbf{w} \cdot (c\mathbf{u}_1) = c\,(\mathbf{w} \cdot \mathbf{u}_1) = \mathbf{0}$, так что $c\mathbf{u}_1$ также вектор из U. Следовательно, U является подпространством. □

Множество всех векторов, ортогональных к подпространству W, называется *ортогональным дополнением* подпространства W и обозначается через $W^\perp$. В векторном пространстве R^n всех n-последовательностей над полем вещественных чисел пересечение подпространств W и $W^\perp$ содержит только нулевой вектор; но в векторном пространстве над $GF(q)$ подпространство $W^\perp$ может иметь нетривиальное пересечение с W или может даже лежать в W, содержать W и совпадать с W. Например, подпространство из двух векторов (00) и (11) пространства GF^2 (2) является своим ортогональным дополнением.

В векторном пространстве V сумма вида

$$\mathbf{u} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_k\mathbf{v}_k,$$

где a_i — скаляры, называется *линейной комбинацией* векторов $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_k$. Говорят, что множество векторов порождает линейное пространство, если каждый вектор этого пространства представим в виде некой линейной комбинации векторов из этого множества. Тогда говорят также, что такое линейное пространство натянуто на это множество векторов. Линейное пространство, которое натянуто на конечное множество векторов, называется *конечномерным векторным пространством*. Число векторов в наименьшем множестве, порождающем некое пространство, называется *размерностью* этого пространства. Пространство n-последовательностей над F дает пример конечномерного векторного пространства размерности n.

Множество векторов $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ называется *линейно зависимым*, если существует множество скаляров $\{a_1, \ldots, a_k\}$, не все из которых равны, таких что

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_k\mathbf{v}_k = \mathbf{0}.$$

Множество векторов, не являющееся линейно зависимым, называется *линейно независимым*. Ни один вектор из линейно независимого множества не может быть представлен как линейная комбинация остальных. Заметим, что нулевой вектор $\mathbf{0}$ не может принадлежать линейно независимому множеству; каждое множество, содержащее $\mathbf{0}$, является линейно зависимым. Множество из k линейно независимых векторов, порождающих линейное пространство, называется *базисом* этого пространства.

2.5. Матричная алгебра

Методы матричной алгебры обычно изучаются только для полей вещественных и комплексных чисел, хотя большинство операций справедливо в произвольном поле (а иногда даже в произвольном кольце).

Определение 2.5.1. $(n\times m)$-*матрицей* $\mathbf{A}$ над полем F называется прямоугольная таблица, состоящая из n строк и m столбцов и содержащая nm элементов из поля F:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ & \ldots & \ldots & \\ a_{n1} & a_{n2} & \ldots & a_{nm} \end{bmatrix} = [a_{ij}].$$

Если $n = m$, то матрица $\mathbf{A}$ называется *квадратной*. Две $(n\times m)$-матрицы $\mathbf{A}$ и $\mathbf{B}$ над полем F можно складывать по правилу

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \ldots & a_{1m}+b_{1m} \\ & \ldots & & \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \ldots & a_{nm}+b_{nm} \end{bmatrix}.$$

Всякую $(n\times m)$-матрицу $\mathbf{A}$ можно умножить на элемент β поля по правилу

$$\beta = \mathbf{A}\begin{bmatrix} \beta a_{11} & \beta a_{12} & \ldots & \beta a_{1m} \\ & \ldots & & \\ \beta a_{n1} & \beta a_{n2} & \ldots & \beta a_{nm} \end{bmatrix}.$$

Всякую $(l\times n)$-матрицу $\mathbf{A}$ можно умножить на $(n\times m)$-матрицу $\mathbf{B}$, получив в результате $(l\times m)$-матрицу $\mathbf{C}$ по правилу

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}, \qquad \begin{matrix} i = 1, \ldots, l, \\ j = 1, \ldots, m. \end{matrix}$$

Множество элементов a_{ij}, для которых номер строки совпадает с номером столбца, называется *главной диагональю*. $(n\times n)$-матрица, все элементы главной диагонали которой равны единице, а остальные элементы равны нулю, называется *единичной матрицей* размера n и обозначается через $\mathbf{I}$. Матрица, все элементы *побочной диагонали* (элементы, для индексов которых $j = n + 1 - i$) которой равны единице, а остальные элементы равны нулю, называется *обменной матрицей* и обозначается через $\mathbf{J}$. Отметим, что $\mathbf{J}^2 = \mathbf{I}$. Примерами единичной и обменной матриц являются следующим (3×3)-матрицы:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{J} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Относительно введенного определения произведения и суммы матриц, как легко проверить, множество квадратных $(n\times n)$-матриц образует кольцо. Это кольцо некоммутативно, но обладает единицей, а именно единичной $(n\times n)$-матрицей.

Транспонированной к $(n\times m)$-матрице $\mathbf{A}$ называется $(m\times n)$-матрица $\mathbf{A}^T$, такая что $a_{ij}^T = a_{ji}$. Таким образом, строками матрицы $\mathbf{A}^T$ служат столбцы матрицы $\mathbf{A}$, а столбцами матрицы $\mathbf{A}^T$ служат строки матрицы $\mathbf{A}$. Легко проверить, что если $\mathbf{C} = \mathbf{AB}$, то $\mathbf{C}^T = \mathbf{B}^T\mathbf{A}^T$.

Обратной к квадратной матрице $\mathbf{A}$, если таковая существует, называется квадратная матрица $\mathbf{A}^{-1}$, такая что $\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$. Как нетрудно проверить, множество всех обратимых квадратных $(n\times n)$-матриц относительно операции умножения образует группу. Следовательно, если матрица имеет обратную, то обратная единственна, так как в силу теоремы 2.1.2 это свойство выполняется в каждой группе. Матрица, имеющая обратную, называется *невырожденной*; в противном случае матрица называется *вырожденной*. Пусть $\mathbf{C} = \mathbf{AB}$. Как следует из п. (iii) теоремы 2.2.5, если хотя бы у одной из матриц $\mathbf{A}$ или $\mathbf{B}$ нет обратной, то и у матрицы $\mathbf{C}$ нет обратной. Если матрицы $\mathbf{A}$ и $\mathbf{B}$ обратимы, то $\mathbf{C}^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$, так как $(\mathbf{B}^{-1}\mathbf{A}^{-1})\,\mathbf{C} = \mathbf{I} = \mathbf{C}\,(\mathbf{B}^{-1}\mathbf{A}^{-1})$.

Определение 2.5.2. Пусть поле F задано. Для каждого n *определитель* квадратной $(n\times n)$-матрицы $\mathbf{A}$ равен величине $\det(\mathbf{A})$, являющейся функцией из множества всех $(n\times n)$-матриц над F в поле F. Функция $\det(\mathbf{A})$ задается формулой

$$\det(\mathbf{A}) = \sum \xi_{i_1 \ldots i_n} a_{1i_1} a_{2i_2} \ldots a_{ni_n},$$

где $i_1, i_2, \ldots, i_n$ — перестановка на множестве целых чисел $\{1, 2, \ldots, n\}$; $\xi_{i_1 \ldots i_n}$ равно $+1$, если перестановка содержит четное число транспозиций, и -1 в противном случае, а сумми-

рование ведется по всем перестановкам. *Транспозицией* называется перестановка двух членов.

Если матрица $\mathbf{A}'$ получается из матрицы $\mathbf{A}$ перестановкой двух строк, то каждую перестановку строк матрицы $\mathbf{A}'$, получаемую в результате четного (нечетного) числа транспозиций, можно рассматривать как соответствующую перестановку строк матрицы $\mathbf{A}$. Отсюда следует, что при перестановке любых двух строк матрицы знак определителя меняется на противоположный. Аналогичные рассуждения показывают, что если две строки матрицы равны, то определитель равен нулю.

Следующая теорема, приводимая без доказательства, содержит вытекающие непосредственно из определения 2.5.2 свойства определителя.

Теорема 2.5.3. (i) *Если все элементы некоторой строки квадратной матрицы равны нулю, то определитель этой матрицы равен нулю.*

(ii) *Определитель матрицы равен определителю транспонированной матрицы.*

(iii) *Если две строки квадратной матрицы поменять местами, то определитель поменяет знак.*

(iv) *Если две строки равны, то определитель равен нулю.*

(v) *Если все элементы одной строки матрицы умножить на элемент поля c, то определитель новой матрицы будет равен определителю исходной матрицы, умноженному на c.*

(vi) *Если матрицы $\mathbf{A}$ и $\mathbf{B}$ отличаются только i-й строкой, то сумма их определителей равна определителю матрицы $\mathbf{C}$, i-я строка которой равна сумме i-х строк матриц $\mathbf{A}$ и $\mathbf{B}$, а остальные строки равны соответствующим строкам матрицы $\mathbf{A}$ или $\mathbf{B}$.*

(vii) *Если к элементам некоторой строки матрицы k раз прибавить соответствующие элементы некоторой другой ее строки, то определитель матрицы не изменится.*

(viii) *Определитель матрицы отличен от нуля тогда и только тогда, когда ее строки (столбцы) линейно независимы.*

Если в квадратной матрице удалить строку и столбец, содержащие элемент a_{ij}, то определитель оставшейся квадратной таблицы размера $n-1$ называется *минором* элемента a_{ij} и обозначается через M_{ij}. *Алгебраическое дополнение*, обозначаемое здесь через C_{ij}, определяется равенством

$$C_{ij} = (-1)^{i+j} M_{ij}.$$

Из способа задания определителя матрицы следует, что алгебраическое дополнение элемента a_{ij} является коэффициентом при a_{ij} в разложении определителя:

$$\det(\mathbf{A}) = \sum_{k=1}^{n} a_{ik} C_{ik}.$$

Это известная формула Лапласа для разложения определителей. Формула разложения Лапласа лежит в основе рекуррентного способа вычисления определителей. Она дает выражение определителя $(n \times n)$-матрицы через определители $((n-1)\times(n-1))$-матриц.

Если a_{ik} заменить на a_{jk}, то получится сумма $\sum_{k=1}^{n} a_{jk} C_{ik}$, равная определителю новой матрицы, полученной из старой заменой элементов i-й строки элементами j-й строки; этот определитель равен нулю, если $j \neq i$. Таким образом,

$$\sum_{k=1}^{n} a_{jk} C_{ik} = \begin{cases} \det(\mathbf{A}), & i = j, \\ 0 & i \neq j. \end{cases}$$

Поэтому если $\det(\mathbf{A}) \neq 0$, то матрица $\mathbf{A}$ имеет обратную, равную

$$\mathbf{A}^{-1} = \left[\frac{C_{ji}}{\det(\mathbf{A})}\right].$$

Если $\det(\mathbf{A}) = 0$, то обратной матрицы не существует.

Матрицу можно разбить на блоки по правилу:

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} \end{array}\right],$$

где $\mathbf{A}_{11}$, $\mathbf{A}_{12}$, $\mathbf{A}_{21}$ и $\mathbf{A}_{22}$ — меньшие матрицы, размеры которых очевидным образом дополняют друг друга до размеров исходной матрицы $\mathbf{A}$. А именно, число строк матрицы $\mathbf{A}_{11}$ (или $\mathbf{A}_{12}$) плюс число строк матрицы $\mathbf{A}_{21}$ (или $\mathbf{A}_{22}$) равно числу строк матрицы $\mathbf{A}$; аналогичное утверждение выполняется для числа столбцов. Матрицы можно перемножать поблочно. А именно, если

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} \end{array}\right], \quad \mathbf{B} = \left[\begin{array}{c|c} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \hline \mathbf{B}_{21} & \mathbf{B}_{22} \end{array}\right]$$

и $\mathbf{C} = \mathbf{AB}$, то

$$\mathbf{C} = \left[\begin{array}{c|c} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \hline \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{array}\right]$$

при условии, что все размеры блоков выбраны корректно в том смысле, что все матричные произведения и суммы определены. Такое разложение может быть выведено как простое следствие аксиом ассоциативности и дистрибутивности основного поля.

Определение 2.5.4. Пусть $\mathbf{A} = [a_{ik}]$ и $\mathbf{B} = [b_{jl}]$ — матрицы соответственно размеров $I \times K$ и $J \times L$. Тогда *кронекеровским про-*

изведением матриц **A** и **B**, обозначаемым $\mathbf{A}\times\mathbf{B}$, называется матрица, содержащая IJ строк и KL столбцов, у которой на пересечении строки с номером $(i-1)\,J+j$ и столбца с номером $(k-1)\,L+l$ стоит элемент

$$c_{ij,\,kl}=a_{ik}b_{jl}.$$

Кронекеровское произведение представляет собой $(I\times K)$-таблицу, состоящую из $(J\times L)$-блоков, $(i,\ k)$-й из которых равен $a_{ik}\mathbf{B}$. Непосредственно из определения вытекает, что кронекеровское произведение матриц некоммутативно, но ассоциативно:

$$\mathbf{A}\times\mathbf{B}\neq\mathbf{B}\times\mathbf{A},\qquad(\mathbf{A}\times\mathbf{B})\times\mathbf{C}=\mathbf{A}\,(\mathbf{B}\times\mathbf{C}).$$

Элементы матрицы $\mathbf{A}\times\mathbf{B}$ те же, что и у матрицы $\mathbf{B}\times\mathbf{A}$, но упорядочены по-другому. Очевидно также, что кронекеровское произведение дистрибутивно относительно обычного сложения матриц.

Наиболее известным примером кронекеровского произведения является *внешнее произведение* двух векторов. Предположим, что **A** и **B** обе представляют собой векторы-столбцы, скажем, $\mathbf{a}=(a_1,\ \ldots,\ a_I)^T$ и $\mathbf{b}=(b_1,\ \ldots,\ b_J)^T$ соответственно. Тогда $K=L=1$, и $\mathbf{a}\times\mathbf{b}^T$ является $(I\times J)$-матрицей, на пересечении i-й строки и j-го столбца которой стоит элемент a_ib_j. Оно обозначается просто через $\mathbf{ab}^T$, так как в этом случае кронекеровское произведение совпадает с обычным произведением матриц.

Следующая полезная теорема утверждает, что кронекеровское произведение произведений матриц равно матричному произведению кронекеровских произведений соответствующих матриц.

Теорема 2.5.5. *Если все матричные произведения определены, то кронекеровское произведение удовлетворяет равенству*

$$(\mathbf{A}\times\mathbf{B})\,(\mathbf{C}\times\mathbf{D})=(\mathbf{AC})\times(\mathbf{BD}).$$

Доказательство. Пусть матрицы **A**, **B**, **C** и **D** имеют соответственно размеры $I\times K$, $J\times L$, $K\times M$ и $L\times N$. Так как матрица $\mathbf{A}\times\mathbf{B}$ содержит KL столбцов, а матрица $\mathbf{C}\times\mathbf{D}$ содержит KL строк, то матричное произведение $(\mathbf{A}\times\mathbf{B})\,(\mathbf{C}\times\mathbf{D})$ определено. Оно содержит IJ строк, которые мы занумеруем парами $(i,\ j)$, и MN столбцов, которые мы занумеруем парами $(m,\ n)$. Элемент, стоящий на пересечении строки $(i,\ j)$ и столбца $(m,\ n)$, равен $\sum\limits_{kl}a_{ik}b_{jl}c_{km}\,d_{ln}$. Так как матрица **AC** содержит I строк и M столбцов, а матрица **BD** содержит J строк и L столбцов, то матрица $(\mathbf{AC})\times(\mathbf{BD})$ также является $(IJ\times KL)$-матрицей. Стоящий на пересечении строки $(i,\ j)$ и столбца $(m,\ n)$ элемент этой матрицы равен

$$\sum_k a_{ik}c_{km}\sum_l b_{jl}d_{ln}=\sum_{k,\,l}a_{ik}b_{jl}c_{km}d_{ln},$$

что и завершает доказательство. □

$(n\times n)$-матрица, в которой $a_{ij}=a_{i'j'}$, если $i-j=i'-j'$, называется *теплицевой* $(n\times n)$-*матрицей*. Теплицева матрица имеет вид

$$\mathbf{A}=\begin{bmatrix} a_0 & a_1 & a_2 & \ldots & a_{n-1}\\ a_{-1} & a_0 & a_1 & & \\ a_{-2} & a_{-1} & a_0 & & \\ \vdots & & & & \\ a_{-(n-1)} & & \ldots & & a_0 \end{bmatrix};$$

вдоль любой ее диагонали стоит один и тот же элемент.

Элементарными операциями над строками матрицы называются следующие действия:

1) перестановка двух произвольных строк;

2) умножение произвольной строки на ненулевой элемент поля;

3) замена произвольной строки на сумму ее самой и некоторого кратного любой другой строки.

Каждая элементарная операция над строками $(n\times m)$-матрицы **A** может быть выполнена путем левого умножения **A** на соответствующим образом подобранную так называемую элементарную $(n\times n)$-матрицу **E**. Элементарные матрицы определяются как следующие модификации единичной матрицы:

$$\begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & 0 & 1 & & \\ & \vdots & & \ddots & \\ & 1 & 0 & & \\ & & & & 1 \end{bmatrix},\qquad \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & a & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}\quad \text{или}\quad \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & 1 & & \\ & a & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix}.$$

Каждая элементарная операция над строками обратима, и обратная операция имеет такой же вид.

Элементарные операции над строками используются для приведения матрицы к стандартному виду, называемому каноническим ступенчатым видом и определяемому следующим образом:

1) ведущий ненулевой элемент каждой ненулевой строки равен единице;

2) все остальные элементы каждого столбца, содержащего такой ведущий элемент, равны нулю;

3) ведущий элемент любой строки находится правее любого ведущего элемента любой расположенной выше строки. Нулевые строки расположены ниже всех ненулевых строк.

Примером матрицы, приведенной к каноническому ступенчатому виду, является

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Заметим, что нулевая строка расположена снизу и что если удалить последнюю строку, то все столбцы единичной (3×3)-матрицы появятся среди столбцов матрицы, но в разбросанном виде. В общем случае если имеется k ненулевых строк и по меньшей мере такое же количество столбцов, то матрица в каноническом ступенчатом виде будет содержать все столбцы единичной $(k\times k)$-матрицы, но в разбросанном виде.

Строки $(n\times m)$-матрицы $\mathbf{A}$ над полем F являются подмножеством векторов пространства F^m, т. е. векторами с m компонентами. *Пространством строк* матрицы $\mathbf{A}$ называется множество всех линейных комбинаций строк матрицы $\mathbf{A}$. Пространство строк образует подпространство в F^m. Размерность пространства строк называется *рангом матрицы по строкам*. Аналогично, столбцы матрицы $\mathbf{A}$ можно рассматривать как множество векторов в пространстве F^n векторов с n компонентами. *Пространство столбцов* матрицы $\mathbf{A}$ определяется как множество всех линейных комбинаций столбцов матрицы $\mathbf{A}$, и размерность этого пространства называется *рангом матрицы по столбцам*. Множество векторов $\mathbf{v}$, таких что $\mathbf{A}\mathbf{v}^T = 0$, называется *нулевым пространством* матрицы $\mathbf{A}$. Нулевое пространство, очевидно, является векторным подпространством пространства F^m. В частности, нулевое пространство матрицы $\mathbf{A}$ является ортогональным дополнением пространства строк матрицы $\mathbf{A}$, так как нулевое пространство может быть описано как множество всех векторов, ортогональных ко всем векторам пространства строк матрицы.

Теорема 2.5.6. *Если две матрицы* $\mathbf{A}$ *и* $\mathbf{A}'$ *переводятся друг в друга некоторой последовательностью элементарных операций над строками, то они имеют одно и то же пространство строк.*

Доказательство. Каждая строка матрицы $\mathbf{A}'$ является линейной комбинацией строк матрицы $\mathbf{A}$; следовательно, любая линейная комбинация строк матрицы $\mathbf{A}'$ также является линейной комбинацией строк матрицы $\mathbf{A}$, и тем самым пространство строк матрицы $\mathbf{A}$ содержит пространство строк матрицы $\mathbf{A}'$. Но $\mathbf{A}$ может быть получена из $\mathbf{A}'$ с помощью обратных операций, и, следовательно, пространство строк матрицы $\mathbf{A}'$ содержит пространство строк матрицы $\mathbf{A}$. Следовательно, пространства строк матриц $\mathbf{A}$ и $\mathbf{A}'$ равны. □

Теорема 2.5.7. *Если матрицы* $\mathbf{A}$ *и* $\mathbf{A}'$ *связаны последовательностью элементарных операций над строками, то любое множество линейно независимых столбцов матрицы* $\mathbf{A}$ *является также линейно независимым в* $\mathbf{A}'$.

Доказательство. Теорема очевидна для первой и второй элементарных операций, так что достаточно провести доказательство для единственной третьей операции. Итак, пусть $\mathbf{A}'$ получается из $\mathbf{A}$ прибавлением строки α, умноженной на элемент поля к строке β. Если в $\mathbf{A}'$ имеется некоторая линейная зависимость столбцов, то она приводит к нулевой линейной комбинации соответствующих элементов строки α, которая поэтому никак не сказывается на строке β. Следовательно, это множество столбцов в $\mathbf{A}$ также линейно зависимо. □

Теорема 2.5.8. *Если* k *строк* $(k\times n)$*-матрицы* $\mathbf{A}$ *линейно независимы, то эта матрица содержит* k *линейно независимых столбцов.*

Доказательство. Приведем матрицу $\mathbf{A}$ к каноническому ступенчатому виду $\mathbf{A}'$. Так как строки линейно независимы, то ни одна строка не будет нулевой. Следовательно, для каждой строки найдется столбец, который в пересечении с этой строкой содержит единицу, а во всех остальных позициях — нуль. Это множество из k столбцов матрицы $\mathbf{A}'$ линейно независимо, и, следовательно, по теореме 2.5.7 в матрице $\mathbf{A}$ это же множество столбцов также линейно независимо. □

Теорема 2.5.9. *Ранг матрицы по строкам равен ее рангу по столбцам и оба равны размерам любой наибольшей квадратной подматрицы, определитель которой отличен от нуля.* (Эта величина называется просто *рангом* матрицы.)

Доказательство. Необходимо только доказать, что ранг матрицы $\mathbf{A}$ по строкам равен размеру наибольшей квадратной подматрицы с отличным от нуля определителем. То же самое доказательство, примененное к транспонированной матрице, тогда дает этот же результат для ранга матрицы по столбцам, и, таким образом, служит доказательством равенства ранга по строкам рангу по столбцам.

Подматрицей матрицы $\mathbf{A}$ называется матрица, получающаяся из $\mathbf{A}$ удалением произвольного числа строк и столбцов. Пусть $\mathbf{M}$ — невырожденная квадратная подматрица матрицы $\mathbf{A}$ наибольшего размера. Так как $\mathbf{M}$ невырожденна, то, согласно п. (viii) теоремы 2.5.3, ее строки линейно независимы, и, следовательно, эти же строки матрицы $\mathbf{A}$ должны быть линейно независимы. Таким образом, ранг матрицы $\mathbf{A}$ по строкам по меньшей мере столь же велик, сколь размер подматрицы $\mathbf{M}$.

С другой стороны, выберем произвольное множество из k линейно независимых строк. Согласно теореме 2.5.7, образованная этими строками матрица содержит k линейно независимых столбцов. Выбирая эти k столбцов на рассматриваемых k строках, получаем подматрицу с ненулевым определителем. Следовательно, размер наибольшей квадратной невырожденной подматрицы по меньшей мере столь же велик, сколь ранг матрицы $\mathbf{A}$ по строкам. Это завершает доказательство. □

2.6. Кольцо целых чисел

Целые числа (положительные, отрицательные и нуль) образуют обманчиво простое математическое множество. Ничего, кажется, не может быть более регулярного и равномерного, чем целые числа, но присмотревшись пристальнее, можно увидеть сложные внутренние связи и модели этого множества. Разумный конструктор использует эти свойства множества целых чисел при построении эффективных алгоритмов цифровой обработки сигналов.

Относительно обычных операций сложения и умножения целые числа образуют кольцо, которое принято обозначать Z. В кольце целых чисел вычитание возможно всегда, а деление не всегда. Ограниченность деления является одной из причин, делающих кольцо целых чисел столь интересной и богатой структурой.

Говорят, что целое число s делится на целое число r, или что r делит s, или что r является делителем s, если $s = ra$ для некоторого целого числа a. Этот факт записывается символом $r \mid s$, который читается «r делит s». Если r и делит s, и делится на s, то $r = \pm s$. Действительно, $r = sa$ и $s = rb$ для некоторых целых чисел a и b. Следовательно, $r = rab$ и ab должно равняться 1. Так как a и b оба целые, то они равны 1 или -1.

Положительное целое число $p > 1$, которое делится только на $\pm p$ или ± 1, называется *простым*. Наименьшими простыми числами являются 2, 3, 5, 7, 11, 13, ...; число 1 не является простым. Не являющееся простым положительное целое число называется *составным*. *Наибольший общий делитель* двух целых чисел r и s обозначается НОД $[r, s]$ и определяется как наибольшее положительное число, которое делит оба из них. *Наименьшее общее кратное* двух положительных чисел r и s обозначается НОК $[r, s]$ и определяется как наименьшее положительное число, которое делится на оба из них. Два числа называются *взаимно простыми*, если их наибольший общий делитель равен 1.

В кольце целых чисел всегда возможно сокращение; если $ca = cb$ и c отлично от нуля, то $a = b$. В кольце целых чисел имеется также слабая форма деления, известная под названием деления с остатком или алгоритма деления.

Теорема 2.6.1 (Алгоритм деления). *Для каждого целого числа c и положительного целого числа d найдется единственная пара целых чисел Q (частное) и s (остаток), таких что $c = dQ + s$, где $0 \leqslant s < d$.*

Частное иногда обозначается

$$Q = \left\lfloor \frac{c}{d} \right\rfloor.$$

Обычно нас будет больше интересовать остаток, чем частное. Если s и c при делении на d имеют один и тот же остаток, то это записывается в виде

$$s \equiv c \pmod{d}.$$

Выражение этого вида называется сравнением и читается так: s сравнимо с c по модулю d. В сравнении ни s, ни c не должны быть обязательно меньше d. Мы больше интересуемся остатком. Остаток записывается в виде равенства

$$s = R_d\,[c],$$

которое читается так: s равно остатку от деления c на d, или s равно вычету c по модулю d. Мы будем также пользоваться скобками $((c))$ для обозначения этих же величин; в этом случае d ясно из контекста. Еще одним обозначением является

$$s = c \pmod{d},$$

в котором вместо знака сравнения стоит знак равенства. Теперь это не сравнение, а остаток от деления c на d.

Вычисление остатка от сложного выражения облегчается следующей теоремой, которая утверждает, что можно менять последовательность выполнения операции вычисления остатка со сложением и умножением.

Теорема 2.6.2. *Для фиксированного модуля d*

$$\text{(i)}\quad R_d\,[a + b] = R_d\,[R_d\,[a] + R_d\,[b]],$$
$$\text{(ii)}\quad R_d\,[a \cdot b] = R_d\,[R_d\,[a] \cdot R_d\,[b]].$$

Доказательство предоставляется читателю в качестве упражнения.

Наибольший общий делитель двух заданных положительных чисел s и t может быть вычислен с помощью итеративного применения алгоритма деления. Эта процедура известна как алгоритм Евклида. Предположим, что $t < s$; алгоритм Евклида сводится к последовательности шагов

$$s = Q^{(1)}t + t^{(1)},$$
$$t = Q^{(2)}t^{(1)} + t^{(2)},$$

$$t^{(1)} = Q^{(3)}t^{(2)} + t^{(3)},$$
$$\cdots\cdots\cdots\cdots$$
$$t^{(n-2)} = Q^{(n)}t^{(n-1)} + t^{(n)},$$
$$t^{(n-1)} = Q^{(n+1)}t^{(n)},$$

где остановка процесса наступает при получении нулевого остатка. Последний ненулевой остаток, $t^{(n)}$, равен наибольшему общему делителю. Этот факт будет доказан в следующей теореме. Матричные обозначения позволяют кратко записать шаги алгоритма Евклида в виде

$$\begin{bmatrix} s^{(r)} \\ t^{(r)} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)} \end{bmatrix} \begin{bmatrix} s^{(r-1)} \\ t^{(r-1)} \end{bmatrix}.$$

Теорема 2.6.3 (Алгоритм Евклида). *Для двух заданных положительных чисел s и t, где $s > t$, пусть $s^{(0)} = s$ и $t^{(0)} = t$. Решение рекуррентных уравнений*

$$Q^{(r)} = \left\lfloor \frac{s^{(r-1)}}{t^{(r-1)}} \right\rfloor,$$

$$\begin{bmatrix} s^{(r)} \\ t^{(r)} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)} \end{bmatrix} \begin{bmatrix} s^{(r-1)} \\ t^{(r-1)} \end{bmatrix},$$

при $r = 1, \ldots, n$ дается величиной

$$s^{(n)} = \text{НОД } [s, t],$$

где n равно наименьшему целому числу, для которого $t^{(n)} = 0$.

Доказательство. Так как $t^{(r+1)} < t^{(r)}$ и все остатки неотрицательны, то в конце концов наступит n, для которого $t^{(n)} = 0$, так что завершение работы алгоритма произойдет обязательно. Легко проверить, что

$$\begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)} \end{bmatrix}^{-1} = \begin{bmatrix} Q^{(r)} & 1 \\ 1 & 0 \end{bmatrix}.$$

Поэтому

$$\begin{bmatrix} s \\ t \end{bmatrix} = \left\{ \prod_{i=1}^{n} \begin{bmatrix} Q^{(i)} & 1 \\ 1 & 0 \end{bmatrix} \right\} \begin{bmatrix} s^{(n)} \\ 0 \end{bmatrix},$$

так что $s^{(n)}$ должно делить оба числа s и t и, следовательно, делит НОД $[s, t]$. Далее,

$$\begin{bmatrix} s^{(n)} \\ 0 \end{bmatrix} = \left\{ \prod_{i=n}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(i)} \end{bmatrix} \right\} \begin{bmatrix} s \\ t \end{bmatrix},$$

так что любой делитель чисел s и t делит $s^{(n)}$. Следовательно, НОД $[s, t]$ делит $s^{(n)}$ и делится на $s^{(n)}$. Таким образом,

$$s^{(n)} = \text{НОД } [s, t].$$

Это завершает доказательство. □

Из этой теоремы вытекает несколько важных следствий. Пусть

$$\mathbf{A}^{(r)} = \prod_{i=r}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(i)} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)} \end{bmatrix} \mathbf{A}^{(r-1)}.$$

Тогда получаем следствие, являющееся важным и интуитивно непредсказуемым результатом теории чисел, утверждающим, что наибольший общий делитель двух целых чисел равен их линейной комбинации.

Следствие 2.6.4. *Для любых целых чисел s и t найдутся такие целые числа a и b, что*

$$\text{НОД } [s, t] = as + bt.$$

Доказательство. Достаточно доказать следствие для положительных s и t. Так как

$$\begin{bmatrix} s^{(n)} \\ 0 \end{bmatrix} = \mathbf{A}^{(n)} \begin{bmatrix} s \\ t \end{bmatrix}$$

и

$$s^{(n)} = \text{НОД } [s, t],$$

то утверждение выполняется при $a = A_{11}^{(n)}$ и $b = A_{12}^{(n)}$. □

Из доказательства этого следствия видно, как целые числа a и b вычисляются в виде элементов матрицы $\mathbf{A}$. Остальные два элемента матрицы также имеют свою интерпретацию, для описания которой нам понадобится обратная к матрице $\mathbf{A}^{(r)}$. Напомним, что

$$\mathbf{A}^{(r)} = \prod_{i=r}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(i)} \end{bmatrix}.$$

Отсюда видно, что определитель матрицы $\mathbf{A}^{(r)}$ равен $(-1)^r$. Обратная к $\mathbf{A}^{(r)}$ матрица равна

$$\begin{bmatrix} A_{11}^{(r)} & A_{12}^{(r)} \\ A_{21}^{(r)} & A_{22}^{(r)} \end{bmatrix}^{-1} = (-1)^r \begin{bmatrix} A_{22}^{(r)} & -A_{12}^{(r)} \\ -A_{21}^{(r)} & A_{11}^{(r)} \end{bmatrix}.$$

Следствие 2.6.5. *Получаемые в процессе алгоритма Евклида матричные элементы $A_{21}^{(n)}$ и $A_{22}^{(n)}$ удовлетворяют равенствам*

$$s = (-1)^n A_{22}^{(n)} \text{ НОД } [s, t],$$
$$t = -(-1)^n A_{21}^{(n)} \text{ НОД } [s, t].$$

Доказательство. Используя выписанное выше выражения для обратной матрицы и обращая первое равенство из доказательства следствия 2.6.4, получаем

$$\begin{bmatrix} s \\ t \end{bmatrix} = (-1)^n \begin{bmatrix} A_{22}^{(n)} & -A_{12}^{(n)} \\ -A_{21}^{(n)} & A_{11}^{(n)} \end{bmatrix} \begin{bmatrix} s^{(n)} \\ 0 \end{bmatrix}.$$

Утверждение вытекает отсюда непосредственно. □

Используя алгоритм деления, можно вычислить наибольший общий делитель двух целых чисел. Например, НОД [814, 187] находится следующим образом:

$$\begin{bmatrix} s^{(n)} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -4 \end{bmatrix} \begin{bmatrix} 814 \\ 187 \end{bmatrix} =$$
$$= \begin{bmatrix} 3 & -13 \\ -17 & 74 \end{bmatrix} \begin{bmatrix} 814 \\ 187 \end{bmatrix} = \begin{bmatrix} 11 \\ 0 \end{bmatrix}.$$

Из этих вычислений сразу следует, что НОД [814, 187] равен 11 и что НОД [814, 187] $= 3^3 \times 814 - 13 \times 187$.

2.7. Кольца многочленов

Для каждого поля F имеется кольцо $F[x]$, называемое кольцом многочленов на F. Во многих отношениях кольцо многочленов аналогично кольцу целых чисел. Чтобы сделать эту аналогию очевидной, в изложении данного раздела мы следуем разд. 2.6.

Многочленом над полем F называется математическое выражение

$$f(x) = f_n x^n + f_{n-1} x^{n-1} + \ldots + f_1 x + f_0 = \sum_{i=0}^{n} f_i x^i,$$

где символ x называется *неопределенной переменной*, а коэффициенты $f_0, \ldots, f_n$ принадлежат полю. *Нулевым многочленом* называется многочлен $f(x) = 0$. *Степень* многочлена обозначается $\deg f(x)$ и определяется как индекс старшего ненулевого коэффициента. По определению степень нулевого многочлена полагается равной отрицательной бесконечности ($-\infty$). *Приведенным многочленом* называется многочлен, старший коэффициент f_n которого равен 1. Два многочлена равны, если равны все их коэффициенты f_i.

В кольце всех многочленов над заданным полем сложение и умножение определяются как обычные сложение и умножение многочленов. Такое кольцо многочленов определено для каждого поля F и обозначается символом $F[x]$. В исследованиях по этому кольцу элементы поля F иногда называются *скалярами*.

Суммой двух многочленов из $F[x]$ называется другой многочлен из $F[x]$, определяемый равенством

$$f(x) + g(x) = \sum_{i=0}^{\infty} (f_i + g_i) x^i,$$

где, конечно, члены с индексом, бо́льшим наибольшей из степеней многочленов $f(x)$ и $g(x)$, равны нулю. Степень суммы не превосходит наибольшей из этих двух степеней. Произведением двух

многочленов из $F[x]$ называется многочлен из $F[x]$, определяемый равенством

$$f(x)\, g(x) = \sum_i \left(\sum_{j=0}^{i} f_j g_{i-j} \right) x^i.$$

Степень произведения равна сумме степеней множителей. Если $f(x) \neq 0$ и $g(x) \neq 0$, то $f(x) \cdot g(x) \neq 0$, так как $\deg p(x)$ равно отрицательной бесконечности тогда и только тогда, когда $p(x) = 0$.

В кольце многочленов вычитание возможно всегда, а деление не всегда. Будем писать $r(x) \mid s(x)$ и говорить, что многочлен $s(x)$ *делится* на многочлен $r(x)$, или что многочлен $r(x)$ *делит* $s(x)$, или что $r(x)$ *является делителем* многочлена $s(x)$, если существует многочлен $a(x)$, такой что $r(x)\, a(x) = s(x)$. Ненулевой многочлен $p(x)$, делящийся только на $p(x)$ или на произвольный элемент α поля, называется *неприводимым многочленом*. Приведенный неприводимый многочлен называется *простым многочленом*.

Чтобы выяснить, является ли многочлен простым, надо знать поле, над которым он рассматривается. Многочлен $x^4 - 2$ является простым над полем рациональных чисел, но не над полем вещественных чисел, над которым он распадается в произведение двух простых многочленов: $p(x) = (x^2 - \sqrt{2})(x^2 + \sqrt{2})$. Над полем комплексных чисел каждый из последних многочленов не является простым.

Если $r(x)$ и делит $s(x)$, и делится на $s(x)$, то $r(x) = \alpha s(x)$, где α — элемент поля F. Это доказывается следующим образом. Должны существовать многочлены $a(x)$ и $b(x)$, такие что $r(x) = s(x)\, a(x)$ и $s(x) = r(x)\, b(x)$. Следовательно, $r(x) = r(x) \times b(x)\, a(x)$. Но степень стоящего справа многочлена равна сумме степеней многочленов $r(x)$, $a(x)$ и $b(x)$. Так как эта сумма должна равняться степени многочлена, стоящего слева, то многочлены $a(x)$ и $b(x)$ должны иметь степени, равные нулю, т. е. являться скалярами.

Наибольший общий делитель двух многочленов $r(x)$ и $s(x)$ обозначается НОД $[r(x), s(x)]$ и определяется как приведенный многочлен наибольшей степени, делящий одновременно оба из них. Если наибольший общий делитель двух многочленов равен 1, то они называются *взаимно простыми*.

Наименьшее общее кратное двух многочленов $r(x)$ и $s(x)$ обозначается НОК $[r(x), s(x)]$ и определяется как приведенный многочлен наименьшей степени, делящийся на оба из них. Мы увидим, что наибольший общий делитель и наименьшее общее кратное многочленов $r(x)$ и $s(x)$ определены однозначно.

В поле вещественных чисел операция дифференцирования вводится через операцию предельного перехода. Это определение возможно не всегда, так как в некоторых полях отсутствует по-

нятие произвольно малого числа. В таких полях удобно просто ввести операцию над многочленами, результат которой ведет себя так, как вела бы производная. Такой многочлен называется формальной производной от многочлена.

Определение 2.7.1. Пусть $r(x) = r_n x^n + r_{n-1}x^{n-1} + \ldots + r_1 x + r_0$ есть многочлен над полем F. *Формальной производной* от $r(x)$ называется многочлен вида

$$r'(k) = (nr_n)\, x^{n-1} + ((n-1)\, r_{n-1})\, x^{n-1} + \ldots + r_1,$$

где новые коэффициенты вычисляются в поле F как суммы i копий элемента r_i:

$$(ir_i) = r_i + r_i + \ldots + r_i.$$

Легко проверить, что сохраняются многие полезные свойства производных, а именно, что

$$[r(x)\, s(x)]' = r'(x)\, s(x) + r(x)\, s'(x),$$

и что если $a^2(x)$ делит $r(x)$, то $a(x)$ делит $r'(x)$.

В кольце многочленов над полем возможно сокращение; если $c(x)\, a(x) = c(x)\, b(x)$ и $c(x)$ отличен от нуля, то $a(x) = b(x)$. Кроме того, в кольце многочленов имеется слабая форма деления, известная под названием деления с остатком или алгоритма деления.

Теорема 2.7.2 (Алгоритм деления для многочленов). *Для каждого многочлена $c(x)$ и ненулевого многочлена $d(x)$ существует единственная пара многочленов $Q(x)$ (частное) и $s(x)$ (остаток), таких что*

$$c(x) = Q(x)\, d(x) + s(x)$$

и $\deg s(x) < \deg d(x)$.

Доказательство. Частное и остаток находятся по элементарному правилу деления многочленов. Они единственны, так как если

$$c(x) = Q_1(x)\, d(x) + s_1(x) = Q_2(x)\, d(x) + s_2(x),$$

то

$$d(x)\, [Q_1(x) - Q_2(x)] = s_1(x) - s_2(x).$$

В правой части равенства стоит ненулевой многочлен, степень которого меньше $\deg d(x)$, а в левой — ненулевой многочлен, степень которого не меньше $\deg d(x)$. Следовательно, оба многочлена равны нулю, и представление единственно. □

Практически вычисление частного и остатка выполняется с помощью простого правила деления «уголком». Частное иногда обозначается

$$Q(x) = \left\lfloor \frac{c(x)}{d(x)} \right\rfloor.$$

Обычно мы будем больше интересоваться остатком, чем частным. Остаток часто записывается в виде

$$s(x) = R_{d(x)}\ [c(x)],$$

или

$$s(x) = c(x) \pmod{d(x)}.$$

Можно записать также сравнение

$$s(x) \equiv c(x) \pmod{d(x)},$$

которое обозначает, что многочлены $s(x)$ и $c(x)$ имеют один и тот же остаток при делении на $d(x)$. Чтобы найти $R_{d(x)}\ [c(x)]$, казалось бы, надо выполнить деление. Однако имеется несколько приемов, позволяющих сократить и упростить необходимую для этого работу. Прежде всего заметим, что

$$R_{d(x)}\ [c(x)] = R_{d(x)}\ [c(x) + a(x)\, d(x)].$$

Не изменяя остатка, к $c(x)$ можно прибавить любое кратное многочлена $d(x)$. Следовательно, не изменяя остатка, можно исключить старший коэффициент многочлена $c(x)$, прибавляя соответствующее кратное многочлена $d(x)$. Используя этот метод при приведении многочлена $c(x)$ по модулю приведенного многочлена

$$d(x) = x^n + \sum_{i=0}^{n-1} d_i x^i,$$

для упрощения можно член x^n заменить многочленом $-\sum_{k=0}^{n-1} d_i x^i$ всюду, где это удобно. Такой прием упрощает вычисление остатка путем деления многочленов «уголком».

Другой способ упрощения задачи вычисления остатка от деления дается следующей теоремой.

Теорема 2.7.3. *Если многочлен $d(x)$ кратен многочлену $g(x)$, то*

$$R_{g(x)}\ [a(x)] = R_{g(x)}\ [R_{d(x)}\ [a(x)]]$$

для любого $a(x)$.

Доказательство. Пусть $d(x) = g(x)\, h(x)$ для некоторого $h(x)$. Раскрывая правую часть, получаем

$$a(x) = Q_1(x)\, d(x) + R_{d(x)}\ [a(x)] =$$
$$= Q_1(x)\, h(x)\, g(x) + Q_2(x)\, g(x) + R_{g(x)}\ [R_{d(x)}\ [a(x)]],$$

где степень остатка меньше $\deg g(x)$. Раскрывая левую часть, имеем

$$a(x) = Q(x)\, g(x) + R_{g(x)}\ [a(x)],$$

и, согласно алгоритму деления, такая запись однозначна при степени остатка, меньшей $\deg g(x)$. Теорема вытекает из отождествления подобных членов в обоих выражениях. □

В качестве примера использования теоремы 2.7.3 разделим

$x^7 + x + 1$ на $x^4 + x^3 + x^2 + x + 1$. Деление «уголком» было бы утомительно, но если вспомнить, что $(x - 1)(x^4 + x^3 + x^2 + x + 1) = x^5 - 1$, то можно сначала разделить на $x^5 - 1$, а затем на $x^4 + x^3 + x^2 + x + 1$. Тогда

$$R_{x^5-1}[x^7 + x + 1] = x^2 + x + 1,$$

и теперь деление на $x^4 + x^3 + x^2 + x + 1$ тривиально, так что

$$R_{x^4+x^3+x^2+x+1}[x^7 + x + 1] = x^2 + x + 1.$$

Другое удобное правило дается следующей теоремой.

Теорема 2.7.4.

(i) $R_{d(x)}[a(x) + b(x)] = R_{d(x)}[a(x)] + R_{d(x)}[b(x)]$.

(ii) $R_{d(x)}[a(x) \cdot b(x)] = R_{d(x)}\{R_{d(x)}[a(x)] \cdot R_{d(x)}[b(x)]\}$.

Доказательство. Применяя алгоритм деления к обеим частям первого равенства, запишем

$$a(x) + b(x) = Q(x)\, d(x) + R_{d(x)}[a(x) + b(x)]$$

и

$$a(x) + b(x) = Q'(x)\, d(x) + R_{d(x)}[a(x)] + Q''(x)\, d(x) + R_{d(x)}[b(x)].$$

Часть (i) вытекает из однозначности алгоритма деления. Часть (ii) доказывается аналогичным образом. □

Подобно тому как часто бывает полезным представление целых чисел в виде произведения простых сомножителей, часто бывает полезным представление многочленов в виде произведения неприводимых многочленов. Чтобы сделать разложение целых чисел на простые множители однозначным, приходится ограничить рассмотрение только положительными простыми числами. Аналогично, для однозначности разложения многочленов приходится условиться, что используемые в качестве простых делителей многочлены являются приведенными многочленами.

Теорема 2.7.5 (теорема об однозначном разложении). *Многочлен над некоторым полем однозначно разлагается в произведение элемента поля и простых над данным полем многочленов, причем степень каждого из них равна по меньшей мере* 1.

Доказательство. Ясно, что входящим в произведение элементом поля является коэффициент p_n, где n — степень разлагаемого многочлена $p(x)$. Этим элементом можно пренебречь и доказывать теорему для приведенных многочленов.

Предположим, что теорема не верна и пусть $p(x)$ — приведенный многочлен наименьшей степени, для которого она не верна.

Тогда имеются два разложения,

$$p(x) = a_1(x)\, a_2(x) \ldots a_k(x) = b_1(x)\, b_2(x) \ldots b_j(x),$$

где $a_k(x)$ и $b_j(x)$ — простые многочлены.

Все многочлены $a_k(x)$ должны отличаться от всех многочленов $b_j(x)$, так как в противном случае можно было бы сократить общие члены и получить многочлен меньшей степени, который можно разложить двумя разными способами.

Без потери общности предположим, что степень многочлена $b_1(x)$ не больше степени многочлена $a_1(x)$. Тогда

$$a_1(x) = b_1(x)\, Q(x) + s(x),$$

где $\deg s(x) < \deg b_1(x) \leqslant \deg a_1(x)$. Далее,

$$s(x)\, a_2(x) \ldots a_k(x) = b_1(x)[b_2(x) \ldots b_j(x) - Q(x)\, a_2(x) \ldots a_k(x)].$$

Разложим многочлен $s(x)$ и стоящий в квадратных скобках многочлен на простые множители, разделив, если надо, на соответствующий элемент поля так, чтобы все множители были приведенными. Поскольку $b_1(x)$ не содержится в левой части, мы получили два различных разложения приведенного многочлена, степень которого меньше степени многочлена $p(x)$. Это противоречие доказывает теорему. □

В силу теоремы об однозначном разложении теперь ясно, что НОД $[s(x), t(x)]$ и НОК $[s(x), t(x)]$ являются единственными для любых двух многочленов $s(x)$ и $t(x)$, так как наибольший общий делитель равен произведению всех общих для $s(x)$ и $t(x)$ простых делителей, причем каждый делитель входит в наименьшей из степеней, в которой он входит в $s(x)$ и в $t(x)$, а наименьшее общее кратное равно произведению всех простых делителей, входящих либо в $s(x)$, либо в $t(x)$, причем каждый делитель входит в наибольшей степени, в которых он входит в $s(x)$ или в $t(x)$.

Из алгоритма деления для многочленов вытекает важное следствие, известное под названием алгоритма Евклида для многочленов. Для заданных двух многочленов $s(x)$ и $t(x)$ их наибольший общий делитель может быть вычислен с помощью итеративного применения алгоритма деления. Без потери общности можно полагать, что $\deg s(x) \geqslant \deg t(x)$; алгоритм состоит из последовательности шагов

$$\begin{aligned}
s(x) &= Q^{(1)}(x)\, t(x) + t^{(1)}(x),\\
t(x) &= Q^{(2)}(x)\, t^{(1)}(x) + t^{(2)}(x),\\
t^{(1)}(x) &= Q^{(3)}(x)\, t^{(2)}(x) + t^{(3)}(x),\\
&\ldots\\
t^{(n-2)}(x) &= Q^{(n)}(x)\, t^{(n-1)}(x) + t^{(n)}(x),\\
t^{(n-1)}(x) &= Q^{(n+1)}(x)\, t^{(n)}(x),
\end{aligned}$$

где остановка процесса наступает при получении нулевого остатка. Последний ненулевой остаток $t^{(n)}(x)$ равен наибольшему общему делителю. Доказательство этого факта дается следующей теоремой.

Теорема 2.7.6 (Алгоритм Евклида для многочленов). *Для двух заданных многочленов $s(x)$ и $t(x)$ с $\deg s(x) \geqslant \deg t(x)$ пусть $s^{(0)}(x) = s(x)$ и $t^{(0)}(x) = t(x)$. Решение рекуррентных уравнений*

$$Q^{(r)}(x) = \left[\frac{s^{(r-1)}(x)}{t^{(r-1)}(x)}\right],$$

$$\begin{bmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} \begin{bmatrix} s^{(r-1)}(x) \\ t^{(r-1)}(x) \end{bmatrix}$$

при $r = 1, \ldots, n$ дается величиной

$$s^{(n)}(x) = \alpha \text{ НОД } [s(x), t(x)],$$

где n — наименьшее положительное число, для которого $t^{(n)}(x) = 0$ и $\alpha \neq 0$ принадлежит полю.

Доказательство. Так как $\deg t^{(r+1)}(x) < \deg t^{(r)}(x)$, то для некоторого n обязательно наступит событие $t^{(n)}(x) = 0$, так что алгоритм будет завершен. Легко проверить, что

$$\begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix}^{-1} = \begin{bmatrix} Q^{(r)}(x) & 1 \\ 1 & 0 \end{bmatrix}.$$

Следовательно,

$$\begin{bmatrix} s(x) \\ t(x) \end{bmatrix} = \left\{ \prod_{l=1}^{n} \begin{bmatrix} Q^{(l)}(x) & 1 \\ 1 & 0 \end{bmatrix} \right\} \begin{bmatrix} s^{(n)}(x) \\ 0 \end{bmatrix},$$

так что многочлен $s^{(n)}(x)$ должен делить оба многочлена $s(x)$ и $t(x)$ и, следовательно, НОД $[s(x), t(x)]$. Далее,

$$\begin{bmatrix} s^{(n)}(x) \\ 0 \end{bmatrix} = \left\{ \prod_{l=n}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{bmatrix} \right\} \begin{bmatrix} s(x) \\ t(x) \end{bmatrix},$$

так что любой делитель обоих многочленов $s(x)$ и $t(x)$ должен делить и многочлен $s^{(n)}(x)$. Таким образом, НОД $[s(x), t(x)]$ делит $s^{(n)}(x)$ и делится на $s^{n}(x)$, и, следовательно,

$$s^{(n)}(x) = \alpha \text{ НОД } [s(x), t(x)],$$

где α — ненулевой элемент поля. Это завершает доказательство теоремы. □

Снова, как и в случае с кольцом целых чисел, получаем два важных следствия. Определим матрицу многочленов

$$\mathbf{A}^{(r)}(x) = \prod_{l=r}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} \mathbf{A}^{(r-1)}(x),$$

где $\mathbf{A}^{(0)}(x)$ есть единичная матрица. Тогда мы имеем следующий результат.

Следствие 2.7.7. *Для любых двух многочленов $s(x)$ и $t(x)$ над полем F существуют два других многочлена $a(x)$ и $b(x)$ над тем же полем, такие что*

$$\text{НОД } [s(x), t(x)] = a(x)\, s(x) + b(x)\, t(x).$$

Доказательство. Так как

$$\begin{bmatrix} s^{(n)}(x) \\ 0 \end{bmatrix} = \mathbf{A}^{(n)}(x) \begin{bmatrix} s(x) \\ t(x) \end{bmatrix}$$

и $s^{(n)}(x) = \alpha$ НОД $[s(x), t(x)]$, то утверждение следствия выполняется при $a(x) = \alpha^{-1} A_{11}^{(n)}(x)$ и $b(x) = \alpha^{-1} A_{12}^{(n)}(x)$. □

Многочлены $a(x)$ и $b(x)$ получаются из элементов матрицы $\mathbf{A}^{(n)}(x)$ делением на α^{-1}. Два других элемента матрицы $\mathbf{A}(x)$ также имеют прямую интерпретацию, для описания которой нам надо вычислить обратную к матрице $\mathbf{A}^{(r)}(x)$. Напомним, что

$$\mathbf{A}^{(r)}(x) = \prod_{l=r}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{bmatrix}.$$

Отсюда видно, что определитель матрицы $\mathbf{A}^{(r)}(x)$ равен $(-1)^r$, а обратная матрица дается равенством

$$\begin{bmatrix} A_{11}^{(r)}(x) & A_{12}^{(r)}(x) \\ A_{21}^{(r)}(x) & A_{22}^{(r)}(x) \end{bmatrix}^{-1} = (-1)^r \begin{bmatrix} A_{22}^{(r)}(x) & -A_{12}^{(r)}(x) \\ -A_{21}^{(r)}(x) & A_{11}^{(r)}(x) \end{bmatrix}.$$

Следствие 2.7.8. *Вычисляемые в процессе алгоритма Евклида элементы $A_{21}^{(n)}(x)$ и $A_{22}^{(n)}(x)$ удовлетворяют равенствам*

$$s(x) = (-1)^n A_{22}^{(n)}(x) \alpha \text{ НОД } [s(x), t(x)],$$

$$t(x) = -(-1)^n A_{21}^{(n)}(x) \alpha \text{ НОД } [s(x), t(x)].$$

Доказательство. Обратим первое равенство в доказательстве следствия 2.7.7, используя выведенную формулу для $[\mathbf{A}^{(r)}(x)]^{-1}$:

$$\begin{bmatrix} s(x) \\ t(x) \end{bmatrix} = (-1)^n \begin{bmatrix} A_{22}^{(n)}(x) & -A_{12}^{(n)}(x) \\ -A_{21}^{(n)}(x) & A_{11}^{(n)}(x) \end{bmatrix} \begin{bmatrix} s^{(n)}(x) \\ 0 \end{bmatrix}.$$

Отсюда утверждение следствия очевидно. □

В качестве примера использования алгоритма Евклида для многочленов вычислим НОД $[s(x), t(x)]$ при $s(x) = x^4 - 1$ и $t(x) = x^3 + 2x^2 + 2x + 1$. Имеем

$$\begin{bmatrix} s^{(n)}(x) \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -\frac{4}{3}x - \frac{4}{3} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -\frac{1}{2}x - \frac{1}{4} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -x + 2 \end{bmatrix} \begin{bmatrix} s(x) \\ t(x) \end{bmatrix} =$$

$$= \begin{bmatrix} -\frac{1}{2}x - \frac{1}{4} & \frac{1}{2}x^2 - \frac{3}{4}x + \frac{1}{2} \\ \frac{4}{3}x^2 + \frac{4}{3}x + \frac{4}{3} & -\frac{4}{3}x^3 + \frac{4}{3}x^2 - \frac{4}{3}x + \frac{4}{3} \end{bmatrix} \begin{bmatrix} x^4 - 1 \\ x^3 + 2x^2 + 2x + 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{4}{3}(x+1) \\ 0 \end{bmatrix}.$$

Таким образом, НОД $[x^4 - 1, x^3 + 2x^2 + 2x + 1] = x + 1$. Кроме того,

$$x + 1 = \left(-\frac{2}{3}x - \frac{1}{3}\right) s(x) + \left(\frac{2}{3}x^2 - x + \frac{2}{3}\right) t(x),$$

как обещано следствием 2.7.7.

Можно вычислить значение многочлена $p(x)$ над полем F в любом элементе β этого поля. Для этого надо вместо неопределенной переменной x подставить соответствующий элемент β и найти элемент $p(\beta)$ этого поля. Можно также вычислить значение многочлена в элементе из любого большего поля, содержащего поле F. Для этого надо вместо неопределенной переменной x подставить значение элемента из расширения поля. Если F — поле вещественных чисел, то вычисление многочлена в расширении поля является знакомой операцией. Многочлены с вещественными коэффициентами часто приходится вычислять в поле комплексных чисел.

Элемент β поля называется *корнем* многочлена $p(x)$, если $p(\beta) = 0$. Корни многочлена не обязательно лежат в его собственном поле. Многочлен $p(x) = x^2 + 1$ не имеет корней в поле вещественных чисел.

Теорема 2.7.9. *Элемент β поля является корнем ненулевого многочлена $p(x)$ тогда и только тогда, когда $(x - \beta)$ является делителем многочлена $p(x)$. Более того, если степень многочлена равна n, то многочлен имеет в поле не более n корней.*

Доказательство. Согласно алгоритму деления,

$$p(x) = (x - \beta) Q(x) + s(x),$$

где степень многочлена $s(x)$ меньше единицы, так что $s(x)$ является элементом поля, s_0. Следовательно,

$$0 = p(\beta) = (\beta - \beta) Q(\beta) + s_0,$$

так что $s(x) = s_0 = 0$. Наоборот, если $(x - \beta)$ делит $p(x)$, то

$$p(x) = (x - \beta) Q(x)$$

и $p(\beta) = (\beta - \beta) Q(\beta) = 0$, так что β является корнем многочлена $p(x)$.

Теперь разложим многочлен в произведение скаляра и простых многочленов. Степень многочлена $p(x)$ равна сумме степеней простых делителей, и один такой простой делитель имеется для каждого корня. Следовательно, число корней не может превосходить n. □

Теорема 2.7.10 (Интерполяция Лагранжа). *Пусть $\beta_0, \ldots, \beta_n$ — множество из $n + 1$ различных точек, и пусть $p(\beta_k)$ заданы для всех $k = 0, 1, \ldots, n$. Тогда найдется* в точности *один многочлен $p(x)$ степени n или меньше, принимающий значения $p(\beta_k)$ для всех $k = 0, \ldots, n$. Этот многочлен дается равенством*

$$p(x) = \sum_{i=0}^{n} p(\beta_i) \frac{\prod_{j \neq i} (x - \beta_j)}{\prod_{j \neq i} (\beta_i - \beta_j)}.$$

Доказательство. Непосредственной подстановкой β_k вместо x можно убедиться, что многочлен $p(x)$ принимает в этих точках указанные значения. Однозначность представления следует из того, что если $p'(x)$ и $p''(x)$ — два таких многочлена, то многочлен $P(x) = p'(x) - p''(x)$ имеет степень, не превосходящую n, но $n + 1$ корней β_k при $k = 0, \ldots, n$. Следовательно, $P(x)$ является нулевым многочленом. □

2.8. Китайские теоремы об остатках

Любое неотрицательное целое число, не превосходящее произведения модулей, можно однозначно восстановить, если известны его вычеты по этим модулям. Этот результат был известен еще в древнем Китае и носит название китайской теоремы об остатках. Формулировка китайской теоремы об остатках приведена на рис. 2.2. Мы будем доказывать ее в два этапа. Сначала будет доказана единственность решения, а затем (построением процедуры отыскания решения) его существование.

Прежде чем строить формальную теорию, приведем простой пример. Выберем в качестве модулей $m_0 = 3$, $m_1 = 4$ и $m_2 = 5$ и положим $M = m_0 m_1 m_2 = 60$. Для заданного числа c, лежащего в интервале $0 \leqslant c < 60$, обозначим $c_i = R_{m_i}[c]$. Китайская теорема утверждает, что между шестьюдесятью такими числами c и шестьюдесятью векторами (c_0, c_1, c_2) соответствующих вычетов

Прямые уравнения

$c_i = R_{m_i}[c] \quad i = 0, \ldots, k$

где m_i взаимно просты

Обратные уравнения

$$c = \sum_{i=0}^{k} c_i N_i M_i \pmod{M}$$

где $M = \prod_{i=0}^{k} m_i \quad M_i = M/m_i$

и N_i являются решениями уравнений

$N_i M_i + n_i m_i = 1$

Рис. 2.2. Китайская теорема об остатках.

существует взаимно однозначное соответствие. Предположим, что $c_0 = 2$, $c_1 = 1$ и $c_2 = 2$. Тогда эти условия приводят к следующим возможностям:

$$c \in \{2, 5, 8, 11, 14, 17, 20, 23, 26, 29, \ldots\},$$
$$c \in \{1, 5, 9, 13, 17, 21, 25, 29, 33, \ldots\},$$
$$c \in \{2, 7, 12, 17, 22, 27, 32, 37, \ldots\}.$$

Единственным решением служит $c = 17$. Позже будет дан простой алгоритм вычисления числа c по его вычетам.

В рассмотренном примере число однозначно было восстановлено по его вычетам. В следующей теореме доказывается, что это имеет место в общем случае.

Теорема 2.8.1. *Для заданного множества целых положительных попарно взаимно простых чисел $m_0, m_1, \ldots, m_k$ и множества неотрицательных целых чисел $c_0, c_1, \ldots, c_k$ при $c_i < m_i$ система*

$$c_i = c \pmod{m_i}, \quad i = 0, \ldots, k,$$

имеет не более одного решения в интервале $0 \leqslant c < \prod_{i=1}^{k} m_i$.

Доказательство. Предположим, что c и c' являются двумя лежащими в рассматриваемом интервале решениями. Тогда

$$c = Q_i m_i + c_i \text{ и } c' = Q_i' m_i + c_i'$$

и, следовательно, $c - c'$ кратно m_i для каждого i, а так как m_i попарно взаимно просты, то $c - c_k'$ кратно $\prod_{i=0}^{k} m_i$. Но число $c - c'$

лежит между $-\left(\prod_{i=0}^{k} m_i - 1\right)$ и $\prod_{i=0}^{k} m_i$. Единственным положительным числом, удовлетворяющим этим условиям, является $c - c' = 0$. Следовательно, $c = c'$. □

Имеется простой путь построения решения системы сравнений из теоремы 2.8.1, основанный на следствии из алгоритма Евклида. Согласно этому следствию, в кольце целых чисел для каждого s и t найдутся такие a и b, что

$$\text{НОД}\,[s, t] = as + bt.$$

Для заданного множества попарно взаимно простых положительных чисел $m_0, m_1, \ldots, m_k$, используемых в качестве модулей, положим $M = \prod_{i=0}^{k} m_i$ и $M_i = M/m_i$. Тогда НОД $[M_i, m_i] = 1$, и, следовательно, существуют такие целые N_i и n_i, что

$$N_i M_i + n_i m_i = 1, \quad i = 0, \ldots, k.$$

Теперь можно доказать следующую теорему.

Теорема 2.8.2. *Пусть $M = \prod_{i=0}^{k} m_i$ — произведение попарно взаимно простых положительных чисел, пусть $M_i = M/m_i$ и пусть для каждого i N_i удовлетворяют равенствам $N_i M_i + n_i m_i = 1$. Тогда единственным решением системы сравнений*

$$c_i = c \pmod{m_i}, \quad i = 0, \ldots, k,$$

является

$$c = \sum_{i=0}^{k} c_i N_i M_i \pmod{M}.$$

Доказательство. Поскольку мы уже знаем, что решение рассматриваемой системы сравнений единственно, надо только доказать, что выписанное выше c действительно является решением. Но для такого c

$$c = \sum_{r=0}^{k} c_r N_r M_r = c_i N_i M_i \pmod{m_i},$$

ибо m_i делит M_r при $r \neq i$. Наконец, так как $N_i M_i + n_i m_i = 1$, то $N_i M_i = 1 \pmod{m_i}$ и $c = c_i \pmod{m_i}$, что и завершает доказательство. □

Чтобы проиллюстрировать теорему 2.8.2, продолжим предыдущий пример. Заметим, что $M = 60$, $M_0 = 20$, $M_1 = 15$ и $M_2 = 12$. Далее, как можно вычислить по алгоритму Евклида или просто проверить, $1 = (-1) M_0 + 7m_0$, $1 = (-1) M_1 + 4m_1$,

Прямые уравнения

$$c^{(i)}(x) = R_{m^{(i)}(x)}[c(x)], \quad i = 0, \ldots, k,$$

где $m^{(i)}(x)$ взаимно просты

Обратные уравнения

$$c(x) = \sum_{i=0}^{k} c^{(i)}(x) N^{(i)}(x) M^{(i)}(x) \pmod{M(x)}$$

где $M(x) = \prod_{i=0}^{k} m^{(i)}(x)$, $M^{(i)}(x) = M(x)/m^{(i)}(x)$

и $N^{(i)}(x)$ являются решениями уравнений

$$N^{(i)}(x) M^{(i)}(x) + n^{(i)}(x) m^{(i)}(x) = 1$$

Рис. 2.3. Китайская теорема об остатках для многочленов.

$1 = (-2) M_2 + 5m_2$. Следовательно, $N_0M_0 = -20$, $N_1M_1 = -15$, $N_2M_2 = -24$ и

$$c = -20c_0 - 15c_1 - 24c_2 \pmod{60}.$$

В частности, при $c_0 = 2$, $c_1 = 1$ и $c_2 = 2$ имеем $c = -103 \pmod{60} = 17$, что мы видели и раньше.

Китайская теорема об остатках является основой представления целых чисел, в котором просто выполняется операция умножения. Допустим, что надо выполнить умножение $c = ab$. Пусть $a_i = R_{m_i}[a]$, $b_i = R_{m_i}[b]$ и $c_i = R_{m_i}[c]$ для каждого $i = 0, \ldots, k$. Тогда $c_i = a_ib_i \pmod{m_i}$, и это умножение вычислить легче, так как a_i и b_i являются малыми целыми числами. Аналогично, при сложении $c = a + b$ имеем $c_i = a_i + b_i \pmod{m_i}$ для всех $i = 0, \ldots, k$. В обоих случаях для получения окончательного ответа c должно быть восстановлено по вычетам в соответствии с китайской теоремой об остатках.

Переход к системе вычетов позволяет разбить большие целые числа на маленькие кусочки, которые легко складывать, вычитать и умножать. Если вычисления состоят только из этих операций, то такое представление является альтернативной арифметической системой. Если вычисления достаточно просты, то переход от естественной записи целых чисел к записи через систему остатков и обратное восстановление ответа в целочисленном виде могут свести на нет все возможные преимущества при вычислениях. Если, однако, объем вычислений достаточно велик, то такой переход может оказаться выгодным. Это происходит потому, что при вычислениях все промежуточные результаты можно сохранять в виде системы остатков, выполняя обратный переход к целочисленному виду только при окончательном ответе.

В кольце многочленов над некоторым полем также имеется китайская теорема об остатках, формулировка которой приведена на рис. 2.3 и которая доказывается по той же схеме, что и для целых чисел.

Теорема 2.8.3. *Для заданного множества попарно взаимно простых многочленов $m^{(0)}(x)$, $m^{(1)}(x)$, ..., $m^{(k)}(x)$ и множества многочленов $c^{(0)}(x)$, $c^{(1)}(x)$, ..., $c^{(k)}(x)$, таких что* $\deg c^{(0)}(x) < \deg m^{(i)}(x)$ *система уравнений*

$$c^{(i)}(x) = c(x) \pmod{m^{(i)}(x)}, \qquad i = 0, \ldots, k,$$

имеет не более одного решения $c(x)$, удовлетворяющего условию

$$\deg c(x) < \sum_{i=0}^{k} \deg m^{(i)}(x).$$

Доказательство. По существу доказательство совпадает с доказательством теоремы 2.8.1. Предположим, что имеются два решения, а именно

$$c(x) = Q^{(i)}(x) m^{(i)}(x) + c^{(i)}(x)$$

и

$$c'(x) = Q'^{(i)}(x) m^{(i)}(x) + c^{(i)}(x),$$

так что разность $c(x) - c'(x)$ кратна многочлену $m^{(i)}(x)$ для каждого i. Тогда многочлен $c(x) - c'(x)$ кратен и многочлену $\prod_{i=0}^{k} m^{(i)}(x)$, причем

$$\deg [c(x) - c'(x)] < \deg \left(\prod_{i=0}^{k} m^{(i)}(x) \right).$$

Следовательно, $c(x) - c'(x) = 0$, и доказательство закончено. □

Система сравнений может быть решена так же, как и в случае кольца целых чисел. В кольце многочленов над некоторым полем для любых заданных $s(x)$ и $t(x)$ существуют многочлены $a(x)$ и $b(x)$, удовлетворяющие условию

$$\text{НОД}[s(x), t(x)] = a(x) s(x) + b(x) t(x).$$

Пусть $M(x) = \prod_{i=0}^{k} m^{(i)}(x)$ и $M^{(i)}(x) = M(x)/m_i(x)$; тогда НОД $[M^{(i)}(x), m^{(i)}(x)] = 1$. Пусть $N^{(i)}(x)$ и $n^{(i)}(x)$ удовлетворяют равенству

$$N^{(i)}(x) M^{(i)}(x) + n^{(i)}(x) m^{(i)}(x) = 1.$$

Теорема 2.8.4. *Пусть $M(x) = \prod_{i=0}^{k} m^{(i)}(x)$ — произведение попарно взаимно простых многочленов; пусть $M^{(i)}(x) = M(x)/m^{(i)}(x)$ и для всех i многочлены $N^{(i)}(x)$ удовлетворяют условиям*

$N^{(i)}(x)\,M^{(i)}(x) + n^{(i)}(x)\,m^{(i)}(x) = 1$. *Система сравнений*

$$c^{(i)}(x) = c(x) \pmod{m^{(i)}(x)}, \quad i = 0, \ldots, k,$$

имеет единственное решение, даваемое многочленом

$$c(x) = \sum_{i=0}^{k} c^{(i)}(x)\,N^{(i)}(x)\,M^{(i)}(x) \pmod{M(x)}.$$

Доказательство. Нам нужно только доказать, что многочлен $c(x)$ удовлетворяет каждому сравнению данной системы. Но

$$c(x) = c^{(i)}(x)\,N^{(i)}(x)\,M^{(i)}(x) \pmod{m^{(i)}(x)},$$

так как $m^{(i)}(x)$ является делителем многочлена $M^{(i)}(x)$, если $r \neq i$. Наконец, так как $N^{(i)}(x)\,M^{(i)}(x) + n^{(i)}(x)\,m^{(i)}(x) = 1$, то $N^{(i)}(x)\,M^{(i)}(x) = 1 \pmod{m^{(i)}(x)}$ и $c(x) = c^{(i)}(x) \pmod{m^{(i)}(x)}$, что и завершает доказательство теоремы. □

Задачи

2.1. **а.** Показать, что существует только одна группа с тремя элементами. Построить ее и показать, что она абелева.
б. Показать, что существуют только две группы с четырьмя элементами. Построить их и показать, что они обе абелевы. Показать, что одна из двух групп с четырьмя элементами не содержит элементов порядка четыре. Эта группа называется четверной группой Клейна.

2.2. Пусть групповая операция на группах из задачи 2.1 называется сложением.
а. Определить умножение на трехэлементной группе так, чтобы она стала кольцом. Является ли такое определение единственным?
б. Для каждой из четырехэлементных групп определить умножение так, чтобы превратить их в кольца. Является ли каждое определение единственным?

2.3. Какое из трех построенных в задаче 2.2 колец является полем? Можно ли задать другое умножение так, чтобы получить поле?

2.4. Доказать, что в циклической группе с q элементами выполняются равенства $a^0 = a^0$ и $(a^i)^{-1} = a^{q-i}$.

2.5. **а.** Показать, что $Z_2 \times Z_3$ изоморфно Z_6.
б. Показать, что $Z_2 \times Z_4$ не изоморфно Z_8.

2.6. Привести пример кольца без единицы.

2.7. Отталкиваясь от пары $\{v_i\} \leftrightarrow \{V_k\}$ преобразования Фурье, доказать следующие стандартные свойства дискретного преобразования Фурье:
а. Линейность: $\{av_i + bv'_i\} \leftrightarrow \{aV_k + bV'_k\}$.
б. Свойство сдвига: $\{v_{((i-1))}\} \leftrightarrow \{\omega^k V_k\}$.
в. Модуляционное свойство: $\{\omega^i v_i\} \leftrightarrow \{V_{((k+1))}\}$.

2.8. Показать, что преобразование Фурье вектора данных $v_i = \omega^{ri}$, где r — целое число, содержит единственную ненулевую спектральную компоненту. Какова эта компонента, если $r = 0$? Показать, что все спектральные компоненты вектора, имеющего только одну ненулевую компоненту, не равны нулю.

2.9. Доказать, что если $\mathbf{A}$ — теплицева матрица, а $\mathbf{J}$ — обменная матрица той же размерности, то $\mathbf{A}^T = \mathbf{JAJ}$.

2.10. **а.** Найти НОД [1573,308], используя алгоритм Евклида.

б. Найти целые числа A и B, удовлетворяющие равенству

$$\text{НОД}\,[1573,308] = A \cdot 1573 + B \cdot 308.$$

2.11. Рассмотрим множество $S = \{0, 1, 2, 3\}$, на котором заданы операции

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Является ли это множество полем?

2.12. Доказать, что комплекснозначное дискретное преобразование Фурье удовлетворяет условию симметрии

$$V_k = V^*_{n-k}, \quad k = 0, \ldots, n-1,$$

тогда и только тогда, когда преобразуемая последовательность данных является вещественной.

2.13. Пусть G — произвольная группа (не обязательно конечная). Условимся называть групповую операцию умножением и единичный элемент единицей. Пусть g — произвольный элемент группы и ν — наименьшее положительное число, если оно существует, такое что $g^\nu = 1$, где под g^ν понимается ν-кратное произведение $g * g * \ldots * g$. Доказать, что подмножество $\{g, g^2, \ldots, g^{\nu-1}, g^\nu\}$ образует в G подгруппу. Доказать, что эта подгруппа является абелевой даже тогда, когда G неабелева группа.

2.14. Доказать, что множество вещественных чисел вида $\{a + b\sqrt{2}\}$, где a и b — рациональные числа, образует поле относительно обычных операций сложения и умножения.

2.15. Кольцо кватернионов состоит из всех выражений вида

$$a = a_0 + a_1 i + a_2 j + a_3 k,$$

где a_0, a_1, a_2 и a_3 — вещественные числа, а i, j и k — неопределенные переменные. Сложение и умножение определяются правилами

$$a + b = (a_0 + b_0) + (a_1 + b_1)\,i + (a_2 + b_2)\,j + (a_3 + b_3)\,k,$$

$$\begin{aligned} ab = {} & (a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3) + \\ & + (a_1b_0 + a_0b_1 - a_3b_2 + a_2b_3)\,i + \\ & + (a_2b_0 + a_3b_1 + a_0b_2 - a_1b_3)\,j + \\ & + (a_3b_0 - a_2b_1 + a_1b_2 + a_0b_3)\,k. \end{aligned}$$

Доказать, что кольцо кватернионов действительно образует кольцо, но не поле. Какое из свойств поля в этом кольце не выполняется?

2.16. Доказать теорему 2.2.5.

2.17. Поле из трех элементов $GF(3)$ задано арифметическими таблицами

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Вычислить определитель следующей матрицы и показать, что ранг ее равен трем:

$$\mathbf{M} = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix}.$$

2.18. (ДПФ переставленной последовательности). Рассмотрим дискретное преобразование Фурье

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \quad k = 0, \ldots, n-1,$$

и предположим, что a взаимно просто с n. Пусть перестановка компонент вектора $\mathbf{v}$ задается равенством $v'_i = v_{((ai))}$. Доказать, что

$$V'_k = \sum_{i=1}^{n-1} \omega^{ik} v'_i$$

является перестановкой компонент вектора $\mathbf{V}$, задаваемой равенством $V'_k = V_{((bk))}$ для некоторого целого b, взаимно простого с n.

2.19. Год содержит самое большое 366 дней. Предположим, что все месяцы за исключением последнего содержат по 31 дню.
а. Можно ли однозначно определить порядковый номер дня в году по заданным его номерам в месяце и неделе?
б. Предположим далее, что месяц содержит 31 день, а неделя — 12 дней. Можно ли теперь однозначно определить порядковый номер дня в году по заданным его номерам в месяце и неделе?
в. Используя 31-дневный месяц и 12-дневную неделю, выписать формулу для вычисления порядкового номера дня в году через его номера в месяце и неделе.
г. Сделать несколько числовых примеров.

2.20. Сколько векторов содержится в векторном пространстве GF^n (2)?

2.21. Правильно ли утверждение о том, что если $\mathbf{x}$, $\mathbf{y}$ и $\mathbf{z}$ линейно независимы над $GF(q)$, то также линейно независимы векторы $\mathbf{x}+\mathbf{y}$, $\mathbf{y}+\mathbf{z}$ и $\mathbf{z}+\mathbf{x}$?

2.22. Доказать, что если S и T — два двумерных различных подпространства трехмерного пространства, то их пересечение образует одномерное подпространство.

2.23. Пусть S — произвольное конечное множество. Пусть G — множество всех подмножеств множества S. Для двух множеств A и B через $A \cup B$ обозначим множество всех элементов, которые принадлежат хотя бы одному из множеств A или B; через $A \cap B$ — множество всех элементов, которые принадлежат одновременно и A, и B; через $A - B$ — множество элементов, принадлежащих A, но не принадлежащих B.
а. Показать, что множество G с операцией объединения $\cup$ множеств в качестве групповой операции $*$ не является группой.
б. Операция симметрической разности множеств $\triangle$ определяется равенством

$$A \triangle B = (A - B) \cup (B - A).$$

Показать, что множество G с симметрической разностью в качестве групповой операции образует группу. Является ли она абелевой?
в. Показать, что множество G с операциями $\triangle$ и $\cap$ образует кольцо. Является ли это кольцо коммутативным? Содержит ли это кольцо единицу?

Замечания

В этой главе рассматривался обычный материал современной алгебры. Можно указать много учебников, в которых этот материал излагается более детально. В качестве легко усваиваемого вводного курса, по уровню достаточного для данной книги, мы рекомендуем книгу Биркгофа и Маклейна [1] (1941). Монография Ван дер Вардена [2] (1949, 1953) представляет собой курс более высокого уровня, адресованный в основном математикам и углубленно излагающий многие вопросы. Материал по линейной алгебре и теории матриц можно найти также и в учебниках, специально посвященных этим разделам алгебры. Особенно подходящей представляется книга Тралля и Торнгейма [3] (1957), так как в отличие от многих других книг в ней не предполагается, что рассматриваемое поле является полем вещественных или комплексных чисел [1]). В книге Полларда [7] в явном виде изложено понятие преобразования Фурье в произвольном поле.

Поля Галуа названы в честь Эвариста Галуа (1811—1832). Абелевы группы названы в честь Нильса Хенрика Абеля (1802—1829).

---

[1]) Не требующее предварительной подготовки хорошее изложение начал линейной алгебры и теории матриц можно найти также в книге: Головина Л. И. Линейная алгебра и некоторые ее приложения. — 4-е изд. — М.: Наука, 1985. Более углубленное изложение содержится в следующих книгах: Курош А. Г. Курс высшей алгебры. — 11-е изд. — М.: Наука, 1975; Мальцев А. И. Основы линейной алгебры. — 4-е изд. — М.: Наука, 1975. — *Прим. перев.*

Глава 3

БЫСТРЫЕ АЛГОРИТМЫ КОРОТКИХ СВЕРТОК

Наилучший известный способ эффективного вычисления сверток состоит в использовании теоремы о свертке и быстрого алгоритма преобразования Фурье. Этот способ обычно бывает достаточно удобен и часто работает с удовлетворительной скоростью, хотя существуют и лучшие методы. Однако в тех приложениях, в которых стоит заботиться о дальнейшем снижении вычислительных затрат, целесообразен переход к другим методам. В случае когда длина свертки мала, лучшими с точки зрения числа умножений и сложений являются алгоритмы Винограда вычисления свертки. Описание алгоритмов Винограда занимает основную часть данной главы. В следующих главах будет показано, как можно, соединяя вместе эти малые алгоритмы, строить алгоритмы для длинных сверток. При таком подходе алгоритмы для длинных сверток окажутся хорошими, только если хороши алгоритмы для коротких. Поэтому поиску алгоритмов коротких сверток уделяется большое внимание.

Детали алгоритмов свертки могут зависеть от конкретного поля, над которым вычисляется свертка, но общая идея построения таких алгоритмов от поля не зависит. Хотя, конечно, наиболее важные приложения имеют поля вещественных и комплексных чисел, мы описываем методы построения алгоритмов свертки применительно к любому представляющему интерес полю.

3.1. Циклические свертки и линейные свертки

В компактном виде линейная свертка записывается как произведение многочленов

$$s(x) = g(x)\, d(x).$$

Коэффициенты s_i этого многочлена даются равенствами

$$s_i = \sum_{k=0}^{N-1} g_{i-k} d_k, \quad i = 0, \ \ldots, \ L + N - 1,$$

где $\deg g(x) = L - 1$ и $\deg d(x) = N - 1$.

Прямые способы вычисления произведения многочленов содержат число умножений и сложений, примерно равное произведению степеней многочленов, LN; возможны, однако, другие способы вычисления такого произведения, содержащие меньшее число вычислений.

Циклическая свертка,

$$s(x) = g(x)\, d(x) \pmod{x^n - 1},$$

где $\deg g(x) = \deg d(x) = n - 1$, коэффициенты даются равенствами

$$s_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k, \quad i = 0, \ \ldots, \ n - 1,$$

и двойные скобки обозначают вычисления по модулю n, если вычислять ее прямо по выписанным формулам, содержит n^2 умножений и $n(n-1)$ сложений. Циклическую свертку можно также вычислять как линейную свертку с последующим приведением по модулю $x^n - 1$ [1]. Следовательно, эффективные способы вычисления линейной свертки приводят также к эффективным методам вычисления циклической свертки. Наоборот, эффективные методы вычисления циклической свертки можно легко превратить в эффективные методы вычисления линейной свертки.

Популярным способом вычисления циклической свертки является использование теоремы о свертке и дискретного преобразования Фурье. Согласно теореме о свертке в частотной области,

$$S_k = G_k D_k, \quad k = 0, \ \ldots, \ n - 1,$$

так что свертку можно вычислять, выполняя последовательно преобразование Фурье, поточечное умножение и обратное преобразование Фурье. Эта процедура иллюстрируется рис. 3.1. Средний блок содержит n комплексных умножений, что мало по сравнению с n^2. Если n имеет много делителей, то описываемое в следующей главе быстрое преобразование Фурье может привести к существенному уменьшению числа вычислений в первом и третьем блоках, сведя их к величинам, также малым по сравнению с n^2.

Приведенная на рис. 3.1 процедура вычисления циклической свертки относится к комплексному случаю. Поэтому естественно ожидать, что в случае вещественных последовательностей этот алгоритм сильнее, чем требуется, и его эффективность может быть повышена. Некоторая модификация позволяет почти тем же самым алгоритмом вычислять одновременно две вещественные свертки.

---

[1]) При $N = L$ линейную $(L \times N)$-свертку часто называют N-точечной линейной сверткой, а циклическую свертку по модулю $x^n - 1$ часто называют n-точечной циклической сверткой. — *Прим. перев.*

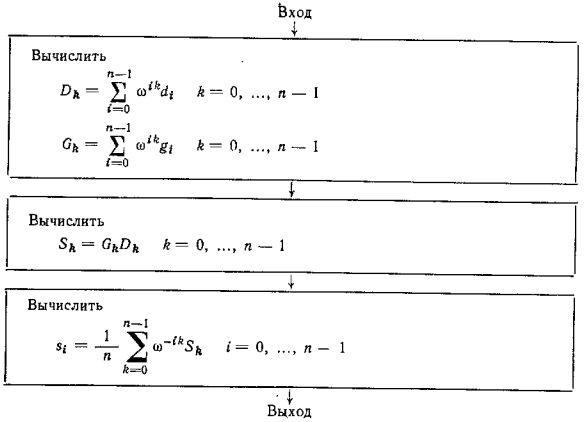


Рис. 3.1. Вычисление циклической свертки с помощью преобразования Фурье.

Вещественное преобразование Фурье обладает свойством симметрии (см. задачу 2.12)

$$V_k = V_{n-k}^*, \quad k = 0, \ldots, n-1.$$

Предположим, что $\mathbf{d}'$ и $\mathbf{d}''$ — вещественные векторы длины n и пусть компоненты комплексного вектора $\mathbf{d}$ длины n задаются правилом

$$d_i = d_i' + jd_i'', \quad i = 0, \ldots, n-1.$$

Тогда его дискретное преобразование Фурье удовлетворяет равенствам

$$\begin{aligned} D_k &= D_k' + jD_k'', \\ D_{n-k}^* &= D_k' - jD_k'', \end{aligned} \quad k = 0, \ldots, n-1.$$

Следовательно,

$$\begin{aligned} D_k' &= \frac{1}{2}[D_k + D_{n-k}^*], \\ D_k'' &= \frac{1}{2j}[D_k - D_{n-k}^*], \end{aligned} \quad k = 0, \ldots, n-1.$$

Эти формулы позволяют вычислить преобразование Фурье двух вещественных последовательностей, выполняя одно дискретное преобразование Фурье и некоторые дополнительные сложения.

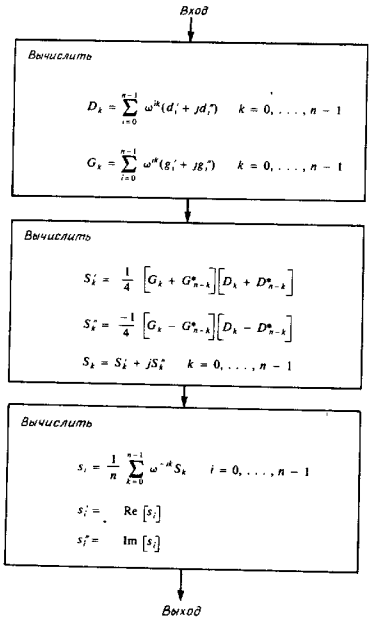


Рис. 3.2. Вычисление двух вещественных циклических сверток с помощью преобразования Фурье.

Описанную идею можно применять и в обратном порядке, начиная с двух комплексных преобразований двух вещественных последовательностей. Для заданных комплексных преобразований $\mathbf{D}'$ и $\mathbf{D}''$ пусть

$$D_k = D_k' + jD_k'', \quad k = 0, \ldots, n-1,$$

где в общем случае и D_k' и D_k'' являются комплексными числами. Тогда обратное преобразование Фурье приводит к равенствам

$$d_i = d_i' + jd_i'', \quad i = 0, \ldots, n-1,$$

из которых обе вещественные последовательности вычисляются немедленно.

Используя описанную идею, процедуру на рис. 3.1 можно заменить более эффективной процедурой одновременного вычисления двух вещественных сверток, представленной на рис. 3.2.

3.2. Алгоритм Кука—Тоома

Алгоритм Кука — Тоома является алгоритмом вычисления линейной свертки, который был разработан как метод умножения двух многочленов. Запишем линейную свертку в виде произведения двух многочленов

$$s(x) = g(x)\, d(x),$$

где $\deg d(x) = N - 1$ и $\deg g(x) = L - 1$. Степень многочлена $s(x)$ равна $N + L - 2$, так что этот многочлен однозначно определяется своими значениями в $N + L - 1$ различных точках. Пусть $\beta_0, \beta_1, \ldots, \beta_{L+N-1}$ — множество из $L + N - 1$ различных вещественных чисел. Если нам известны $s(\beta_k)$ для $k = 0, \ldots, N + L - 1$, то $s(x)$ можно вычислить по интерполяционной формуле Лагранжа. Согласно теореме 2.7.10, многочлен

$$s(x) = \sum_{i=0}^{n-1} s(\beta_i) \frac{\prod_{j \neq i} (x - \beta_j)}{\prod_{j \neq i} (\beta_i - \beta_j)}$$

является единственным многочленом степени $n - 1$, принимающим значения $s(\beta_k)$ при $x = \beta_k$ для всех $k = 0, \ldots, n - 1$. Идея алгоритма Кука — Тоома состоит в том, чтобы сначала вычислить величины $s(\beta_k)$ для $k = 0, \ldots, n - 1$, а затем воспользоваться интерполяцией Лагранжа.

Алгоритм Кука — Тоома приведен на рис. 3.3. Умножения даются равенствами

$$s(\beta_k) = g(\beta_k)\, d(\beta_k), \quad k = 0, \ldots, L + N - 2.$$

Всего имеется $L + N - 1$ таких равенств, так что здесь мы имеем $L + N - 1$ умножений, и, если разумно выбирать точки β_k, то полное число умножений будет исчерпываться этими умножениями. Имеются еще и другие умножения, а именно те, которые входят в вычисление величин $d(\beta_k)$ и $g(\beta_k)$, и те, которые участвуют в интерполяционной формуле Лагранжа. Но эти умножения представляют собой умножения на малые константы, и мы не будем их учитывать при подсчете полного числа умножений [1], хотя полностью их игнорировать нельзя.

[1]) На самом деле не учитывать можно только умножения на 0 и ± 1; но, как правило, за счет некоторых предварительных проделанных раз и навсегда вычислений удается свести рассматриваемые константы к трем указанным. — *Прим. перев.*

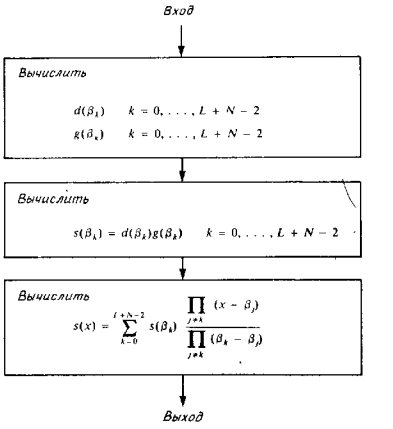


Рис. 3.3. Структура алгоритма Кука—Тоома.

Простейшим примером является линейная (2×2)-свертка: $d(x) = d_1 x + d_0$, $g(x) = g_1 x + g_0$ и $s(x) = d(x)\, g(x)$. К такому вычислению сводится прохождение двух отсчетов данных через КИО-фильтр с двумя отводами. Очевидный алгоритм выполнения такого вычисления содержит четыре умножения и одно сложение, но мы построим алгоритм с тремя умножениями, тремя сложениями. Может показаться, что эта задача слишком мала для каких-либо практических применений. Но на самом деле хороший алгоритм решения этой задачи является хорошим блоком, из которого можно строить тщательно проработанные алгоритмы. Таким образом, рассматриваемый пример представляет собой нечто большее, чем простую иллюстрацию алгоритма Кука — Тоома; он важен практически [1]).

В качестве первой попытки опишем алгоритм с тремя умножениями и пятью сложениями, а затем его улучшим. Так как

[1]) Первым, еще до Тоома (1963) и Кука (1966), способ выполнения (2×2)-свертки с тремя умножениями и четырьмя сложениями описал А. Карацуба [ДАН, 145 (1962), 293, 294]; его алгоритм состоит в следующем: $s_0 = d_0 g_0$, $s_2 = d_1 g_1$, $l = (d_0 + d_1)(g_0 + g_1)$, $s_1 = l - s_0 - s_2$. — *Прим. перев.*

степень многочлена $s(x)$ равна двум, то надо выбрать три точки. Выберем их равными $\beta_0 = 0$, $\beta_1 = 1$ и $\beta_2 = -1$. Тогда

$$d(\beta_0) = d_0, \qquad g(\beta_0) = g_0,$$
$$d(\beta_1) = d_0 + d_1, \qquad g(\beta_1) = g_0 + g_1,$$
$$d(\beta_2) = d_0 - d_1, \qquad g(\beta_2) = g_0 - g_1$$

и

$$s(\beta_0) = g(\beta_0)\, d(\beta_0),$$
$$s(\beta_1) = g(\beta_1)\, d(\beta_1),$$
$$s(\beta_2) = g(\beta_2)\, d(\beta_2),$$

так что требуются три умножения.

Если КИО-фильтр задается фиксированным многочленом $g(x)$, то константы $g(\beta_k)$ не надо каждый раз вычислять заново; их можно вычислить заранее один раз и запомнить. Коэффициенты многочлена $g(x)$ в запоминании не нуждаются.

Наконец, согласно интерполяционной формуле Лагранжа

$$s(x) = s(\beta_0)\, L_0(x) + s(\beta_1)\, L_1(x) + s(\beta_2)\, L_2(x),$$

где интерполяционные многочлены равны $L_0(x) = -x^2 + 1$, $L_1(x) = (1/2)(x^2 + x)$, $L_2(x) = (1/2)(x^2 - x)$. Это завершает описание алгоритма Кука — Тоома для вычисления $s(x)$. Но вычисления можно организовать в более компактной форме. Деления на 2 можно «похоронить» так, чтобы они не были видны. Для этого просто заменим константы $g(\beta_k)$ новыми, поглощающими этот делитель. Пусть

$$G_0 = g_0, \quad G_1 = (1/2)(g_0 + g_1), \quad G_2 = (1/2)(g_0 - g_1).$$

Так как в большинстве случаев многочлен $g(x)$ фиксирован, то эти константы можно вычислить заранее, и ничто не мешает нам определить их таким образом. Тогда

$$L_0(x) = -x^2 + 1, \quad L_1(x) = x^2 + x, \quad L_2(x) = x^2 - x.$$

Построенный алгоритм показан на рис. 3.4, *a* и в компактной форме может быть записан в матрично-векторных обозначениях в виде

$$\mathbf{s} = \mathbf{C}\{[\mathbf{Bg}] \cdot [\mathbf{Ad}]\},$$

где точкой обозначено покомпонентное умножение векторов **Bg** и **Ad**. Матричная запись дает удобную форму обозрения алгоритма, но вычисления, конечно, проводятся не посредством умножения матриц. Умножение на матрицы **A** и **C** выполняется как серия сложений. Возможная последовательность вычислений показана на рис. 3.4, b.

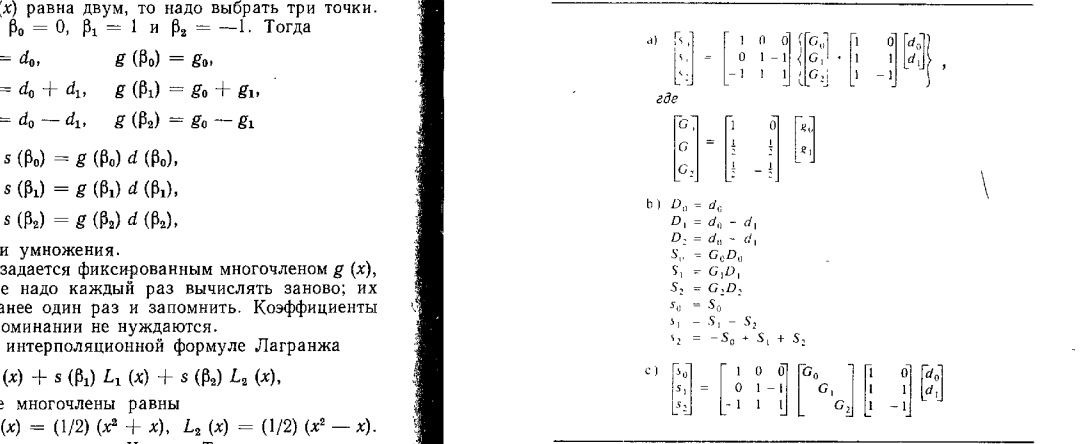


Рис. 3.4. Алгоритм Кука—Тоома вычисления линейной свертки 2-точечных векторов.

Алгоритм Кука — Тоома можно рассматривать как факторизацию матрицы. В рассматриваемом примере свертка может быть записана в виде

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} g_0 & 0 \\ g_1 & g_0 \\ 0 & g_1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}$$

или, кратко, $\mathbf{s} = \mathbf{T}\mathbf{d}$. На рис. 3.4c алгоритм Кука — Тоома для этой свертки переписан с использованием диагональной матрицы, также обозначенной G и содержащей элементы вектора G. Алгоритм теперь имеет форму

$$\mathbf{s} = \mathbf{CGAd},$$

что позволяет интерпретировать его как факторизацию матрицы $\mathbf{T} = \mathbf{CGA}$, где **A** — матрица предсложений, **C** есть матрица постсложений, а **G** — диагональная матрица, ответственная за все умножения. Число умножений равно размерности матрицы **G**. Такое представление алгоритма очень полезно для обозрения его структуры.

В общем случае линейная свертка может быть выражена в виде

$$\mathbf{s} = \mathbf{T}\mathbf{d},$$

где $\mathbf{d}$ — входной вектор длины N, $\mathbf{s}$ — выходной вектор длины $N + L - 1$ и $\mathbf{T}$ представляет собой $((N + L - 1) \times N)$-матрицу, элементами которой являются компоненты вектора g. Алгоритм Кука — Тоома в такой форме представляет собой алгоритм факторизации матрицы

$$\mathbf{T} = \mathbf{CGA},$$

где $\mathbf{G}$ — диагональная матрица, а матрицы $\mathbf{A}$ и $\mathbf{C}$ содержат только малые целые числа.

Алгоритм Кука — Тоома можно модифицировать к другому варианту, содержащему то же число умножений, но меньшее число сложений. Заметим, что $s_{L+N-2} = g_{L-1}d_{N-1}$. Для вычисления этого коэффициента необходимо одно умножение. Степень модифицированного многочлена

$$s(x) - s_{L+N-2}x^{L+N-2} = g(x)\,d(x) - s_{L+N-2}x^{L+N-2}$$

равна $L + N - 3$, так что использующее идеи алгоритма Кука — Тоома вычисление этого многочлена требует $L + N - 2$ умножений. Общее число умножений опять равно $L + N - 1$, но сложений теперь будет меньше.

Рассмотрим поведение этой модификации алгоритма Кука — Тоома при вычислении линейной (2×2)-свертки. Выберем $\beta_0 = 0$ и $\beta_1 = -1$. Тогда

$$d(\beta_0) = d_0, \qquad g(\beta_0) = g_0,$$
$$d(\beta_1) = d_0 - d_1, \quad g(\beta_1) = g_0 - g_1$$

и

$$t(\beta_0) = g(\beta_0)\,d(\beta_0) - g_1d_1\beta_0^2,$$
$$t(\beta_1) = g(\beta_1)\,d(\beta_1) - g_1d_1\beta_1^2,$$

где $t(x) = g(x)\,d(x) - g_1d_1x^2$. Согласно интерполяционной формуле Лагранжа,

$$s(x) - g_1d_1x^2 = t(\beta_1)\,L_1(x) + t(\beta_0)\,L_0(x),$$

где $L_0(x) = x + 1$ и $L_1(x) = -x$.

Объединение всех этих фрагментов дает искомый алгоритм [1]):

$$s(x) = g_1d_1x^2 + [-(d_0 - d_1)\,(g_0 - g_1) + g_1d_1 + g_0d_0]\,x + g_0d_0.$$

Алгоритм содержит три сложения и три умножения. Его матричная форма выписана на рис. 3.5, который полезно сравнить с рис. 3.4.

[1]) Это приведенный на стр. 85 алгоритм А. Карацубы (1962), в котором не учтено в числе сложений вычисление $g_0 - g_1$. — *Прим. перев.*

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & (g_0 - g_1) & \\ & & g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}$$

Рис. 3.5. Улучшенный алгоритм Кука—Тоома вычисления линейной (2×2)-свертки.

Алгоритм Кука — Тоома эффективен по числу умножений; но если объем задачи растет, то число необходимых сложений также начинает быстро расти. Это происходит потому, что хороший выбор чисел β_k в виде 0, 1 и —1 уже сделан, и с ростом задачи мы должны использовать также ±2, ±3 и другие малые целые числа. При этом матрицы $\mathbf{A}$ и $\mathbf{C}$ будут содержать эти числа. Конечно, умножения на них можно заменить соответствующим числом сложений, но это резонно делать только в случае, когда выбираемые числа малы. Поэтому алгоритм Кука — Тоома становится слишком громоздок при вычислении сверток, больших чем 3×4 или 4×4. Для бо́льших задач надо пользоваться описываемыми в следующем разделе алгоритмами Винограда вычисления сверток или итерировать малые алгоритмы Кука — Тоома, используя изучаемые в гл. 7 гнездовые методы.

Имеется еще один подход к пониманию алгоритма Кука — Тоома, и эта альтернатива перекидывает мостик к следующему

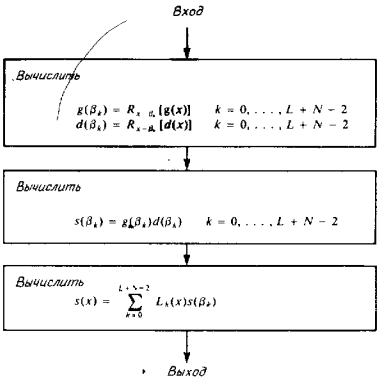


Рис. 3.6. Другое описание алгоритма Кука—Тоома.

разделу. Вместо того чтобы выбирать множество различных чисел $\{\beta_0, \beta_1, \ldots, \beta_{L+N-2}\}$, выберем множество многочленов $\{x - \beta_0, x - \beta_1, \ldots, x - \beta_{L+N-2}\}$. Тогда можно, как на рис. 3.6, записать

$$g(\beta_k) = R_{x-\beta_k}[g(x)], \qquad d(\beta_k) = R_{x-\beta_k}[d(x)].$$

Это просто несколько более сложный способ сказать то же, что говорилось и раньше. Преимущество, реализуемое в следующем разделе, состоит в замене многочленов первой степени многочленами более высоких степеней, что увеличивает конструктивные возможности. С этой более изощренной точки зрения интерполяционную формулу Лагранжа можно рассматривать как некоторую форму обращения китайской теоремы об остатках для того частного случая, когда в качестве многочленов-модулей выбраны только многочлены первой степени. В более общем случае интерполяционную формулу Лагранжа приходится отвергнуть, отдавая преимущество китайской теореме об остатках.

3.3. Алгоритмы Винограда вычисления коротких сверток

Предположим, что требуется вычислить

$$s(x) = g(x)\, d(x) \pmod{m(x)},$$

где $m(x)$ — фиксированный многочлен степени n над полем F, а $g(x)$ и $d(x)$ — многочлены над этим же полем, причем их степени меньше n. Задача вычисления линейной свертки $s(x) = g(x)\, d(x)$ может быть вложена в эту задачу, если в качестве n выбрать целое число, большее степени многочлена $s(x)$, а в качестве $m(x)$ — произвольный многочлен этой степени n. Тогда задача

$$s(x) = g(x)\, d(x) \pmod{m(x)}$$

становится тривиальной переформулировкой задачи вычисления линейной свертки, так как приведение по модулю $m(x)$ не меняет $s(x)$, коль скоро степень $m(x)$ больше степени $s(x)$. Это позволяет включить задачу о линейной свертке в общую задачу, рассматриваемую в данном разделе. Выбирая $m(x)$ равным $x^n - 1$, мы включаем сюда также задачу вычисления циклической свертки,

$$s(x) = g(x)\, d(x) \pmod{x^n - 1}.$$

Мы хотим заменить задачу вычисления

$$s(x) = g(x)\, d(x) \pmod{m(x)}$$

некоторым множеством задач с меньшим числом вычислений. Чтобы разбить задачу на подзадачи, разложим многочлен $m(x)$ на взаимно простые над некоторым подполем поля F многочлены:

$$m(x) = m^{(0)}(x)\, m^{(1)}(x) \ldots m^{(K-1)}(x).$$

Обычно если F — поле вещественных или комплексных чисел, то в качестве подполя разложения выбирается поле рациональных чисел, и мы будем ссылаться на такой выбор, хотя теория допускает и другие подполя. (Если свертка вычисляется в конечном поле $GF(p^m)$, то в качестве подполя разложения обычно выбирается простое подполе $GF(p)$). Процедура ставит своей целью минимизацию числа умножений в поле F, не пытаясь минимизировать число умножений в подполе. В большинстве практических случаев этими умножениями в подполе оказываются умножения на малые целые числа, обычно на -1, 0 или 1, которые тривиальны. Начиная с данного момента, мы не будем учитывать умножения на рациональные числа в полном числе умножений, хотя практически всегда надо проверять, являются ли эти рациональные числа на самом деле малыми целыми числами.

Быстрые алгоритмы свертки основаны на вычетах

$$s^{(k)}(x) = R_{m^{(k)}(x)}[s(x)], \quad k = 0, \ldots, K-1.$$

Согласно китайской теореме для многочленов, многочлен $s(x)$ можно вычислить по системе остатков в соответствии с формулой

$$s(x) = a^{(0)}(x)\, s^{(0)}(x) + \cdots + a^{(K-1)}(x)\, s^{(K-1)}(x) \pmod{m(x)},$$

где $a^{(0)}(x), \ldots, a^{(K-1)}(x)$ — соответствующие многочлены с рациональными коэффициентами. Разобьем это вычисление на три шага. Сначала вычисляются остатки $d^{(k)}(x) = R_{m^{(k)}(x)}[d(x)]$ и $g^{(k)}(x) = R_{m^{(k)}(x)}[g(x)]$ для всех $k = 0, \ldots, K-1$. Эти вычисления не содержат умножений. Затем вычисляются величины

$$\begin{aligned} s^{(k)}(x) &= g(x)\, d(x) \pmod{m^{(k)}(x)} = \\ &= R_{m^{(k)}(x)}\{R_{m^{(k)}(x)}[g(x)] \cdot R_{m^{(k)}(x)}[d(x)]\} = \\ &= R_{m^{(k)}(x)}[g^{(k)}(x)\, d^{(k)}(x)]. \end{aligned}$$

Наконец, вычисляется многочлен

$$s(x) = a^{(0)}(x)\, s^{(0)}(x) + \cdots + a^{(K-1)}(x)\, s^{(K-1)}(x) \pmod{m(x)}.$$

Так как коэффициенты всех многочленов $a^{(k)}(x)$ являются рациональными числами, то последний шаг также не содержит умножений.

Структура алгоритма Винограда для вычисления свертки показана на рис. 3.7. Умножения возникают только на втором шаге при вычислении коротких сверток, задаваемых произведениями многочленов, $g^{(k)}(x)\, d^{(k)}(x)$; так как число коэффициентов у многочленов $g^{(k)}(x)$ и $d^{(k)}(x)$ равно степени много-

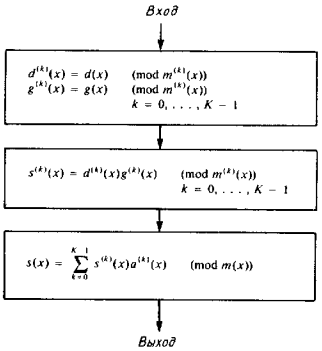


Рис. 3.7. Структура алгоритма Винограда для коротких сверток.

члена $m^{(k)}(x)$, то полное число умножений, необходимых для стандартного способа вычисления этих произведений, равно $\sum_{k=0}^{K-1} [\deg m^{(k)}(x)]^2$. Это дает существенное уменьшение числа необходимых умножений. Позже мы увидим, что использование той же самой идеи разбиения на еще более мелкие подзадачи позволяет добиться улучшения и при вычислении этих составляющих коротких сверток.

На рис. 3.8 дается поучительное сравнение алгоритма Винограда вычисления свертки и алгоритма, основанного на дискретном преобразовании Фурье. Для того чтобы сделать это сравнение более наглядным, алгоритм Винограда выписан в приведенных выше матричных обозначениях. После группировки членов и перехода к матричным обозначениям этот алгоритм записывается в виде равенства

$$\mathbf{s} = \mathbf{C}\{[\mathbf{Bg}] \cdot [\mathbf{Ad}]\},$$

где точка обозначает покомпонентное произведение векторов **Bg** и **Ad**. Из этого сравнения видно, что алгоритм Винограда является обобщением метода вычисления сверток с помощью преобразования Фурье.

В качестве примера рассмотрим вычисление свертки 3-точечного вектора с 2-точечным вектором. Пусть

$$g(x) = g_1 x + g_0,$$
$$d(x) = d_2 x^2 + d_1 x + d_0.$$

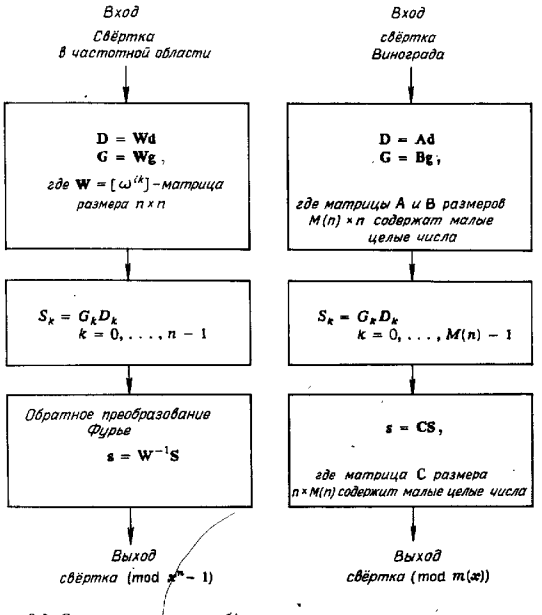


Рис. 3.8. Сравнение двух способов вычисления сверток.

Прямое вычисление содержит шесть умножений и два сложения. Мы сначала построим алгоритм, содержащий пять умножений и восемь сложений. Это не очень хороший алгоритм, но его построение является поучительным. Позже будет приведен алгоритм, который лучше и содержит четыре умножения и семь сложений.

Степень линейной свертки $s(x) = g(x)\,d(x)$ равна трем. Выберем

$$m(x) = x(x-1)(x^2+1) = m^{(0)}(x)\,m^{(1)}(x)\,m^{(2)}(x).$$

Множители взаимно просты; возможно и другое разложение многочлена $m(x)$, но для иллюстрации метода мы остановимся

на выбранном разложении. Вычеты равны

$$g^{(0)}(x) = g_0, \qquad d^{(0)}(x) = d_0,$$
$$g^{(1)}(x) = g_1 + g_0, \qquad d^{(1)}(x) = d_2 + d_1 + d_0,$$
$$g^{(2)}(x) = g_1 x + g_0, \qquad d^{(2)}(x) = d_1 x + (d_0 - d_2).$$

Следовательно,

$$s^{(0)}(x) = g_0 d_0,$$
$$s^{(1)}(x) = (g_1 + g_0)(d_2 + d_1 + d_0),$$
$$s^{(2)}(x) = (g_1 x + g_0)(d_1 x + (d_0 - d_2)) \pmod{x^2 + 1}.$$

При вычислении $s^{(0)}(x)$ требуется одно умножение, при вычислении $s^{(1)}(x)$ требуется одно умножение и, как мы увидим, при вычислении $s^{(2)}(x)$ требуются три умножения.

Вычисление $s^{(2)}(x)$ сводится к вычислению двух величин

$$s_0^{(2)} = g_0^{(2)} d_0^{(2)} - g_1^{(2)} d_1^{(2)},$$
$$s_1^{(2)} = g_0^{(2)} d_1^{(2)} + g_1^{(2)} d_0^{(2)}$$

(которые случайно имеют структуру произведения комплексных чисел). Алгоритм этой части вычислений дается равенством

$$\begin{bmatrix} s_0^{(2)} \\ s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} g_0^{(2)} & & \\ & (g_1^{(2)} - g_0^{(2)}) & \\ & & (g_1^{(2)} + g_0^{(2)}) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0^{(2)} \\ d_1^{(2)} \end{bmatrix}$$

и содержит три умножения.

Последний шаг состоит в вычислении $s(x)$ по формуле $s(x) = a^{(0)}(x)s^{(0)}(x) + a^{(1)}(x)s^{(1)}(x) + a^{(2)}(x)s^{(2)}(x) \pmod{x^4 - x^3 + x^2 - x}$, где $a^{(0)}(x)$, $a^{(1)}(x)$ и $a^{(2)}(x)$ вычисляются по китайской теореме об остатках следующим образом. Воспользуемся соотношением

$$n^{(k)}(x)\, m^{(k)}(x) + N^{(k)}(x)\, M^{(k)}(x) = 1$$

и построим следующую таблицу:

k	$m^{(k)}(x)$	$M^{(k)}(x)$	$n^{(k)}(x)$	$N^{(k)}(x)$
0	x	$x^3 - x^2 + x - 1$	$x^2 - x + 1$	-1
1	$x - 1$	$x^3 + x$	$-\frac{1}{2}(x^2 + x + 2)$	$\frac{1}{2}$
2	$x^2 + 1$	$x^2 - x$	$-\frac{1}{2}(x - 2)$	$\frac{1}{2}(x - 1)$

Тогда $a^{(k)}(x) = N^{(k)}(x)\, M^{(k)}(x)$. Следовательно,

$$s(x) = -(x^3 - x^2 + x - 1)\, s^{(0)}(x) + \frac{1}{2}(x^3 + x)\, s^{(1)}(x) + \frac{1}{2}(x^3 - 2x^2 + x)\, s^{(2)}(x) \pmod{x^4 - x^3 + x^2 - x}.$$

Последнее равенство может быть перезаписано в виде матричного равенства

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 \\ 1 & 0 & -2 & 0 \\ -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} s_0^{(0)} \\ \frac{1}{2} s_0^{(1)} \\ \frac{1}{2} s_0^{(2)} \\ \frac{1}{2} s_1^{(2)} \end{bmatrix}.$$

Теперь для построения искомого алгоритма надо собрать вместе его отдельные части. Всего имеются пять умножений, которые мы выпишем в выбранной единообразной форме

$$S_k = G_k D_k, \quad k = 0, \ldots, 4.$$

Определим новые обозначения, соответствующие этой форме. Используя проделанные вычисления, для вектора **D** получаем определение

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0^{(0)} \\ d_0^{(1)} \\ d_0^{(2)} \\ d_1^{(2)} \end{bmatrix} =$$
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}.$$

Вектор **G** определяется аналогично, но в его определение мы включим знаменатели, которые должны появиться в матрице постсложений. Поэтому в данном ниже определении вектора **G** появляется самая левая матрица, содержащая извлеченные из

матрицы постсложений множители 1/2. Таким образом,

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ & \frac{1}{2} & & & \\ & & \frac{1}{2} & & \\ & & & \frac{1}{2} & \\ & & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}$$

Так как вычисление вектора $\mathbf{G}$ является предваряющим алгоритм вычислением, то эту матрицу можно в дальнейшем просто отбросить.

Наконец, матрица постсложений имеет вид

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 \\ 1 & 0 & -2 & 0 \\ -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & 1 & -1 \\ 1 & 0 & -2 & 0 & 2 \\ -1 & 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix}.$$

Описанный алгоритм в матричной форме приведен на рис. 3.9. Порядок выполнения сложений в такой форме алгоритма не указан. Читатель может сам поэкспериментировать с выбором порядка сложений, минимизируя их число; никакой специальной теории выполнения такой минимизации не разработано. Легко выполнить все предсложения с помощью четырех вещественных сложений. Постсложения можно реализовать с помощью следующих восьми вещественных сложений:

$$s_0 = S_0, \qquad s_2 = c_1 + c_1 + S_0,$$
$$c_1 = S_4 - S_2, \qquad s_1 = S_1 + c_3 - s_2,$$
$$c_3 = S_3 + S_4, \qquad s_3 = c_3 - S_0 + S_1.$$

Это завершает рассматриваемый пример построения алгоритма Винограда для свертки, но построенный алгоритм можно улучшить. Более общая форма алгоритма Винограда соответствует

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & 1 & -1 \\ 1 & 0 & -2 & 0 & 2 \\ -1 & 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} G_0 & & & & \\ & G_1 & & & \\ & & G_2 & & \\ & & & G_3 & \\ & & & & G_4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}$$

Рис. 3.9. Пример алгоритма Винограда вычисления свертки.

выбору многочлена $m(x)$ меньшей степени. Это приводит к неправильному вычислению свертки, но ошибка легко подправляется с помощью нескольких дополнительных вычислений.

Согласно алгоритму деления для многочленов можно записать

$$s(x) = Q(x)\, m(x) + R_{m(x)}\,[s(x)].$$

Мы уже рассмотрели случай $\deg m(x) > \deg s(x)$, в котором частное $Q(x)$ тождественно равно нулю. Если $\deg m(x) \leqslant \deg s(x)$, то алгоритм Винограда позволяет вычислить $s(x)$ по модулю многочлена $m(x)$. Член $Q(x)\, m(x)$ представляет собой погрешность, которую можно вычислить дополнительно и прибавить к результату. Простейшим является случай, когда $\deg m(x) = \deg s(x)$. Тогда многочлен $Q(x)$ является константой, так как его степень должна быть равной нулю. Если $m(x)$ — приведенный многочлен степени n, то, очевидно, $Q(x) = s_n$, где s_n — коэффициент при x_n в многочлене $s(x)$. Следовательно,

$$s(x) = s_n m(x) + R_{m(x)}\,[s(x)],$$

и s_n легко вычисляется как произведение старших коэффициентов многочленов $g(x)$ и $d(x)$.

Эту модификацию можно формально включить в основной алгоритм Винограда для вычисления свертки путем замены многочлена $m(x)$ формальным выражением $m(x)$ $(x - \infty)$. Утверждение

$$s(x) = s(x) \pmod{m(x)\,(x - \infty)}$$

представляет собой просто удобное сокращение для данного выше точного утверждения.

Вернемся к предыдущему примеру и применим описанную модификацию алгоритма к вычислению свертки

$$s(x) = (g_1 x + g_0)(d_2 x^2 + d_1 x + d_0).$$

В качестве модуля выберем теперь многочлен $x\,(x - 1)\,(x + 1)\,(x - \infty)$. Алгоритм будет содержать четыре умножения;

множителю $(x - \infty)$ соответствует произведение $g_1 d_2$, а остальные три умножения — это произведения вычетов по модулям x, $x - 1$ и $x + 1$. Сразу видно, что

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \left\{ \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} \right\} \cdot \left\{ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} \right\}.$$

Коэффициенты многочлена $s(x)$ вычисляются в соответствии с китайской теоремой об остатках:

$$s(x) = a^{(0)}(x) S_0 + a^{(1)}(x) S_1 + a^{(2)}(x) S_2 + x(x-1)(x+1) S_3 =$$

$$= (-x^2 + 1) S_0 + \left(\frac{1}{2} x^2 + \frac{1}{2} x\right) S_1 + \left(\frac{1}{2} x^2 - \frac{1}{2} x\right) S_2 +$$

$$+ (x^3 - x) S_3.$$

Следовательно,

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_0 \\ \frac{1}{2} S_1 \\ \frac{1}{2} S_2 \\ S_3 \end{bmatrix}.$$

Окончательная формулировка алгоритма приведена на рис. 3.10. Алгоритм содержит четыре умножения и семь сложений при условии, что диагональная матрица вычисляется заранее.

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & (g_0 - g_1) & \\ & & g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}$$

Рис. 3.10. Другой пример алгоритма Винограда вычисления свертки.

3.4. Построение алгоритмов коротких линейных сверток

Рассмотренный в предыдущем разделе метод Винограда построения коротких сверток приводит к хорошим алгоритмам. Однако следует подчеркнуть, что эта конструкция не исчерпывает всех возможных путей построения хороших алгоритмов.

Иногда хорошие алгоритмы удается получить с помощью удачной факторизации. Рассмотрим следующие тождества:

$$g_0 d_0 = g_0 d_0,$$

$$g_0 d_1 + g_1 d_0 = (g_0 + g_1)(d_0 + d_1) - g_0 d_0 - g_1 d_1,$$

$$g_0 d_2 + g_1 d_1 + g_2 d_0 = (g_0 + g_2)(d_0 + d_2) - g_0 d_0 + g_1 d_1 - g_2 d_2,$$

$$g_1 d_2 + g_2 d_1 = (g_1 + g_2)(d_1 + d_2) - g_1 d_1 - g_2 d_2,$$

$$g_2 d_2 = g_2 d_2.$$

Эти тождества позволяют вычислить коэффициенты свертки $s(x) = (g_0 + g_1 x + g_2 x^2)(d_0 + d_1 x + d_2 x^2)$ с помощью шести умножений и десяти сложений. Этот алгоритм, однако, не может быть построен с помощью китайской теоремы об остатках. Матричная форма записи алгоритма дается равенством

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} g_0 & & & & & \\ & g_1 & & & & \\ & & g_2 & & & \\ & & & (g_0 + g_1) & & \\ & & & & (g_0 + g_2) & \\ & & & & & (g_1 + g_2) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}.$$

Этот алгоритм следует сравнить с «наивным» алгоритмом, содержащим девять умножений и четыре сложения, и с «оптимальным» алгоритмом, который вскоре будет построен и содержит пять умножений и 20 сложений. Нельзя сказать, какой из этих алгоритмов лучше. Это зависит от области применения. Позже мы будем строить большие алгоритмы из малых так, что и число умножений, и число сложений большого алгоритма будут в основном зависеть от числа умножений в малом алгоритме. В этой ситуации число сложений малого алгоритма не играет существенной роли.

На рис. 3.11 затабулированы характеристики некоторых алгоритмов линейных сверток; на рис. 3.12 приведены некоторые из них.

Оптимальный алгоритм линейной (3×3)-свертки выписан в виде примера на рис. 3.12. Оптимальный по числу умножений алгоритм содержит пять умножений. Он подсказан описанным в разд. 3.8 алгоритмом Кука — Тоома, который также содержит пять умножений. По существу он совпадает с алгоритмом Кука — Тоома и получается выбором многочлена

$$m(x) = x(x - 1)(x + 1)(x - 2)(x - \infty).$$

Так как все множители являются многочленами первой степени, то всего имеется пять умножений. (Если заменить один из множи-

$$s(x) = g(x)\,d(x)$$
$$\deg g(x) = L - 1$$
$$\deg d(x) = N - 1$$

Длина n		Вещественные свертки	
L	N	Число вещественных умножений	Число вещественных сложений
2	2	3	3
2	2	4	7
3	3	5	20
3	3	6	10
3	3	9	4
4	4	7	41
4	4	9	15
		Комплексные свертки	
		Число комплексных умножений	Число комплексных сложений
2	2	3	3
3	3	5	15
4	4	7	

Рис. 3.11. Характеристики некоторых алгоритмов вычисления коротких линейных сверток.

Линейная свертка 2×3
3 умножения, 3 сложения

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_0 + g_1 \\ g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}$$

Линейная свертка 3×3
5 умножений, 20 сложений

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ -1 & 2 & -2 & -1 & 2 \\ -2 & 1 & 3 & 0 & -1 \\ 1 & -1 & -1 & 1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} g_0 \\ \frac{1}{2}(g_0 + g_1 + g_2) \\ \frac{1}{6}(g_0 - g_1 + g_2) \\ \frac{1}{6}(g_0 + 2g_1 + 4g_2) \\ g_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Предсложения	Постсложения
$t_1 = d_1 + d_2$	$s_0 = S_0 + S_0$
$t_2 = d_2 - d_1$	$T_1 = S_1 + S_1$
$D_0 = d_0$	$T_2 = S_2 + S_2$
$D_1 = d_0 + t_1$	$T_3 = S_4 + S_4$
$D_2 = d_0 + t_2$	$T_4 = T_3 - S_0 - S_3$
$D_3 = t_1 + t_1 + t_2 + D_1$	$T_5 = S_1 + S_2$
$D_4 = d_2$	$s_1 = T_1 - T_2 + T_4$
	$s_2 = -s_0 + T_2 + T_5 - S_4$
	$s_3 = -T_4 - T_5$
	$s_4 = S_4$

Рис. 3.12. Некоторые алгоритмы вычисления линейных сверток.

телей многочленом второй степени, то получим шесть умножений, но меньшее число сложений.) Выходной многочлен дается равенством

$$s(x) = R_{x(x-1)(x+1)(x-2)}[g(x)\,d(x)] + g_2 d_2 x(x-1)(x+1)(x-2).$$

Остатки определяются матричными уравнениями

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix},$$

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix},$$

где с целью сохранения стройности и упорядоченности принятых обозначений G_4 и D_4 формально определяются как вычеты по модулю многочлена $(x - \infty)$. Таким образом, $S_k = G_k D_k$ для $k = 0, \ldots, 4$. Наконец, используя китайскую теорему об остатках, для многочлена $s(x)$ получаем выражение

$$s(x) = \frac{1}{2} S_0 (x^3 - 2x^3 - x + 2) - \frac{1}{2} S_1 (x^3 - x^2 - 2x) +$$
$$+ \frac{1}{6} S_2 (-x^3 + 3x^2 - 2x) + \frac{1}{6} S_3 (x^3 - x) + S^4 (x^4 - 2x^3 + x^2 + 2x).$$

Запись последнего шага в матричном виде дается равенством

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ -1 & 2 & -2 & -1 & 2 \\ -2 & 1 & 3 & 0 & -1 \\ 1 & -1 & -1 & 1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} S_0 \\ \frac{1}{2} S_1 \\ \frac{1}{6} S_2 \\ \frac{1}{6} S_3 \\ S_4 \end{bmatrix}.$$

Постоянные множители можно похоронить в константах диагональной матрицы, переопределяя величины S_k и G_k. Для выполнения свертки требуется пять умножений. Матрица предсложений может быть реализована семью сложениями, а матрица постсложений 13 сложениями.

Окончательная форма алгоритма показана на рис. 3.12. Хотя алгоритм построен для поля вещественных чисел, он пригоден и в поле комплексных чисел: пять умножений становятся пятью комплексными умножениями, а 20 сложений становятся 20-ю комплексными сложениями.

Можно также специально строить алгоритм над полем комплексных чисел. Он также будет содержать пять комплексных умножений, но число комплексных сложений уменьшится. Выберем многочлен

$$s(x) = x(x-1)(x+1)(x-j)(x+j).$$

Вычисление остатков дается равенствами

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & j & -1 \\ 1 & -j & -1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix},$$

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & j & -1 \\ 1 & -j & -1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}.$$

Китайская теорема об остатках (или интерполяционная формула Лагранжа) дает

$$s(x) = S_0(-x^4+1) + \frac{1}{4}S_1(x^4+x^3+x^2+x) + \frac{1}{4}S_2(x^4-x^3+x^2-x) + \frac{1}{4}S_3(x^4+jx^3-x^2-jx) + \frac{1}{4}S_4(x^4-jx^3-x^2+jx).$$

В матричной форме это равенство записывается в виде

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -j & j \\ 0 & 1 & 1 & -1 & -1 \\ 0 & 1 & -1 & j & -j \\ -1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} S_0 \\ \frac{1}{4}S_1 \\ \frac{1}{4}S_2 \\ \frac{1}{4}S_3 \\ \frac{1}{4}S_4 \end{bmatrix}.$$

Множители 1/4 можно спрятать в диагональной матрице. Алгоритм содержит пять комплексных умножений, эквивалент шести комплексных сложений в матрице предсложений и эквивалент девяти комплексных сложений в матрице постсложений.

Процедуру Винограда можно использовать для построения сверток и большей длины, чем те, что приведены на рис. 3.12; однако построенные таким образом большие алгоритмы, как правило, громоздки и содержат слишком много сложений. Вместо этого можно строить большие алгоритмы из малых. Есть несколько способов построения алгоритмов данных сверток из

алгоритмов коротких сверток. Некоторые из них, а именно гнездовой метод и метод Агарвала — Кули, будут рассмотрены в гл. 7. Такие алгоритмы содержат несколько больше умножений, чем требуется для алгоритма Винограда, но число сложений у них существенно меньше.

3.5. Вычисление произведения многочленов по модулю некоторого многочлена

Мы уже видели, как можно разбивать задачу вычисления циклической свертки или задачу вычисления линейной свертки на несколько подзадач меньшего размера. Меньшие задачи опять имеют вид

$$s(x) = g(x)\,d(x) \pmod{m(x)},$$

где $\deg m(x) = n$ и степени обоих многочленов $g(x)$ и $d(x)$ меньше n, т. е. форму произведения многочленов в кольце многочленов по модулю $m(x)$. Для того чтобы построить хороший алгоритм исходной линейной свертки, необходимо построить хорошие алгоритмы для каждого из меньших произведений многочленов. Достаточно рассмотреть только случай неприводимого многочлена $m(x)$, так как в противном случае разложение можно продолжить дальше.

Как будет доказано в разд. 3.8, для неприводимого многочлена $m(x)$ степени n число умножений, необходимых для вычисления $s(x)$, равно $2n-1$. Обычно в алгоритме с наименьшим возможным числом умножений требуется большое число сложений. В практически приемлемых алгоритмах должен достигаться разумный баланс между числом умножений и числом сложений. В данном разделе рассматриваются практические аспекты построения таких алгоритмов.

Наиболее прямой метод состоит в вычислении линейной свертки $g(x)\,d(x)$ (на что требуется по меньшей мере $2n-1$ умножений) и дальнейшем приведении результата по модулю $m(x)$ (что вообще не требует умножений). Это на первый взгляд может показаться парадоксальным, так как мы предлагали вычислять линейную свертку путем разбиения на подзадачи, соответствующие неприводимым многочленам, а теперь сводим задачу обратно к линейным сверткам. Однако этим маневрированием «вперед-назад» мы существенно снижаем степень линейной свертки, что и улучшает ситуацию.

Мы уже рассматривали произведение многочленов

$$s(x) = g(x)\,d(x) \pmod{x^2+1},$$

которое формально совпадает с комплексным умножением. Сейчас пришло время построить этот алгоритм. Одновременно мы будем рассматривать произведения многочленов

$$s(x) = g(x)\,d(x) \pmod{x^2}$$

и

$$s(x) = g(x)\,d(x) \pmod{x^2 + x + 1}.$$

Во всех этих примерах $g(x) = g_1x + g_0$ и $d(x) = d_1x + d_0$. Мы будем модифицировать описанную в разд. 3.2 линейную (2×2)-свертку. Алгоритм этой линейной свертки в полиномиальной форме записывается в виде равенства

$$s(x) = g(x)\,d(x) = g_1d_1x^2 + [g_1d_1 + g_0d_0 - (g_1 - g_0)(d_1 - d_0)]x + g_0d_0$$

и содержит три умножения.

Наша задача состоит в вычислении этого произведения по трем модулям: $x^2 + 1$, x^2 и $x^2 + x + 1$. Соответствующие три произведения получаются простой заменой x^2 на -1, 0 и $-x - 1$ соответственно. А именно,

$$s(x) = g(x)\,d(x) \pmod{x^2 + 1} =$$
$$= [g_1d_1 - g_0d_0 - (g_1 - g_0)(d_1 - d_0)]\,x + g_0d_0 - g_1d_1,$$

$$s(x) = g(x)\,d(x) \pmod{x^2} =$$
$$= [g_1d_1 - g_0d_0 - (g_1 - g_0)(d_1 - d_0)]\,x + g_0d_0,$$

$$s(x) = g(x)\,d(x) \pmod{x^2 + x + 1} =$$
$$= [g_0d_0 - (g_1 - g_0)(d_1 - d_0)]\,x + g_0d_0 - g_1d_1.$$

В матричной форме эти три алгоритма соответственно имеют вид

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_1 & \\ & & g_0 - g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix},$$

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_1 & \\ & & g_0 - g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}$$

и

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_1 & \\ & & g_0 - g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}.$$

Второй из этих алгоритмов не очень хорош. Следуя по формальному пути, мы построили алгоритм, характеристики которого хуже следующего эвристического алгоритма, содержащего только одно сложение:

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_0 & \\ & & g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}.$$

Характеристики некоторых алгоритмов вычисления произведения многочленов по модулю неприводимого многочлена затабулированы на рис. 3.13. В некоторых случаях указаны характеристики двух алгоритмов решения одной и той же задачи. Например, для модуля $x^4 + 1$ указан алгоритм, содержащий 9 умножений и 15 сложений, и алгоритм, содержащий 7 умножений

Простой многочлен $p(x)$	Число умножений	Число сложений
$x^2 + x + 1$	3	3
$x^4 + 1$	9	15
	7	41
$x^4 + x^3 + x^2 + x + 1$	9	16
	7	46
$x^6 + x^3 + 1$	15	39
$x^8 + x^5 + x^4 + x^3 + x^2 + x + 1$	15	53
$x^{10} + x^9 + 1$	75	267

Рис. 3.13. Характеристики некоторых алгоритмов вычисления произведения многочленов по модулю простых многочленов.

и 41 сложение. При выборе того или иного алгоритма надо оценивать ситуацию и понимать, как данный алгоритм вкладывается в алгоритм большей задачи. Часто случается, что несущественные преимущества в числе умножений в малом алгоритме приводят к большим преимуществам, когда этот алгоритм используется в виде блока в большом алгоритме, в то время как большие потери в числе сложений малого алгоритма приводят лишь к незначительным потерям в числе сложений большого алгоритма. Так, например, для модуля $x^4 + 1$ алгоритм с семью умножениями может на самом деле дать больше, чем это может показаться.

3.6. Построение алгоритмов коротких циклических сверток

Можно построить также и библиотеку хороших алгоритмов для вычисления коротких циклических сверток. Эти алгоритмы можно строить как по методу Винограда построения коротких

сверток, так и другими способами. Для построения алгоритма по методу Винограда надо быть уверенным в том, что каждое из рассматриваемых малых произведений многочленов может быть вычислено хорошим алгоритмом. Для этих подзадач мы будем пользоваться уже построенными в предыдущем разделе алгоритмами.

В настоящем разделе в качестве примеров будут построены несколько хороших алгоритмов. Некоторые алгоритмы вычисления коротких циклических сверток приведены в приложении А. Излагаемый в гл. 7 метод, известный под названием алгоритма Агарвала — Кули, позволяет строить алгоритмы вычисления сверток больших длин, комбинируя известные алгоритмы для коротких циклических сверток.

Рассмотрим задачу вычисления циклической свертки

$$s(x) = g(x)\, d(x) \pmod{x^n - 1},$$

где оба многочлена $g(x)$ и $d(x)$ имеют степень, равную $n-1$. Один из способов решения состоит в том, чтобы найти сначала линейную свертку, а затем выполнить приведение по модулю $x^n - 1$. Чтобы реализовать такой способ, надо модуль $m(x)$ для вычисления линейной свертки выбрать так, чтобы его степень была по меньшей мере равна $2n-1$, так как она должна быть больше суммы степеней $g(x)$ и $d(x)$.

Альтернативный способ состоит в использовании китайской теоремы об остатках, на последнем шаге которой для восстановления многочлена $s(x)$ в качестве модуля $m(x)$ используется непосредственно $x^n - 1$, а в качестве взаимно простых модулей используются делители $m^{(0)}(x)$, $m^{(1)}(x)$, ..., $m^{(K-1)}(x)$ многочлена $m(x)$. Так как сумма степеней этих простых делителей должна быть равна n, что меньше чем $2n-1$, то можно ожидать упрощения вычислений. С другой стороны, делители уже нельзя выбирать так, как нам удобно. Первый метод допускает выбор произвольных делителей, но сумма их степеней должна быть равной $2n-1$, а не n.

В качестве простейшего примера рассмотрим вычисление 4-точечной циклической свертки:

$$s(x) = g(x)\, d(x) \pmod{x^4 - 1}.$$

Делителями многочлена $x^4 - 1$ служат циклотомические многочлены,

$$x^4 - 1 = (x-1)(x+1)(x^2+1) = m^{(0)}(x)\, m^{(1)}(x)\, m^{(2)}(x).$$

В этой задаче у нас нет реального выбора: циклотомические многочлены представляют собой такие модули, которые необходимо

использовать в алгоритме Винограда. Коэффициенты вычетов даются равенствами

$$\begin{bmatrix} d_0^{(0)} \\ d_0^{(1)} \\ d_0^{(2)} \\ d_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix},$$

$$\begin{bmatrix} g_0^{(0)} \\ g_0^{(1)} \\ g_0^{(2)} \\ g_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}.$$

Мы уже построили несколько алгоритмов вычисления умножения по модулю $x^2 + 1$. Одним из них является правило

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_1 - g_0 & \\ & & g_1 + g_0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}.$$

Следовательно, внутренние переменные даются определениями

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix},$$

и

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}.$$

Далее, $S_k = G_k D_k$ для $k = 0, \ldots, 4$. Китайская теорема об остатках дает

$$s(x) = \frac{1}{4}(x^3 + x^2 + x + 1)\, s^{(0)}(x) - \frac{1}{4}(x^3 - x^2 + x - 1)\, s^{(1)}(x) -$$

$$- \frac{1}{2}(x^2 - 1)\, s^{(2)}(x) \pmod{x^4 - 1}.$$

Таким образом,

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{4}s_0^{(0)} \\ \frac{1}{4}s_0^{(1)} \\ \frac{1}{2}s_0^{(2)} \\ \frac{1}{2}s_1^{(2)} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{4}S_0 \\ \frac{1}{4}S_1 \\ \frac{1}{2}S_2 \\ \frac{1}{2}S_3 \\ \frac{1}{2}S_4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{4}S_0 \\ \frac{1}{4}S_1 \\ \frac{1}{2}S_2 \\ \frac{1}{2}S_3 \\ \frac{1}{2}S_4 \end{bmatrix}.$$

Окончательный вид алгоритма показан на рис. 3.14. В упорядочивании элементов матриц, конечно, имеется некоторый произвол. В матрице предсложений допустима произвольная перестановка строк, а в матрице постсложений — перестановка столбцов; это не влияет на результаты вычислений.

На рис. 3 15 приведены характеристики некоторых наилучших известных алгоритмов вычисления коротких сверток. Эти алгоритмы, часть из которых приведена в приложении А, построены по описанному методу. В некоторых случаях для уменьшения числа сложений использовались дополнительные методы, аналогичные тем, которые будут описаны ниже в оставшейся части раздела.

Рис. 3.15 следовало бы дополнить характеристиками коротких сверток для поля комплексных чисел. Каждый алгоритм вычисления свертки в вещественном поле подходит в качестве алгоритма для комплексного поля. Но иногда многочлен $x^n - 1$ над полем комплексных чисел разлагается на большее число простых делителей, чем над полем вещественных чисел. Это в комплексном случае дает конструктору алгоритма дополнительные возможности.

Алгоритм Винограда можно рассматривать как метод разложения некоторых матриц. Пусть $\mathbf{s}$, $\mathbf{g}$ и $\mathbf{d}$ — векторы длины n, компонентами которых являются соответственно коэффициенты многочленов $s(x)$, $g(x)$ и $d(x)$. Циклическая свертка $s(x) =$

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} G_0 & & & & \\ & G_1 & & & \\ & & G_2 & & \\ & & & G_3 & \\ & & & & G_4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix},$$

где

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 2 & 0 & -2 & 0 \\ -2 & 2 & 2 & -2 \\ 2 & 2 & -2 & -2 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}$$

Рис. 3.14. Алгоритм 4-точечной циклической свертки.

$$s(x) = g(x)\, d(x) \pmod{x^n - 1}$$
$$\deg g(x) = n - 1$$
$$\deg d(x) = n - 1$$

Длина n	Вещественные свертки	
	Число вещественных умножений	Число вещественных сложений
2	2	4
3	4	11
4	5	15
5	8	62
5	10	31
7	16	70
8	12	72
8	14	46
9	19	74
16	33	181
16	35	155

Рис. 3.15. Характеристики некоторых алгоритмов вычисления циклической свертки.

$= g(x)\, d(x) \pmod{x^n - 1}$ может быть записана в виде матричного произведения

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \cdot \\ \cdot \\ \cdot \\ s_{n-1} \end{bmatrix} = \begin{bmatrix} g_0 & g_{n-1} & \cdots & g_1 \\ g_1 & g_0 & & g_2 \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ g_{n-2} & & & g_{n-1} \\ g_{n-1} & g_{n-2} & \cdots & g_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \cdot \\ \cdot \\ \cdot \\ d_{n-1} \end{bmatrix},$$

или

$$\mathbf{s} = \mathbf{T}\mathbf{d}.$$

Алгоритм Винограда, записываемый равенством

$$\mathbf{s} = \mathbf{C}\,[(\mathbf{B}\mathbf{g})\cdot(\mathbf{A}\mathbf{d})],$$

может быть записан также в виде

$$\mathbf{s} = \mathbf{C}\mathbf{G}\mathbf{A}\mathbf{d},$$

где $\mathbf{A}$ представляет собой матрицу размера $M(n) \times n$, $\mathbf{G}$ является диагональной матрицей размера $M(n) \times M(n)$, а $\mathbf{C}$ является матрицей размера $n \times M(n)$. Таким образом, алгоритм Винограда можно рассматривать как алгоритм разложения матрицы

$$\mathbf{T} = \mathbf{C}\mathbf{G}\mathbf{A},$$

где $\mathbf{G}$ — диагональная матрица, а $\mathbf{C}$ и $\mathbf{A}$ — матрицы, элементами которых являются малые целые числа.

Так как в большинстве приложений многочлену $g(x)$ соответствует фиксированный КИО-фильтр, или по крайней мере КИО-фильтр, коэффициенты которого меняются не слишком часто, то вычисление $\mathbf{G} = \mathbf{B}\mathbf{g}$ приходится выполнять не слишком часто и при подсчете общей вычислительной нагрузки им можно пренебречь. Следовательно, сложность матрицы $\mathbf{B}$ особой роли не играет.

Роли матриц $\mathbf{A}$ и $\mathbf{B}$ симметричны, так как их можно поменять местами путем простого переименования векторов $\mathbf{d}$ и $\mathbf{g}$. Алгоритм Винограда вычисления свертки допускает построение матриц $\mathbf{A}$ и $\mathbf{B}$, отличающихся только перестановкой строк, и, следовательно, имеющих одну и ту же сложность; но в общем случае более целесообразно одну из матриц строить более сложной и именно ей приписывать роль матрицы $\mathbf{A}$. Удивительно то, что в алгоритме циклической свертки можно менять ролями даже матрицы $\mathbf{C}$ и $\mathbf{A}$. Способ сделать это описывается следующей теоремой.

Теорема 3.6.1 (теорема об обмене матриц). *Если теплицева матрица* $\mathbf{S}$ *допускает факторизацию вида*

$$\mathbf{S} = \mathbf{C}\mathbf{D}\mathbf{E},$$

то она может быть также записана в виде

$$\mathbf{S} = (\tilde{\mathbf{E}})^T\,\mathbf{D}^T\,(\underset{\sim}{\mathbf{C}})^T,$$

где матрица $\tilde{\mathbf{E}}$ *отличается от матрицы* $\mathbf{E}$ *обращенным порядком столбцов, а матрица* $\underset{\sim}{\mathbf{C}}$ *отличается от матрицы* $\mathbf{C}$ *обращенным порядком строк.*

Доказательство. Пусть $\mathbf{J}$ представляет собой обменную матрицу того же размера, что и матрица $\mathbf{S}$. Тогда

$$\mathbf{S}^T = \mathbf{J}\mathbf{S}\mathbf{J} = (\mathbf{J}\mathbf{C})\,\mathbf{D}\,(\mathbf{E}\mathbf{J}) = \underset{\sim}{\mathbf{C}}\mathbf{D}\tilde{\mathbf{E}}.$$

Операция транспонирования завершает доказательство. □

Применим эту теорему к умножению многочленов по модулю многочлена $x^n - \xi$, где ξ — некоторая константа. Пусть надо вычислить

$$s(x) = g(x)\,d(x) \pmod{x^n - \xi},$$

где $g(x)$ и $d(x)$ многочлены степени не более $n-1$. Если $\xi = 1$, то мы имеем задачу о циклической свертке. Случай $\xi = -1$ (а в поле комплексных чисел и случаи $\xi = \pm j$) также очень важен. Запишем это произведение многочленов в матрично-векторном виде

$$\begin{bmatrix} s_0 \\ s_1 \\ \cdot \\ \cdot \\ \cdot \\ s_{n-1} \end{bmatrix} = \begin{bmatrix} d_0 & \xi d_{n-1} & \xi d_{n-2} & \cdots & \xi d_1 \\ d_1 & d_0 & \xi d_{n-1} & & \\ d_2 & d_1 & d_0 & & \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ d_{n-1} & & & \cdots & d_0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \cdot \\ \cdot \\ \cdot \\ g_{n-1} \end{bmatrix},$$

или

$$\mathbf{s} = \mathbf{T}\mathbf{g},$$

где $\mathbf{T}$ — теплицева матрица. Алгоритм

$$\mathbf{s} = \mathbf{C}\,[(\mathbf{B}\mathbf{d})\cdot(\mathbf{A}\mathbf{g})]$$

можно перезаписать в виде

$$\mathbf{s} = \mathbf{C}\mathbf{D}\mathbf{A}\mathbf{g},$$

где $\mathbf{D}$ — диагональная матрица, диагональные элементы которой равны компонентам вектора $\mathbf{B}\mathbf{d}$. Согласно теореме 3.6.1,

$$\mathbf{s} = (\tilde{\mathbf{A}})^T\,\mathbf{D}\,(\underset{\sim}{\mathbf{C}})^T\,\mathbf{g}.$$

Применительно к задаче вычисления циклической свертки теорема 3.6.1 означает, что наиболее сложную из трех матриц $\mathbf{A}$, $\mathbf{B}$ и $\mathbf{C}$ можно выбрать множителем при векторе $\mathbf{g}$ и «включить» ее в матрицу $\mathbf{G}$.

3.7. Свертки в общих полях и кольцах

Любой алгоритм свертки, такой как алгоритм Кука — Тоома или алгоритм Винограда, представляет собой некоторое тождество, опирающееся на свойства операций в данном поле, в частности на свойства ассоциативности, дистрибутивности и комму-

тативности. Алгоритм, построенный в одном поле, может быть использован в любом расширении этого поля. Он, однако, может оказаться не столь хорошим, как алгоритм, разработанный специально для данного поля.

В частности, алгоритм свертки двух вещественных последовательностей можно без изменений использовать для свертки последовательностей комплексных чисел. Если обозначить через M и A соответственно числа вещественных умножений и сложений, необходимых для вычисления вещественной свертки, и если для вычисления комплексной свертки использовать тот же алгоритм с обычными комплексными умножениями и обычными комплексными сложениями, то в общей сложности получится $4M$ вещественных умножений и $2A + 2M$ вещественных сложений. Если вместо этого воспользоваться построенным в разд. 1.1 комплексным умножением и рассматривать одну часть свертки как отводы фиксированного фильтра, то потребуется только $3M$ вещественных умножений и $2A + 3M$ вещественных сложений.

Если длина комплексной циклической свертки равна некоторой степени двойки, то можно построить даже еще лучший алгоритм. Для пояснения того, как это делается, рассмотрим свертку по модулю $x^{2^t} + 1$.

В кольце многочленов по модулю многочлена $x^{2^t} + 1$, $t \geqslant 1$, имеется элемент, квадратный корень из которого равен минус единице. Действительно,

$$-1 = x^{2^t} \pmod{x^{2^t} + 1}$$

так что в рассматриваемом кольце $\sqrt{-1} = x^{2^{t-1}}$. Воспользуемся этим элементом, чтобы выписать комплексную свертку через две вещественные свертки.

Для вычисления комплексной свертки

$$s(x) = g(x)\, d(x) \pmod{x^{2^t} + 1},$$

где

$$g(x) = g_R(x) + j g_I(x) \quad \text{и} \quad d(x) = d_R(x) + j d_I(x),$$

можно вычислить по модулю $x^{2^t} + 1$ четыре вещественных свертки $g_R(x)\, d_R(x)$, $g_I(x)\, d_R(x)$, $g_R(x)\, d_I(x)$ и $g_I(x)\, d_I(x)$ и записать

$$s_R(x) = g_R(x)\, d_R(x) - g_I(x)\, d_I(x),$$
$$s_I(x) = g_R(x)\, d_I(x) + g_I(x)\, d_R(x).$$

Лучшая процедура, содержащая вдвое меньше умножений, сводится к введению многочленов

$$a(x) = \frac{1}{2}\left(g_R(x) - x^{2^{t-1}} g_I(x)\right)\left(d_R(x) - x^{2^{t-1}} d_I(x)\right) \pmod{x^{2^t} + 1}$$

$$b(x) = \frac{1}{2}\left(g_R(x) + x^{2^{t-1}} g_I(x)\right)\left(d_R(x) + x^{2^{t-1}} d_I(x)\right) \pmod{x^{2^t} + 1},$$

вычисление которых сводится к вычислению двух вещественных сверток. Выходной многочлен $s(x) = g(x)\, d(x) \pmod{x^{2^t} + 1}$ дается при этом равенствами

$$s_R(x) = a(x) + b(x),$$
$$s_I(x) = x^{2^{t-1}}(a(x) - b(x)) \pmod{x^{2^t} + 1}.$$

Мы хотим воспользоваться этой процедурой для вычисления комплексной циклической свертки

$$s(x) = g(x)\, d(x) \pmod{x^n - 1},$$

где n равно степени двойки. Запишем

$$x^n - 1 = (x-1)(x+1)(x^2+1)(x^4+1) \ldots (x^{n/2}+1).$$

Тогда задача сводится к коротким циклическим сверткам вид

$$s^{(i)}(x) = g^{(i)}(x)\, d^{(i)}(x) \pmod{x^{2^i} + 1}.$$

Комплексные свертки по модулям $x - 1$ и $x + 1$ являются скалярами, и их произведение вычисляется как произведение комплексных чисел. Их вклад в общую сложность очень мал. Если остальные комплексные свертки вычислять через две вещественные свертки, то рассматриваемый способ вычисления комплексной свертки потребует примерно вдвое больше вычислений, чем вещественная свертка той же длины.

Этот метод конкурирует с методом разложения многочлена $x^n - 1$ в поле комплексных чисел:

$$x^n - 1 = (x-1)(x+1)(x-j)(x+j) \ldots (x^{n/4} - j)(x^{n/4} + j).$$

Каждая из индивидуальных подзадач теперь меньше, но арифметика является комплексной.

3.8. Сложность алгоритмов свертки

Пусть задан алгоритм решения некоторой задачи; следует ли этим удовлетвориться или следует попытаться найти лучший алгоритм? На этот вопрос трудно ответить по нескольким причинам. Во-первых, трудно выбрать критерий, согласно которому можно утверждать, что алгоритм является наилучшим. Далее, если даже такой критерий выбран, то трудно утверждать, что характеристики оптимального алгоритма соответствуют выбранному критерию.

Во многих задачах критерием оптимальности служит минимизация числа умножений. Этот критерий достаточно прост, так что удается доказать некоторые теоремы, описывающие характеристики оптимального алгоритма. Конечно, после отыскания оптимального алгоритма может оказаться, что он нам не подойдет,

скажем, по числу сложений, но все-таки хочется знать, что представляет собой такой алгоритм.

В данном разделе мы будем исследовать характеристики оптимального (в смысле минимального числа умножений) алгоритма свертки. Для этого надо уточнить понятие умножения, и мы сделаем это так, чтобы можно было ответить на интересующие нас вопросы. А именно, мы хотим определить умножение таким специальным образом, чтобы под произведением $d \cdot g$ понималось произведение произвольных вещественных чисел, но туда бы не входило произведение $2g$, так как его можно интерпретировать как $g + g$. Такое различие легко принять, но как быть с произведением $3g$ или $(5/7)\,g$? Мы выберем определение, которое приводит к полезным результатам. Вычисление $d \cdot g$ будем считать умножением только тогда, когда оба сомножителя принимают произвольные вещественные значения, и не будем считать умножением, если произвольные вещественные значения допустимы только для одного из них, а другой должен быть рациональным. Мы увидим, что это определение, хотя интуитивно и кажется несколько ущербным, приводит к осмысленным результатам. Выбранный нами критерий может также вызвать подозрение потому, что возникающие в приложениях числа всегда записываются словом конечной длины, так что все возникающие в приложениях числа являются рациональными. Тем не менее если в принципе переменные принимают вещественные значения, то указанное вычисление будет называться произведением.

В указанном смысле алгоритмы Винограда вычисления коротких циклических сверток являются оптимальными. Ни один алгоритм вычисления n-точечной циклической свертки не содержит умножений меньше, чем алгоритм Винограда. Доказательство этого факта составляет одну из сложных задач данного раздела. Мы докажем следующие результаты:

1. Никакой алгоритм вычисления линейной свертки двух последовательностей длин L и N не может содержать меньшее число умножений, чем $L + N - 1$.

2. Если число простых делителей многочлена $x^n - 1$ равно t, то никакой алгоритм вычисления n-точечной циклической свертки не может содержать меньшее число умножений, чем $2n - t$.

3. Если число простых делителей многочлена $p(x)$ равно t, то никакой алгоритм вычисления произведения многочленов $g(x)\,d(x)$ по модулю $p(x)$ не может содержать меньшее число умножений, чем $2n - t$.

Вторая задача, конечно, является частным случаем третьей, но она столь важна, что ее следует выделить отдельно.

Рассматриваемые идеи не зависят от поля. Пусть F — некоторое поле, которое мы назовем *полем вычислений*, а E — его подполе, называемое *полем констант* или *основным полем*. Элементы

из E называются *скалярами*. Пусть $\mathbf{d} = (d_0, \ldots, d_{n-1})$ и $\mathbf{g} = (g_0, \ldots, g_{r-1})$ — произвольные векторы фиксированных длин n и r с компонентами из поля F. Компоненты векторов $\mathbf{d}$ и $\mathbf{g}$ будем называть *неопределенными переменными* или просто *переменными*. Эти $n + r$ переменных величин независимы: их не связывают никакие соотношения. Алгоритмом называется правило вычисления последовательности элементов $f_1, \ldots, f_t$ из F, таких что каждый элемент f_t последовательности равен одной из следующих величин: (1) или компоненте вектора $\mathbf{d}$, или компоненте вектора $\mathbf{g}$, или сумме, разности или произведению двух таких компонент; (2) сумме, разности или произведению компонент вектора $\mathbf{d}$ или вектора $\mathbf{g}$ на элемент f_j последовательности с номером j, меньшим номера i; (3) сумме, разности или произведению двух элементов f_j и f_k последовательности, номера j и k которых меньше номера i; (4) элементу поля E.

Скажем, что алгоритм вычисляет выходной вектор $\mathbf{s}$, если компоненты вектора $\mathbf{s}$ содержатся в последовательности $f_1, \ldots, f_t$. Необходимо подчеркнуть, что являющиеся компонентами вектора $\mathbf{s}$ переменные величины связаны некоторыми функциональными соотношениями с переменными, являющимися компонентами векторов $\mathbf{d}$ и $\mathbf{g}$. Алгоритм, вычисляющий вектор $\mathbf{s}$, представляет собой фиксированную процедуру правильного вычисления $\mathbf{s}$ для любых возможных значений входящих в $\mathbf{d}$ и $\mathbf{g}$ переменных.

Заметим, что данное определение алгоритма не включает в себя ни ветвления, ни деления. В алгоритмах, которыми мы занимаемся, эти операции не являются необходимыми, к тому же ни деление, ни ветвление не могут уменьшить числа умножений.

Данное определение алгоритма можно проиллюстрировать примером комплексного умножения. Сначала запишем его в виде

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}.$$

Тогда равенства

$$f_1 = ca, \quad f_2 = db, \quad f_3 = f_1 - f_2, \quad f_4 = da, \quad f_5 = cb, \quad f_6 = f_4 + f_5$$

дают описание алгоритма в виде последовательности правил вычисления.

Вычисление произведения комплексных чисел можно также записать в виде

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} (c-d) & 0 & 0 \\ 0 & (c+d) & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}.$$

Тогда равенства

$$f_1 = a - b,\ f_2 = c - d,\ f_3 = c + d,\ f_4 = f_2 a,$$
$$f_5 = f_3 b,\ f_6 = df_1,\ f_7 = f_4 + f_6,\ f_8 = f_5 + f_6$$

дают описание последнего алгоритма в виде последовательности правил вычисления.

Рассмотрим набор всех таких множеств правил вычисления комплексного умножения. Оптимальным из этого набора алгоритмов является тот из них, который содержит наименьшее число умножений.

Теперь можно формализовать определение задачи вычислений. Будем рассматривать только задачи вида

$$\mathbf{s} = \mathbf{H}\mathbf{d},$$

где $\mathbf{d}$ — входной вектор данных длины k, а $\mathbf{s}$ — выходной вектор длины n. Элементы матрицы $\mathbf{H}$ представляют собой линейные комбинации r неопределенных переменных $g_0, \ldots, g_{r-1}$. В типичных случаях r меньше числа элементов матрицы $\mathbf{H}$ и каждая переменная может присутствовать в матрице $\mathbf{H}$ более одного раза. Такая структура включает в себя все рассмотренные задачи вычисления свертки.

Под элементами матрицы $\mathbf{H}$ будем понимать не элементы поля, а линейные комбинации неопределенных переменных вида

$$H_{ij} = \sum_{k=0}^{r-1} \alpha_{ijk} g_k,$$

где α_{ijk} являются скалярами. Две такие линейные формы можно складывать, любую линейную форму можно умножать на скаляр. Множество таких линейных форм над полем E образует векторное пространство, которое мы обозначим $E\ [g_0, \ldots, g_{r-1}]$. Таким образом, $\mathbf{H}$ является матрицей над множеством линейных форм. Для матрицы $\mathbf{H}$ не выполняются многие известные свойства матриц, поскольку она является матрицей не над полем (и даже не над кольцом), а над множеством $E\ [g_0, \ldots, g_{r-1}]$. В частности, ранг по столбцам не обязательно равен рангу по строкам. Как мы увидим, ранг по строкам и ранг по столбцам каждый дают нижнюю границу для числа умножений в задаче $\mathbf{s} = \mathbf{H}\mathbf{d}$. Возможность менять ролями векторы $\mathbf{g}$ и $\mathbf{d}$ дает два пути применения этих двух границ.

Ранг по строкам определяется следующим образом (аналогично определяется ранг по столбцам). Каждая строка матрицы $\mathbf{H}$ представляет собой вектор длины k с компонентами из множества [1])

[1]) Это множество образует векторное пространство, но мы предпочитаем называть его множеством, так как не рискуем говорить «вектор векторов».

$E\ [g_0, \ldots, g_{r-1}]$. Линейная комбинация строк матрицы $\mathbf{H}$ также является вектором длины k с компонентами из того же множества вида $\sum_{i=1}^{n} \beta_i \mathbf{h}_i$, где элементы β_i принадлежат полю E для всех $i = 1, \ldots, n$. Рангом по строкам матрицы $\mathbf{H}$ называется мощность наибольшего множества линейно независимых строк, т. е. такого наибольшего множества строк, что никакая ненулевая их линейная комбинация не равна нулю.

Например, над полем рациональных (вещественных) чисел ранг по столбцам матрицы

$$\mathbf{H} = \begin{bmatrix} 4g_0 & -g_0 & g_0 \\ 2g_1 & -g_1 & 0 \\ 2g_1 - 2g_0 & g_0 & g_1 \end{bmatrix}$$

равен двум, так как

$$\frac{1}{2}\begin{bmatrix} 4g_0 \\ 2g_1 \\ 2g_1 - 2g_0 \end{bmatrix} + \begin{bmatrix} -g_0 \\ -g_1 \\ g_0 \end{bmatrix} - \begin{bmatrix} g_0 \\ 0 \\ g_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

но никакая линейная комбинация двух столбцов не равна тождественно нулю.

Теорема 3.8.1 (теорема о ранге по строкам). *Для произвольного алгоритма вычисления* $\mathbf{s} = \mathbf{H}\mathbf{d}$ *число умножений по меньшей мере равно рангу по строкам матрицы* $\mathbf{H}$.

Доказательство. Без потери общности можно предположить, что линейно независимыми являются первые ρ строк матрицы $\mathbf{H}$, и в дальнейшем рассматривать только эти строки. Обозначим через $\mathbf{M}$ матрицу, образованную этими строками, и рассмотрим только частичное вычисление

$$\begin{bmatrix} s_0 \\ \vdots \\ s_{\rho-1} \end{bmatrix} = \mathbf{M}\mathbf{d}.$$

Идея доказательства сводится к построению подходящей матрицы $\mathbf{A}$ над полем E, для которой выполняются известные свойства матриц.

Предположим, что в алгоритме, задаваемом последовательностью $f_1, \ldots, f_N$, имеется l шагов умножения, задаваемых соответственно l членами последовательности $e_1, \ldots, e_l$. Тогда первые ρ компонент вектора $\mathbf{s}$ должны быть линейными комбинациями

этих членов-произведений, линейных членов и элементов основного поля. Таким образом,

$$\begin{bmatrix} s_0 \\ \vdots \\ s_{\rho-1} \end{bmatrix} = \mathbf{A} \begin{bmatrix} e_1 \\ \vdots \\ e_l \end{bmatrix} + \begin{bmatrix} b_0 \\ \vdots \\ b_{\rho-1} \end{bmatrix}$$

где элементы матрицы $\mathbf{A}$ принадлежат основному полю E, а компоненты вектора $\mathbf{b}$ являются линейными комбинациями неопределенных переменных и элементов основного поля.

Предположим теперь, что ρ больше, чем l. Тогда матрица $\mathbf{A}$ содержит строк больше, чем столбцов, и, следовательно, строки матрицы $\mathbf{A}$ линейно зависимы, т. е. существует такой вектор $\mathbf{c}$ над полем E, что $\mathbf{c}^T\mathbf{A} = \mathbf{0}^T$. Следовательно,

$$\mathbf{c}^T(\mathbf{Md}) = \mathbf{c}^T(\mathbf{Ae} + \mathbf{b}),$$

и так как $\mathbf{c}^T\mathbf{A} = \mathbf{0}^T$, то это приводится к виду

$$(\mathbf{c}^T\mathbf{M})\,\mathbf{d} = \mathbf{c}^T\mathbf{b}.$$

В этом равенстве правая часть не содержит произведений неопределенных переменных, так как $\mathbf{c}$ содержит только элементы поля E, а в $\mathbf{b}$ не входят произведения неопределенных переменных; следовательно, и левая часть не содержит произведений неопределенных переменных. Так как компонентами вектора $\mathbf{d}$ являются неопределенные переменные, то отсюда следует, что $\mathbf{c}^T\mathbf{M}$ не содержит неопределенных переменных. Но $\mathbf{c}^T\mathbf{M}$ является вектором, компоненты которого равны линейным комбинациям неопределенных переменных. Следовательно, он равен нулю и строки матрицы $\mathbf{M}$ линейно зависимы. Полученное противоречие означает, что l больше, чем ρ, и число умножений по меньшей мере равно рангу по строкам матрицы $\mathbf{H}$. □

Теорема 3.8.2 (теорема о ранге по столбцам). *Для произвольного алгоритма вычисления* $\mathbf{s} = \mathbf{Hd}$ *число умножений по меньшей мере равно рангу по столбцам матрицы* $\mathbf{H}$.

Доказательство. Доказательство проведем индукцией. Если ранг по столбцам равен единице, то любой алгоритм должен содержать по меньшей мере одно умножение. Пусть при ранге по столбцам, равном $l - 1$, теорема верна. То есть если ранг по столбцам матрицы $\mathbf{H}$ равен $l - 1$, то любой алгоритм вычисления $\mathbf{s} = \mathbf{Hd}$ содержит по меньшей мере $l - 1$ умножение. Шаг индукции состоит в доказательстве соответствующего утверждения для случая, когда ранг по столбцам равен l.

Предположим, что задан алгоритм вычисления

$$\mathbf{s} = \mathbf{Hd},$$

причем ранг по столбцам матрицы $\mathbf{H}$ равен l. Без потери общности можно предположить, что последний столбец матрицы $\mathbf{H}$ содержит по крайней мере один ненулевой элемент (в противном случае его можно удалить из $\mathbf{H}$). Следовательно, переменная d_l содержится в некотором члене-произведении, скажем, последнем таком члене.

Это означает, что для некоторого множества скаляров α_i сумма $\sum_i \alpha_i d_i$ является множителем в некотором умножении, причем $\alpha_l \neq 0$. Так как на скаляры не наложено никаких ограничений, то можно полагать α_l равным 1, так что член $d_l + \sum_{i=0}^{l-1} \alpha_i d_i$ является множителем в некотором члене-произведении.

Чтобы завершить доказательство, нам надо при любом данном алгоритме решения исходной задачи построить, хотя и искусственную, новую задачу

$$\mathbf{s}' = \mathbf{H}'\mathbf{d}'$$

ранга $l - 1$, которую можно решить данным алгоритмом, удаляя из него одно умножение. Тогда, согласно предположению индукции, любой алгоритм решения новой задачи содержит по меньшей мере $l - 1$ умножение, и, следовательно, исходная задача данным алгоритмом решается по меньшей мере с l умножениями.

Чтобы это сделать, заменим в исходной задаче переменную d_l на $-\sum_{i=0}^{l-1} \alpha_i d_i$. Тогда последний член-произведение, содержащий множителем сумму $d_l + \sum_{i=0}^{l-1} \alpha_i d_i$, будет представлять собой умножение на нуль и, следовательно, может быть удален из алгоритма. Алгоритм теперь решает некоторую новую задачу, а именно,

$$\mathbf{s}' = \mathbf{H}'\mathbf{d}',$$

где $\mathbf{d}'$ представляет собой вектор длины $l - 1$, формируемый из вектора $\mathbf{d}$ удалением последней компоненты, а матрица $\mathbf{H}'$ получается из матрицы $\mathbf{H}$ заменой j-го столбца h_j на столбец $h_j - \alpha_j h_l$. Таким образом,

$$\mathbf{H}' \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{l-1} \end{bmatrix} = \mathbf{H} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{l-1} \\ -\sum_{i=0}^{l-1} \alpha_i d_i \end{bmatrix}$$

и данный алгоритм вычисляет $\mathbf{H}'\mathbf{d}'$, что и завершает доказательство. □

Теорема 3.8.3. *Каждый алгоритм вычисления линейной свертки*

$$s(x) = g(x)\,d(x),$$

где $\deg g(x) = L - 1$ *и* $\deg d(x) = N - 1$, *содержит по меньшей мере* $L + N - 1$ *умножений.*

Доказательство. Рассматриваемое вычисление можно в матричном виде записать равенством

$$\mathbf{s} = \mathbf{H}\mathbf{d},$$

где

$$\mathbf{H} = \begin{bmatrix} g_0 & 0 & \cdots & 0 & 0 \\ g_1 & g_0 & \cdots & 0 & 0 \\ \vdots & & & & \\ 0 & & \cdots & g_{L-1} & g_{L-2} \\ 0 & & \cdots & 0 & g_{L-1} \end{bmatrix}$$

является $((L + N - 1) \times N)$-матрицей. Как элементы множества $E\,[g_0, g_1, \ldots, g_{L-1}]$, строки матрицы $\mathbf{H}$, очевидно, линейно независимы, следовательно, применение теоремы 3.8.1 показывает, что число умножений равно по меньшей мере $L + N - 1$. □

Теорема 3.8.4. *Пусть* $p(x)$ *является простым многочленом степени* n. *Каждый алгоритм вычисления произведения многочленов*

$$s(x) = g(x)\,d(x) \pmod{p(x)}$$

содержит по меньшей мере $2n - 1$ *умножений.*

Доказательство. Предположим, что алгоритм содержит t умножений. Тогда на выходе алгоритма получим линейную комбинацию этих t членов-произведений

$$\mathbf{s} = \mathbf{A}\mathbf{S},$$

где $\mathbf{A}$ является $(n \times t)$-матрицей над полем E, а $\mathbf{S}$ — вектор длины t, компонентами которого являются эти члены-произведения.

Ясно, что многочлены $g(x)$ и $d(x)$ всегда можно выбрать так, чтобы сделать некоторую одну компоненту многочлена $s(x)$ ненулевой, а остальные нулевыми. Следовательно, n строк матрицы $\mathbf{A}$ должны быть линейно независимыми, так что $\mathbf{A}$ должна содержать и n линейно независимых столбцов. Без потери общности можно полагать, что первые n столбцов матрицы $\mathbf{A}$ являются линейно независимыми, так что матрица $\mathbf{A}$ может быть разбита на блоки вида

$$\mathbf{s} = [\mathbf{A}' \mid \mathbf{A}'']\mathbf{S},$$

где $\mathbf{A}'$ — обратимая $(n \times n)$-матрица. Умножим на обратную к $\mathbf{A}'$ матрицу $\mathbf{C}$:

$$\mathbf{C}\mathbf{s} = [\mathbf{I} \mid \mathbf{P}]\mathbf{S}$$
$$= \mathbf{C}\mathbf{H}\mathbf{d}.$$

Так как многочлен $p(x)$ неприводим, то можно показать, что все элементы любой строки матрицы $\mathbf{H}$ линейно независимы, равно как и все элементы любой линейной комбинации строк матрицы $\mathbf{H}$. Это утверждение является стандартным результатом теории матриц и будет доказано отдельно в виде теоремы 3.8.6 в конце настоящего раздела. Следовательно, все элементы первой строки матрицы $\mathbf{C}\mathbf{H}$ линейно независимы, так что согласно теореме 3.8.2 для вычисления первой компоненты вектора $\mathbf{C}\mathbf{H}\mathbf{d}$ требуется по меньшей мере n умножений.

С другой стороны, первая строка вычисления

$$\mathbf{C}\mathbf{s} = [\mathbf{I} \mid \mathbf{P}]\mathbf{S},$$

содержит самое большое $1 + (t - n)$ умножений, так как матрица $\mathbf{P}$ содержит $t - n$ столбцов, а вектор $\mathbf{S}$ длины t содержит все члены-произведения. Следовательно, $1 + (t - n) \geqslant n$, так что $t \geqslant 2n - 1$, что и доказывает теорему. □

В качестве приложения этой теоремы рассмотрим комплексное умножение как задачу умножения по модулю многочлена $x^2 + 1$. Согласно теореме 3.8.4, для выполнения одного комплексного умножения необходимо по меньшей мере три вещественных умножения.

Хотя мы и не будем этого делать, теорему 3.8.4 можно доказать и для умножения многочленов по модулю многочлена $p^l(x)$, где $p(x)$ — простой многочлен. Вместо этого мы рассмотрим случай, когда $p(x)$ распадается в произведение k простых многочленов. В этом случае китайская теорема об остатках позволяет разбить задачу на k подзадач, каждая длины n_i, где $\sum_i n_i = n$. Дополнительных умножений при этом не возникает. Следовательно, используя для каждой из подзадач теорему 3.8.4, с помощью китайской теоремы об остатках для полной задачи вычисления произведения многочленов по составному модулю получаем по меньшей мере

$$\sum_{i=1}^{k} (2n_i - 1) = 2n - k$$

умножений. Следующая теорема утверждает, что ни один алгоритм решения этой задачи не может быть лучше такого использования китайской теоремы об остатках.

Теорема 3.8.5. *Пусть многочлен $p(x)$ степени n распадается в произведение k различных простых множителей. Каждый алгоритм вычисления произведения многочленов*

$$s(x) = g(x)\, d(x) \pmod{p(x)}$$

содержит по меньшей мере $2n - k$ умножений.

Доказательство. Так как китайская теорема об остатках задает не требующее умножений обратимое преобразование, то достаточно рассмотреть вычисление $\mathbf{s} = \mathbf{H}\mathbf{d}$, в котором матрица $\mathbf{H}$ блочно-диагональная, т. е.

$$\begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \vdots \\ \mathbf{s}_k \end{bmatrix} = \left[\begin{array}{c|c|c|c} \mathbf{H}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{H}_2 & & \\ \hline \vdots & & & \\ \hline \mathbf{0} & & & \mathbf{H}_k \end{array}\right] \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \vdots \\ \mathbf{d}_k \end{bmatrix},$$

и соответствующая китайской теореме об остатках i-я подзадача задается вычислением $\mathbf{s}_i = \mathbf{H}_i\mathbf{d}_i$.

Теперь повторим доказательство теоремы 3.8.4. Предположим, что алгоритм содержит t умножений. Тогда

$$\mathbf{s} = \mathbf{A}\mathbf{S},$$

где $\mathbf{S}$ — вектор длины t, содержащий все члены-произведения, и $\mathbf{A}$ представляет собой $(n \times t)$-матрицу над полем E. Так как $\mathbf{A}$ содержит n линейно независимых строк, то она содержит и n линейно независимых столбцов, в качестве которых можно выбрать первые n столбцов. Тогда

$$\mathbf{s} = [\mathbf{A}' \mid \mathbf{A}'']\mathbf{S} \qquad \text{и} \qquad \mathbf{C}\mathbf{s} = [\mathbf{I} \mid \mathbf{P}]\mathbf{S}$$

где $\mathbf{C}$ обозначает матрицу, обратную к матрице $\mathbf{A}'$. Следовательно, для вычисления первой компоненты вектора $\mathbf{C}\mathbf{s}$ требуется самое большое $1 + (t - n)$ умножений. Но, кроме того, выполняется равенство

$$\mathbf{C}\mathbf{s} = \mathbf{C}\begin{bmatrix} \mathbf{H}_1 & & & \\ & \mathbf{H}_2 & & \\ & & \ddots & \\ & & & \mathbf{H}_k \end{bmatrix}\mathbf{d}.$$

Рассмотрим теперь некоторую линейную комбинацию строк матрицы $\mathbf{H}$. Любая линейная комбинация строк каждой из матриц $\mathbf{H}_i$ содержит лишь линейно независимые столбцы. Следовательно, любая линейно независимая комбинация строк матрицы $\mathbf{H}$ содержит по меньшей мере $n - (k - 1)$ линейно независимых столбцов. Следовательно, по теореме 3.8.2 вычисление первой компоненты вектора $\mathbf{C}\mathbf{H}\mathbf{d}$ требует по меньшей мере $n - (k - 1)$ умножений. Эти верхняя и нижняя границы числа умножений, необходимых для вычисления первой компоненты вектора $\mathbf{C}\mathbf{s}$, приводят к неравенству

$$1 + (t - n) \geqslant n - (k - 1),$$

так что $t \geqslant 2n - k$, что доказывает теорему. □

Теперь мы должны завершить незаконченное доказательство теоремы 3.8.4.

Теорема 3.8.6. *Пусть $\mathbf{s} = \mathbf{H}\mathbf{d}$ представляет собой матричную запись произведения многочленов*

$$s(x) = g(x)\, d(x) \pmod{p(x)},$$

где $p(x)$ — неприводимый многочлен, и матрица $\mathbf{H}$ составлена из коэффициентов многочлена $g(x)$. Тогда все элементы любой строки матрицы $\mathbf{H}$, так же как и все элементы любой линейной комбинации строк матрицы $\mathbf{H}$, линейно независимы.

Доказательство.

Шаг 1. Обозначим через $\mathbf{C}_p$ *сопровождающую матрицу многочлена* $p(x)$, определяемую следующей $(n \times n)$-матрицей:

$$\mathbf{C}_p = \begin{bmatrix} 0 & 0 & \cdots & 0 & -p_0 \\ 1 & 0 & \cdots & 0 & -p_1 \\ 0 & 1 & \cdots & 0 & -p_2 \\ \vdots & \vdots & & & \\ 0 & 0 & \cdots & 1 & -p_{n-1} \end{bmatrix}.$$

Тогда i-й столбец матрицы $\mathbf{H}$ равен $\mathbf{C}_p^i\mathbf{g}$, и через свои вектор-столбцы матрица $\mathbf{H}$ записывается в виде

$$\mathbf{H} = [\mathbf{g}\ \mathbf{C}_p\mathbf{g}\ \mathbf{C}_p^2\mathbf{g}\ \ldots\ \mathbf{C}_p^{n-1}\mathbf{g}].$$

Пусть $\mathbf{w}$ — произвольная ненулевая вектор-строка, а $\mathbf{w}\mathbf{H}$ — соответствующая линейная комбинация строк матрицы $\mathbf{H}$. Надо показать, что никакая линейная комбинация столбцов $\mathbf{w}\mathbf{H}$ не равна нулю. Предположим, что

$$0 = \sum_{i=0}^{n-1} a_i(\mathbf{w}\mathbf{C}_p^i\mathbf{g}) = \left[\mathbf{w}\sum_{i=0}^{n-1} a_i\mathbf{C}_p^i\right]\mathbf{g}.$$

Так как это равенство должно выполняться для любого $\mathbf{g}$, то

$$\mathbf{w} \cdot \mathbf{a}(\mathbf{C}_p) = \mathbf{0},$$

где

$$\mathbf{a}(\mathbf{C}_p) = \sum_{i=0}^{n-1} a_i \mathbf{C}_p^i$$

представляет собой матрицу размера $n \times n$, вычисленную по матрице $\mathbf{C}_p$. Поскольку строка $\mathbf{w}$ отлична от нулевой, то матрица $\mathbf{a}(\mathbf{C}_p)$ должна быть вырожденной, так как в противном случае $\mathbf{w} \cdot \mathbf{a}(\mathbf{C}_p)$ не может равняться нулю. Следовательно, нам надо показать, что единственным многочленом степени не более $n-1$, таким что подстановка в него матрицы $\mathbf{C}_p$ приводит к вырожденной матрице, является нулевой многочлен.

Шаг 2. Легко проверить, что любой простой многочлен $p(x)$ обращается в нуль при подстановке в него его сопровождающей матрицы. Таким образом, $\mathbf{p}(\mathbf{C}_p) = \mathbf{0}$, где $\mathbf{0}$ обозначает нулевую $(n \times n)$-матрицу. Пусть $a(x)$ обозначает многочлен степени не более $n-1$, такой что матрица $\mathbf{a}(\mathbf{C}_p)$ вырожденна, и пусть $\mathbf{v}$ обозначает ненулевой вектор из нулевого пространства матрицы $\mathbf{a}(\mathbf{C}_p)$. Тогда, так как $p(x)$ является неприводимым многочленом степени n, то существуют многочлены $A(x)$ и $P(x)$, такие что

$$A(x)\, a(x) + P(x)\, p(x) = 1.$$

Следовательно,

$$[\mathbf{A}(\mathbf{C}_p)\, \mathbf{a}(\mathbf{C}_p) + \mathbf{P}(\mathbf{C}_p)\, \mathbf{p}(\mathbf{C}_p)]\, \mathbf{v} = \mathbf{I}\mathbf{v}.$$

Но $\mathbf{p}(\mathbf{C}_p) = \mathbf{0}$, так что мы имеем $\mathbf{A}(\mathbf{C}_p)\, \mathbf{a}(\mathbf{C}_p)\, \mathbf{v} = \mathbf{v}$, что противоречит выбору $\mathbf{v} : \mathbf{a}(\mathbf{C}_p)\, \mathbf{v} = \mathbf{0}$. Следовательно, за исключением нулевого многочлена, не существует многочлена $a(x)$ степени $n-1$ или меньше, такого что матрица $\mathbf{a}(\mathbf{C}_p)$ является вырожденной. Это завершает доказательство теоремы. □

Задачи

3.1. **а.** Комплексное умножение $(e + jf) = (a + jb)(c + jd)$ можно вычислить с тремя вещественными умножениями и пятью вещественными сложениями по алгоритму

$$e = (a - b)\, d + a\, (c - d),$$

$$f = (a - b)\, d + b\, (c + d).$$

Положить c и d константами и записать алгоритм умножения в матричной форме

$$\begin{bmatrix} e \\ f \end{bmatrix} = \mathbf{BDA} \begin{bmatrix} a \\ b \end{bmatrix},$$

где $\mathbf{A}$ и $\mathbf{B}$ представляют собой матрицы предсложений и постсложений, а $\mathbf{D}$ — диагональную матрицу.

б. 2-точечная циклическая свертка $s(x) = g(x)\, d(x) \pmod{x^2 - 1}$ может быть вычислена по алгоритму

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} g_0 + g_1 & 0 \\ 0 & g_0 - g_1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}.$$

Предположим, что многочлены $d(x)$ и $g(x)$ имеют комплексные коэффициенты. Сколько потребуется вещественных сложений и вещественных умножений для выполнения этого алгоритма, если комплексная арифметика реализуется обычным классическим способом?

в. Представить теперь входные и выходные данные в виде векторов длины четыре и выписать объединенный алгоритм с шестью вещественными умножениями. Сколько вещественных сложений содержит этот алгоритм?

3.2. При заданном устройстве для вычисления линейной свертки двух последовательностей длины n описать, как оно может быть использовано для вычисления взаимной корреляционной функции $\sum_{i=0}^{n-1} g_{i+j} d_i$ этих последовательностей.

3.3. Построить алгоритм фильтрации 4-точечной последовательности входных данных в КИО-фильтре с тремя отводами, используя для этого алгоритм Кука — Тоома для свертки.

3.4. **а.** Начиная с алгоритма вычисления 2-точечной линейной свертки

$$s(x) = g_1 d_1 x^2 + [g_1 d_1 + g_0 d_0 - (g_1 - g_0)(d_1 - d_0)]\, x + g_0 d_0,$$

построить алгоритм вычисления

$$s(x) = g(x)\, d(x) \pmod{x^2 - x + 1}.$$

б. Повторить задачу, начиная с алгоритма

$$s(x) = g_1 d_1 x^2 + [(g_1 + g_0)(d_1 + d_0) - g_1 d_1 - g_0 d_0]\, x + g_0 d_0.$$

Какой из алгоритмов заслуживает предпочтения?

3.5. Допустим, что у нас имеется устройство (жесткий модуль или подпрограмма) для вычисления 315-точечной циклической свертки, и что требуется пропустить 1000-точечный вектор данных через КИО-фильтр с 100 отводами. Описать, как надо разбить данные и организовать устройство свертки, чтобы получить требуемый ответ.

3.6. Построить алгоритм 4-точечной циклической свертки комплексных последовательностей, содержащий четыре комплексных умножения. Сравнить построенный алгоритм с алгоритмом 4-точечной циклической свертки вещественных последовательностей, используемым для свертки комплексных последовательностей.

3.7. Одним из способов определения поля комплексных чисел является расширение поля вещественных чисел с помощью неприводимого многочлена $p(x) = x^2 + 1$. Комплексное умножение при этом определяется равенством

$$e + jx = (a + bx)(c + dx) \pmod{x^2 + 1}.$$

Используя это определение, преобразуйте алгоритм линейной свертки в алгоритм умножения комплексных чисел.

3.8. **а.** Сколько умножений содержит алгоритм Винограда вычисления 16-точечной циклической свертки, если в качестве модулей используются только разложения

$$x^{16} - 1 = (x - 1)(x + 1)(x^2 + 1)(x^4 + 1)(x^8 + 1)?$$

б. Предположим, что допускается разложение многочлена над полем комплексных рациональных чисел, т. е. чисел вида $a + jb$, где a и b — рациональные числа. Сколько теперь потребуется комплексных умножений, если в качестве модулей допускаются многочлены, имеющие числа 1, $\pm j$ в качестве коэффициентов? (*Замечание*: теперь комплексные числа считаются внутренними переменными и умножение на j не считается умножением.)

в. Имеет ли алгоритм (б) преимущества перед алгоритмом (а)? (*Указание*: рассмотреть, является ли циклическая свертка вещественной или комплексной.)

3.9. Выписать всю последовательность равенств вычисления линейной (4×3)-свертки по алгоритму Винограда, основанному на многочлене

$$m(x) = x^2(x+1)(x-1)(x^2+1).$$

3.10. Для представления в ЭВМ чисел с двойной точностью можно использовать запись числа в виде многочлена $d_1x + d_0$. Для такой записи чисел с двойной точностью выписать алгоритм умножения без округления, содержащий три умножения и четыре сложения чисел с одинарной точностью.

3.11. Используя алгоритм

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \left\{ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} \right\},$$

и принцип трансформации [1]), построить пять новых алгоритмов. Чем они интересны?

3.12. Доказать, что для любого $n > 1$ в поле вещественных чисел лучший алгоритм вычисления

$$s(x) = g(x)\, d(x) \pmod{x^n + 1}$$

содержит больше умножений, чем лучший алгоритм вычисления

$$s(x) = g(x)\, d(x) \pmod{x^n - 1}.$$

3.13. Найти такие целые числа n и n', что $n < n'$, но n-точечная циклическая свертка требует большего числа умножений, чем n'-точечная циклическая свертка.

3.14. Используя знание быстрых алгоритмов свертки, построить алгоритм 3-точечного преобразования Фурье, $V_k = \sum_{l=0}^{2} \omega^{lk} v_l$, $k = 0, 1, 2$, содержащий только два нетривиальных вещественных умножения.

3.15. Построить алгоритмы для следующих вычислений:

а. $s(x) = g(x)\, d(x) \pmod{x^3 + x + 1}$.

б. $s(x) = g(x)\, d(x) \pmod{x^3}$.

в. $s(x) = g(x)\, d(x) \pmod{x^3 + x^2 + 1}$.

3.16. **а.** Сколько вещественных умножений требует описанный в разд. 3.7 метод вычисления циклической *комплексной* свертки по модулю $x^8 - 1$?

б. Сколько комплексных умножений требуется по теореме Винограда о сложности для вычисления в поле комплексных чисел циклической свертки по модулю многочлена $x^8 - 1$?

в. Объяснить все кажущиеся несоответствия.

[1]) См. теорему 9.2.1. — *Прим. перев.*

3.17. **а.** Предполагая, что все переменные являются независимыми вещественными величинами, доказать, что для вычисления двух комплексных произведений

$$e + jf = (a + jb)(c + jd),$$
$$e' + jf' = (a' + jb')(c' + jd'),$$

требуется выполнить шесть вещественных умножений.

б. Сколько потребуется умножений для одновременного вычисления следующих двух комплексных призведений:

$$e + jf = (a + jb)(c + jd),$$
$$e' + jf' = (a - jb)(c + jd)?$$

В данном случае переменные повторяются.

в. Сколько потребуется умножений для одновременного вычисления следующих двух комплексных произведений:

$$e + jf = (a + jb)(c + jd),$$
$$e' + jf' = (a - jb)(c - jd)?$$

3.18. **а.** Доказать, что вычисление

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} g_0 & g_1 \\ g_1 & g_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix},$$

где все переменные являются комплексными, требует шести вещественных умножений.

б. Доказать, что вычисление

$$\begin{bmatrix} s_0' + js_0'' \\ s_1' + js_1'' \end{bmatrix} = \begin{bmatrix} g_0' & jg_1'' \\ jg_1' & g_0'' \end{bmatrix} \begin{bmatrix} d_0' + jd_0'' \\ d_1' + jd_1'' \end{bmatrix},$$

где все переменные являются вещественными, также требует шести вещественных умножений. Таким образом, равенство нулю величин g_0'' и g_1', как и в задаче а, не уменьшает числа необходимых умножений.

в. Доказать, что вычисление

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} g_0 & g_1 \\ -g_1 & g_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix},$$

где все переменные являются комплексными, требует шести вещественных умножений.

г. Доказать, что вычисление

$$\begin{bmatrix} s_0' + js_0'' \\ s_1' + js_1'' \end{bmatrix} = \begin{bmatrix} g_0' & jg_1' \\ -jg_1'' & g_0'' \end{bmatrix} \begin{bmatrix} d_0' + jd_0'' \\ d_1' + jd_1'' \end{bmatrix},$$

где все переменные вещественны, требует четырех вещественных умножений. В этом случае равенство нулю переменных g_0'' и g_1' уменьшает необходимое число умножений.

3.19. Построить алгоритм 4-точечной циклической свертки над полем комплексных чисел. Сравнить этот алгоритм с алгоритмом, основанным на БПФ и теореме о свертке.

Замечания

Без какой-либо общей теории первые быстрые алгоритмы сверток коротких длин были описаны Агарвалом и Кули [1] (1977) в связи с развитыми ими многомерными гнездовыми методами. Описанный нами общий метод был предложен Виноградом [2] (1978). Примеры использования этого метода приведены в монографии [4] 1980 г. Виноград также доказал [3] (1977) теоремы о сложности рассмотренных алгоритмов свертки для полей комплексных и вещественных чисел.

Глава 4

БЫСТРЫЕ АЛГОРИТМЫ ДИСКРЕТНОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ

Построение набора методов вычисления дискретного преобразования Фурье является одной из основных наших целей. Мы построим много таких методов, каждый из которых обладает различными преимуществами перед другими и каждый из которых наиболее пригоден при соответствующих обстоятельствах. Основных стратегий имеется две. Одна из них состоит в сведении дискретного преобразования Фурье к свертке, которая затем вычисляется описанными в предыдущей главе методами. Другая стратегия состоит в переходе от одномерного преобразования Фурье к двумерному, которое вычисляется проще. Хорошие алгоритмы вычисления дискретного преобразования Фурье для минимизации сложности вычислений используют обе эти стратегии.

Многие из изучаемых нами алгоритмов не зависят от частных особенностей поля, над которым определено дискретное преобразование Фурье. Поэтому мы часто будем опускать указание на конкретное поле и в этом случае алгоритм справедлив для любого поля. В других случаях, хотя общая идея алгоритма и не зависит от поля, над которым рассматривается дискретное преобразование Фурье, но некоторые детали алгоритма определяются спецификой поля. В этих случаях алгоритм должен строиться для конкретного интересующего нас поля.

4.1. Алгоритм Кули—Тьюки быстрого преобразования Фурье

Преобразование Фурье вектора $\mathbf{v}$,

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i,$$

в том виде, как оно записано, требует порядка n^2 умножений и n^2 сложений. Если число n является составным, то имеется несколько способов перейти от этого преобразования Фурье к двумерному преобразованию или к чему-либо ему аналогичному. Это позволяет перевести вычисления в более эффективную форму,

БПФ-алгоритм Кули—Тьюки (1965)

$$n = n'n''$$

$$i = i' + n'i'' \qquad \begin{matrix} i' = 0, \ldots, n' - 1; \\ i'' = 0, \ldots, n'' - 1. \end{matrix}$$

$$k = n''k' + k'' \qquad \begin{matrix} k' = 0, \ldots, n' - 1; \\ k'' = 0, \ldots, n'' - 1; \end{matrix}$$

$$V_{k',k''} = \sum_{i=0}^{n'-1} \beta^{i'k'} \left[\omega^{i'k''} \sum_{i''=0}^{n''-1} \gamma^{i''k''} v_{i',i''} \right]$$

Число умножений

$$\approx n(n' + n'') + n$$

Рис. 4.1. БПФ-алгоритм Кули—Тьюки.

но за это приходится платить усложнением структуры. Алгоритмы подобного сорта известны под общим названием быстрого преобразования Фурье (БПФ). На рис. 4.1 приведена структура БПФ-алгоритма Кули—Тьюки, изучаемого в данном разделе. Рис. 4.1 интересно сравнить с рис. 4.8 из разд. 4.3, на котором приведена структура БПФ-алгоритма Гуда—Томаса.

Для построения БПФ-алгоритма Кули—Тьюки предположим, что $n = n'n''$. В выражении для преобразования Фурье сделаем следующую замену записи каждого индекса:

$$\begin{aligned} i &= i' + n'i'', & i' &= 0, \ldots, n' - 1, \\ & & i'' &= 0, \ldots, n'' - 1, \\ k &= n''k' + k'', & k' &= 0, \ldots, n' - 1, \\ & & k'' &= 0, \ldots, n'' - 1. \end{aligned}$$

Тогда

$$V_{n''k'+k''} = \sum_{i''=0}^{n''-1} \sum_{i'=0}^{n'-1} \omega^{(i'+n'i'')(n''k'+k'')} v_{i'+n'i''}.$$

Раскроем скобки в показателе степени и положим $\omega^{n'} = \gamma$ и $\omega^{n''} = \beta$. Так как порядок элемента ω равен $n'n''$, то член $\omega^{n'n''i''k'} = 1$ можно опустить. Определим теперь двумерные переменные, которые тоже обозначим через $\mathbf{v}$ и $\mathbf{V}$, задавая их равенствами

$$v_{i',i''} = v_{i'+n'i''}, \qquad \begin{matrix} i' = 0, \ldots, n' - 1, \\ i'' = 0, \ldots, n'' - 1, \end{matrix}$$

$$V_{k',k''} = V_{n''k'+k''}, \qquad \begin{matrix} k' = 0, \ldots, n' - 1, \\ k'' = 0, \ldots, n'' - 1. \end{matrix}$$

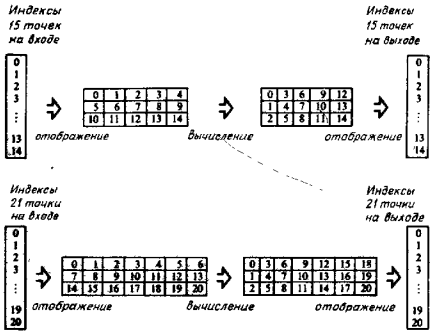


Рис. 4.2. Примеры переиндексации по методу Кули—Тьюки.

При этом векторы входных и выходных данных преобразуются в двумерные массивы. Заметим, что компоненты преобразования $\mathbf{V}$ упорядочены в таблице не так, как компоненты сигнала $\mathbf{v}$. Это упорядочивание известно под названием адресного тасования. В терминах двумерных переменных формула преобразуется к виду

$$V_{k',k''} = \sum_{i'=0}^{n'-1} \beta^{i'k'} \left[\omega^{i'k''} \sum_{i=0}^{n''-1} \gamma^{i''k''} v_{i'',i''} \right].$$

Хотя для понимания эта формула более трудна, чем исходная, но число входящих в нее умножений и сложений существенно меньше. А именно, она содержит не более $n(n' + n'' + 1)$ комплексных умножений и $n(n' + n'' - 2)$ комплексных сложений вместо n^2 комплексных умножений и n^2 комплексных сложений исходной формулы.

Визуально БПФ-алгоритм Кули—Тьюки выглядит как отображение двумерной таблицы в двумерную таблицу, как показано на рис. 4.2 для примеров с $n = 15$ и $n = 21$. Вычисления состоят из n''-точечного дискретного преобразования Фурье каждого столбца, поэлементного умножения всех элементов таблицы соответственно на $\omega^{i'k''}$ и n'-точечного дискретного преобразования Фурье каждой строки.

Чтобы для комплексного входного вектора $\mathbf{v}$ подсчитать полное число комплексных умножений и комплексных сложений, предположим, что внутреннее и внешнее преобразования Фурье вычисляются обычным способом, содержащим соответственно $(n'')^2$ и $(n')^2$ комплексных умножений и $n''(n''-1)$ и $n'(n'-1)$

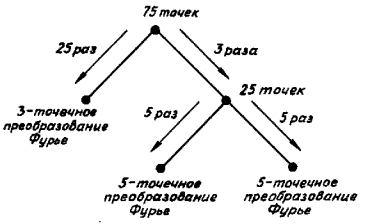


Рис. 4.3. Структура 75-точечного БПФ-алгоритма Кули — Тьюки.

комплексных сложений. Каждое из этих преобразований Фурье используется n' и n'' раз соответственно. Помимо этого имеется $n'n''$ комплексных умножений на множители «регулировки» $\omega^{i'k''}$. Таким образом, полное число умножений, $M_c(n)$, и полное число сложений, $A_c(n)$, в данном случае равны

$$M_c(n) = n'(n'')^2 + n''(n')^2 + n'n'' = n(n' + n'' + 1),$$
$$A_c(n) = n'n''(n'' - 1) + n''n'(n' - 1) = n(n' + n'' - 2),$$

что и утверждалось ранее. Среди умножений имеются и тривиальные умножения на 1. Это происходит всякий раз, когда в множителе регулировки целые числа i' или k'' в показателе обращаются в нуль. При желании алгоритм можно перестроить так, чтобы выкинуть эти умножения; тогда число комплексных умножений станет равным

$$M_c(n) = n(n' + n'') + (n' - 1)(n'' - 1) =$$
$$= (n - 1)(n' + n'') + (n + 1).$$

Внешнее и внутреннее преобразования Фурье в свою очередь могут быть вычислены с помощью быстрых алгоритмов, не обязательно с помощью БПФ-алгоритма Кули—Тьюки. Тогда для числа комплексных сложений и для числа комплексных умножений, входящих в БПФ-алгоритм Кули—Тьюки, получаем соотношения

$$M_c(n) = n'M_c(n'') + n''M_c(n') + n,$$
$$A_c(n) = n'A_c(n'') + n''A_c(n'),$$

где новые члены, стоящие в правых частях равенств, обозначают числа комплексных умножений и комплексных сложений, входящих соответственно в n'-точечный и n''-точечный быстрые алгоритмы преобразования Фурье. Конечно, если n' или n'' опять является составным числом, то меньшее преобразование опять

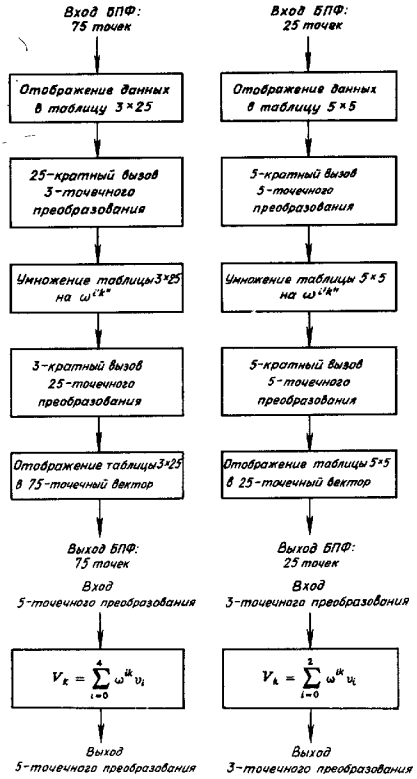


Рис. 4.4. Разбиение БПФ-алгоритма Кули—Тьюки на подпрограммы.

можно вычислять с помощью БПФ-алгоритма Кули—Тьюки. На этом пути преобразование длины $n = \prod_l n_l$ может быть представлено в такой форме, в которой требуется выполнения $n \sum_l n_l$ комплексных умножений. На рис. 4.3 показан один из многих способов разбиения 75-точечного преобразования Фурье. Каждый из узлов этого рисунка можно представлять себе как обращение к подпрограмме. Если вычисления реализуются с помощью такого разбиения на подпрограммы, то один из способов организации программного вычисления показан на рис. 4.4. На самом нижнем уровне вычислений находятся 3-точечное и 5-точечное преобразования Фурье, выполняемые буквально. Позже мы построим малый БПФ-алгоритм Винограда, который можно использовать на этом уровне в качестве подпрограммы. Малые БПФ-алгоритмы Винограда представляют собой сильно оптимизированные процедуры вычисления преобразования Фурье для длин, являющихся простым числом или степенью простого числа.

4.2. Алгоритм Кули—Тьюки по малому основанию

Во многих приложениях, в которых используется алгоритм Кули—Тьюки, длина преобразования равна степени двух или четырех. Для построения БПФ-алгоритма длина 2^m преобразования представляется в виде $2 \cdot 2^{m-1}$ или $2^{m-1} \cdot 2$. В этом случае говорят о БПФ-алгоритме Кули—Тьюки по основанию два [1]). Аналогично, длина 4^m преобразования разлагается в виде $4 \cdot 4^{m-1}$ или $4^{m-1} \cdot 4$, и тогда говорят о БПФ-алгоритме Кули—Тьюки по основанию четыре. Если в 2^m-точечном алгоритме Кули—Тьюки по основанию два полагается $n' = 2$ и $n'' = 2^{m-1}$, то он называется БПФ-алгоритмом Кули—Тьюки по основанию два *с прореживанием по времени*. Используя тот факт, что $\beta = \omega^{n/2} = -1$, в этом случае уравнения, задающие БПФ, можно записать в следующем простом виде:

$$V_k = \sum_{i=0}^{n/2-1} \omega^{2ik} v_{2i} + \omega^k \sum_{i=0}^{n/2-1} \omega^{2ik} v_{2i+1},$$

$$V_{k+n/2} = \sum_{i=0}^{n/2-1} \omega^{2ik} v_{2i} - \omega^k \sum_{i=0}^{n/2-1} \omega^{2ik} v_{2i+1}.$$

$$k = 0, \ldots, n/2 - 1,$$

[1]) Термин «по основанию два» относится к факту записи индексов по основанию два. Компоненты данных могут быть представлены в любой системе счисления, в частности по основанию два.

Прореживание по времени разбивает множество компонент входного вектора на два подмножества: множество компонент с четными индексами и множество компонент с нечетными индексами. Множество компонент выходного вектора разбивается при этом на множество первых $n/2$ компонент и множество вторых $n/2$ компонент.

2^m-точечный БПФ-алгоритм Кули—Тьюки по основанию два, в котором $n' = 2^{m-1}$ и $n'' = 2$, называется БПФ-алгоритмом Кули—Тьюки по основанию два *с прореживанием по частоте*. Уравнения БПФ в этом случае преобразуются к виду

$$V_{2k'} = \sum_{t'=0}^{n/2-1} (v_{t'} + v_{t'+n/2})\,\omega^{2t'k'},$$

$$V_{2k'+1} = \sum_{t'=0}^{n/2-1} (v_{t'} - v_{t'+n/2})\,\omega^{t'}\omega^{2t'k'}, \qquad k' = 0, \ldots, n/2-1.$$

Прореживание по частоте разбивает компоненты входного вектора на два подмножества, содержащие соответственно первые $n/2$ компонент и вторые $n/2$ компонент. Компоненты выходного вектора разбиваются на подмножество компонент с четными индексами и подмножество компонент с нечетными индексами.

Алгоритм с прореживанием по времени и алгоритм с прореживанием по частоте отличаются структурой и последовательностью вычислений, хотя имеют одно и то же число операций. Характеристики алгоритмов совпадают, но пользователь может предпочесть один из них из соображений реализации. Мы подробно рассмотрим только характеристики алгоритма с прореживанием по времени.

Алгоритм с прореживанием по времени сводит n-точечное преобразование Фурье к двум $(n/2)$-точечным преобразованиям Фурье с некоторыми дополнительными сложениями и умножениями. Часть из умножений представляют собой умножения на единицу или $\pm j$. Они тривиальны и не требуют действительного вычисления. Чтобы обойтись без тривиальных умножений в алгоритме, их следует обрабатывать отдельно. Иногда конструктор предпочитает включить в процедуру вычисления все умножения, даже тривиальные.

Алгоритм с прореживанием по времени работает рекурсивно, разбивая на каждом шаге n-точечное преобразование на два $(n/2)$-точечных преобразования, которые, в свою очередь, разбиваются точно таким же образом. Из уравнений ясно видно, что число $M_c(n)$ комплексных умножений n-точечного БПФ удовлетворяет рекуррентному уравнению

$$M_c(n) = 2M_c(n/2) + n/2,$$

а число $A_c(n)$ комплексных сложений удовлетворяет рекуррентному уравнению

$$A_c(n) = 2A_c(n/2) + n,$$

где n равно степени двойки. Решения этих уравнений даются равенствами

$$M_c(n) = (n/2)\log_2 n, \quad A_c(n) = n\log_2 n.$$

Комплексные умножения можно реализовать алгоритмом с четырьмя вещественными умножениями и двумя вещественными сложениями. В этом случае характеристики БПФ по основанию два даются равенствами

$$M_R(n) = 2n\log_2 n, \quad A_R(n) = 3n\log_2 n.$$

Альтернативный алгоритм выполнения комплексного умножения содержит три вещественных умножения и три вещественных сложения. В этом случае характеристики имеют вид

$$M_R(n) = (3/2)\,n\log_2 n, \quad A_R(n) = (7/2)\,n\log_2 n.$$

Теперь предположим, что мы хотим построить алгоритм, в котором выброшены все тривиальные умножения. Тогда полученные нами характеристики сложности алгоритма улучшатся. Тщательный анализ алгоритма показывает, что все умножения на самом внутреннем шаге алгоритма тривиальны и имеют вид умножений на $(-1)^k$ для $k = 0, 1$; все умножения на следующем шаге тривиальны и имеют вид умножений на j^k для $k = 0, 1, 2, 3$; на последующих шагах число тривиальных умножений равно $n/4$, $n/8$, ... Следовательно, число комплексных умножений равно

$$M_c(n) = (n/2)(-3 + \log_2 n) + 2.$$

Используя для реализации комплексного умножения алгоритм с четырьмя вещественными умножениями и двумя вещественными сложениями, для БПФ-алгоритма по основанию два получаем

$$M_R(n) = 2n(-3 + \log_2 n) + 8,$$
$$A_R(n) = 3n(-1 + \log_2 n) + 4.$$

Используя для комплексного умножения алгоритм с тремя вещественными умножениями и тремя вещественными сложениями, получаем характеристики

$$M_R(n) = (3/2)\,n(-3 + \log_2 n) + 6,$$
$$A_R(n) = (1/2)\,n(-9 + 7\log_2 n) + 6.$$

Еще немного можно улучшить алгоритм, если воспользоваться свойством симметрии тригонометрических функций. Заметим, что

$$\omega^{n/8} = (1 - j)/\sqrt{2}.$$

Умножение на эту комплексную константу требует всего двух вещественных умножений и двух вещественных сложений. Таких умножений на шагах 3, 4, ... алгоритм содержит соответственно $n/4$, $n/8$, ... Их можно реализовать с помощью специальной процедуры умножения. При этом в зависимости от выбранной реализации комплексного умножения характеристики алгоритма будут даваться равенствами

$$M_R(n) = n(-7 + 2\log_2 n) + 12,$$
$$A_R(n) = 3n(-1 + \log_2 n) + 4$$

или

$$M_R(n) = (1/2)\, n(-10 + 3\log_2 n) + 8.$$
$$A_R(n) = (1/2)\, n(-10 + 7\log_2 n) + 8.$$

Как мы видим, число вариантов алгоритма Кули—Тьюки весьма велико, но мы еще не перечислили все возможности; имеются и другие возможности улучшения алгоритма. На рис. 4.5 приведены характеристики некоторых БПФ-алгоритмов Кули—Тьюки. В таблицу помимо обычной формы алгоритма Кули—Тьюки включен БПФ-алгоритм Рейдера—Бреннера. Этот БПФ-алгоритм является модификацией алгоритма Кули—Тьюки, основанной на том, что с помощью переупорядочивания уравнений БПФ некоторые умножения на комплексные константы можно заменить умножением на вещественные константы.

Алгоритм Рейдера—Бреннера можно строить, исходя из уравнений для алгоритма с прореживанием по времени, но мы предпочитаем отталкиваться от уравнений для алгоритма с прореживанием по частоте:

$$V_{2k} = \sum_{l=0}^{n/2-1} (v_l + v_{l+n/2})\,\omega^{2lk},$$
$$V_{2k+1} = \sum_{l=0}^{n/2-1} (v_l - v_{l+n/2})\,\omega^{l}\omega^{2lk}. \qquad k = 0, \ldots, n/2-1,$$

Определим рабочий вектор **a** равенствами

$$a_j = \begin{cases} 0, & i = 0, \\ (v_l - v_{l+n/2})/[2j\sin(2\pi i/n)], & i = 1, \ldots, n/2-1, \end{cases}$$

и положим

$$A_k = \sum_{i=0}^{n/2-1} \omega^{2ik} a_i, \quad k = 0, \ldots, n/2-1.$$

Так как

$$A_{k+1} - A_k = \sum_{i=0}^{n/2-1} \omega^{2ik} a_i(\omega^{2i} - 1) =$$
$$= \sum_{i=0}^{n/2-1} \omega^{2ik}\omega^{i} a_i 2_j \sin(2\pi i/n) = \sum_{i=1}^{n/2-1} (v_i - v_{i+n/2})\,\omega^{i}\omega^{2ik},$$

Длина n	Основной алгоритм комплексного БПФ по основанию два		Полностью оптимизированный алгоритм комплексного БПФ по основанию два		Алгоритм Рейдера—Бреннера комплексного БПФ по основанию два	
	Число вещественных умножений	Число вещественных сложений	Число вещественных умножений	Число вещественных сложений	Число вещественных умножений	Число вещественных сложений
8	48	72	4	52	4	64
16	128	192	24	152	20	192
32	320	480	88	408	68	512
64	768	1 152	264	1 032	196	1 280
128	1 792	2 688	712	2 504	516	3 072
256	4 096	6 144	1 800	5 896	1 284	7 168
512	9 216	13 824	4 360	13 576	3 076	16 384
1024	20 480	30 720	10 248	30 728	7 172	36 864
2048	45 056	67 584	23 560	68 616	16 388	81 920
4096	98 304	147 456	53 256	151 560	36 868	180 224

Примечания: 1) В полностью оптимизированном алгоритме комплексное умножение проводится посредством трех вещественных умножений и трех вещественных сложений; 2) тривиальные умножения (на ± 1 или на $\pm i$) не учитываются; 3) полностью используется симметрия тригонометрических функций.

Рис. 4.5. Характеристики некоторых БПФ-алгоритмов Кули—Тьюки.

$$a_i = \begin{cases} 0 & i = 0 \\ j\,(v_{i+n/2} - v_i)/[2\sin(2\pi i/n)] & i = 1, \ldots, n/2-1 \end{cases}$$

$$A_k = \sum_{i=0}^{n/2-1} a_i \omega^{2ik} \quad k = 0, \ldots, n/2-1$$

$$V_{2k} = \sum_{i=0}^{n/2-1} (v_i + v_{i+n/2})\,\omega^{2ik} \quad k = 0, \ldots, n/2-1$$

$$V_{2k+1} = A_{k+1} - A_k + (v_0 - v_{n/2}) \quad k = 0, \ldots, n/2-1$$

Рис. 4.6. БПФ-алгоритм Рейдера—Бреннера.

то величины V_{2k+1} и A_k связаны равенствами

$$V_{2k+1} = A_{k+1} - A_k + (V_0 - V_{n/2}).$$

Таким образом, умножение на комплексные константы ω^t удалось заменить на умножения на мнимые константы $[2j\sin(2\pi i/n)]^{-1}$, что уменьшает вычислительную сложность. Необходимо, однако, проследить, чтобы не произошло переполнение по длине слова потому, что когда n велико, а i мало, новые константы становятся весьма большими.

Вкратце форма алгоритма показана на рис. 4.6. Каждое из двух $(n/2)$-точечных дискретных преобразований Фурье можно в свою очередь разбить точно таким же образом. На каждом шаге алгоритма требуется $(n/2) - 2$ умножений комплексных чисел на мнимые, для чего в итоге используется $n - 4$ вещественных умножений. (Мы предпочли избежать умножения на $1/2$ при i, равном $n/4$.) На каждом шаге алгоритма всего имеется $2n$ комплексных или $4n$ вещественных сложений. Характеристики алгоритма Рейдера—Бреннера описываются рекуррентными уравнениями

$$M_R(n) = n - 4 + 2M_R(n/2), \quad A_R(n) = 4n + 2A_R(n/2)$$

при начальных условиях $M_R(4) = 0$, $A_R(4) = 16$. Здесь не учтены имеющиеся на двух самых внутренних шагах умножения на (± 1) и $(\pm j)$.

Вместе с другими формами БПФ-алгоритма Кули—Тьюки на рис. 4.5 приведены характеристики БПФ-алгоритма Рейдера—Бреннера. Отметим, что полностью оптимизированные БПФ-алгоритмы по основанию 2 для малых длин содержат меньше сложений. Это позволяет предложить гибридный алгоритм: разбивать преобразование Фурье по алгоритму Рейдера—Бреннера до тех пор, пока не достигнем длины 16, а затем переходить на полностью оптимизированный алгоритм. Использование на самом внутреннем этапе рассматриваемого в разд. 4.6 16-точечного БПФ-алгоритма Винограда позволяет еще больше улучшить

результаты. Характеристики гибридного алгоритма описываются теми же рекуррентными уравнениями, но с начальными условиями $M_R(16) = 20$, $A_R(16) = 148$.

Очень популярны также БПФ-алгоритмы Кули—Тьюки по основанию 4. Они могут быть использованы в случаях, когда длина преобразования n равна степени 4, и получаются ее разложением в виде $4 \cdot 4^{m-1}$ или $4^{m-1} \cdot 4$. Мы рассмотрим БПФ-алгоритм Кули—Тьюки по основанию 4 с прореживанием по времени. Уравнения этой формы БПФ можно получить простой подстановкой $n' = 4$ и $n'' = n/4$ в приведенные на рис. 4.1 общие уравнения БПФ-алгоритма Кули—Тьюки. При $k = 0, \ldots, n/4 - 1$ соответственно имеем

$$\begin{bmatrix} V_k \\ V_{k+n/4} \\ V_{k+n/2} \\ V_{k+3n/4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i} \\ \omega^k \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i+1} \\ \omega^{2k} \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i+2} \\ \omega^{3k} \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i+3} \end{bmatrix}.$$

Для каждого из $n/4$ рассматриваемых значений индекса k такое матричное уравнение определяет четыре компоненты преобразования. Этот метод позволяет n-точечное преобразование Фурье заменить четырьмя $(n/4)$-точечными преобразованиями Фурье и некоторыми дополнительными вычислениями. Из выписанного уравнения следует, что для каждого k имеется только три различных комплексных умножения и 12 комплексных сложений. Самый внутренний шаг — 4-точечное преобразование Фурье — вообще не содержит умножений.

Число сложений можно еще уменьшить. Для этого перепишем последние уравнения для $k = 0, \ldots, n/4 - 1$ в виде

$$\begin{bmatrix} V_k \\ V_{k+n/4} \\ V_{k+n/2} \\ V_{k+3n/4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i} \\ \omega^k \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i+1} \\ \omega^{2k} \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i+2} \\ \omega^{3k} \sum_{i=0}^{n/4-1} \omega^{4ik} v_{4i+3} \end{bmatrix}.$$

Разложив исходную (4×4)-матрицу в произведение двух, мы получили 8 сложений вместо 12. Это дает окончательную форму БПФ-алгоритма Кули—Тьюки по основанию 4. Характеристики алгоритма описываются равенствами

$$M_c(n) = (3/4)\, n\,(\log_4 n - 1) = (3/8)\, n\,(\log_2 n - 2),$$
$$A_c(n) = 2n \log_4 n = n \log_2 n$$

для числа комплексных умножений и сложений соответственно.

Если для вычисления комплексных умножений используется алгоритм с тремя вещественными умножениями и тремя вещественными сложениями, то характеристики описываются равенствами

$$M_R(n) = (9/8)\, n\,(\log_2 n - 2), \quad A_R(n) = (25/8)\, n \log_2 n - (9/4)\, n.$$

Можно получить и лучшие результаты, если вложить в программу способность выловить все умножения на (±1) и на $(\pm j)$ или на нечетные степени элемента $(1/\sqrt{2})\,(1-j)$, поскольку они не требуют трех вещественных умножений, и убрать их из алгоритма. Тогда характеристики будут даваться равенствами

$$M_R(n) = \frac{9}{8}\, n \log_2 n - \frac{43}{12}\, n + \frac{16}{3},$$
$$A_R(n) = \frac{25}{8}\, n \log_2 n - \frac{43}{12}\, n + \frac{16}{3}.$$

Характеристики БПФ-алгоритма Кули—Тьюки подытожены на рис. 4.7.

Длина n	Основной алгоритм комплексного БПФ по основанию 4: Число вещественных умножений	Основной алгоритм комплексного БПФ по основанию 4: Число вещественных сложений	Полностью оптимизированное * комплексное БПФ по основанию 4: Число вещественных умножений	Полностью оптимизированное * комплексное БПФ по основанию 4: Число вещественных сложений
4	0	16	0	16
16	36	164	20	148
64	288	1 128	208	976
256	1 728	5 824	1 392	5 488
1024	9 216	29 696	7 856	28 336
4096	46 080	144 384	40 642	138 928

* Комплексное умножение с использованием 3 вещественных умножений и 3 вещественных сложений.
Тривиальные умножения на ±1 или $\pm j$ не учитываются.

Рис. 4.7. Характеристики некоторых БПФ-алгоритмов по основанию 4.

4.3. Алгоритм Гуда—Томаса быстрого преобразования Фурье

Алгоритм Гуда—Томаса с использованием простых делителей представляет собой БПФ-алгоритм второго типа. Концептуально он несколько сложнее алгоритма Кули—Тьюки, но в вычислительном отношении несколько проще. Показанный на рис. 4.8 алгоритм Гуда—Томаса представляет собой другой способ отображения линейной последовательности из $n = n'n''$ целых чисел в $(n' \times n'')$-таблицу, преобразующего одномерное преобразование Фурье в двумерное преобразование Фурье. Лежащая в основе отображения идея сильно отличается от идеи алгоритма Кули—Тьюки. Теперь числа n' и n'' должны быть взаимно простыми, а основой отображения линейной последовательности в таблицу служит китайская теорема об остатках. Обращаясь к рис. 4.9, мы видим, как переупорядочены входные данные. В двумерную таблицу они выписываются, начиная с верхнего левого угла, вдоль «расширенной диагонали». Поскольку число строк таблицы взаимно просто с числом ее столбцов, расширенная диагональ последовательно проходит через все элементы таблицы. После выполнения над этой таблицей двумерного преобразования Фурье компоненты преобразования оказываются записанными в двумерной таблице по иному правилу, чем были записаны компоненты преобразуемого вектора. Способ упорядочивания входной и выходной таблиц будет описан ниже.

БПФ-алгоритм Гуда—Томаса (1960—1963)

$n = n'n''$, где n' и n'' взаимно просты

Перестановка входных индексов:

$$\left.\begin{aligned} i' &= i \pmod{n'} \\ i'' &= i \pmod{n''} \end{aligned}\right\} \leftrightarrow \left\{\begin{aligned} &i = i'N''n'' + i''N'n' \pmod{n}, \\ &\text{где} \\ &N'n' + N''n'' = 1. \end{aligned}\right.$$

Перестановка выходных индексов:

$$\left.\begin{aligned} k' &= N''k \pmod{n'} \\ k'' &= N'k \pmod{n''} \end{aligned}\right\} \leftrightarrow k = n''k' + n'k'' \pmod{n},$$

$$V_{k',k''} = \sum_{i'=0}^{n'-1} \beta^{i'k'} \left[\sum_{i''=0}^{n''-1} \gamma^{i''k''} v_{i',i''} \right].$$

Число умножений $\approx n\,(n' + n'')$

Рис. 4.8. БПФ-алгоритм Гуда—Томаса.

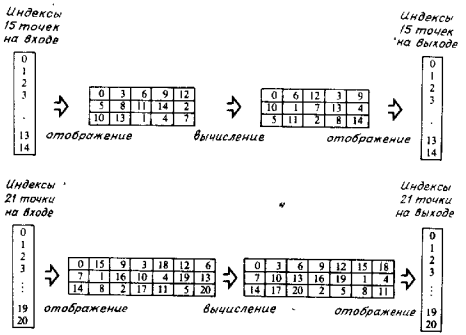


Рис. 4.9. Примеры переиндексации по методу Гуда—Томаса.

Построение БПФ-алгоритма Гуда—Томаса основано на китайской теореме об остатках для целых чисел. Входные индексы задаются вычетами по правилу

$$i' = i \pmod{n'}, \quad i'' = i \pmod{n''}.$$

Это правило представляет собой отображение индекса i на расширенную диагональ двумерной таблицы, элементы которой занумерованы парами индексов (i', i''). Согласно китайской теореме об остатках, существуют такие целые числа N' и N'', что выполняется равенство

$$i = i'N''n'' + i''N'n' \pmod{n},$$

где

$$N'n' + N''n'' = 1.$$

Выходные индексы определяются несколько иначе. Пусть

$$k' = N''k \pmod{n'}, \qquad k'' = N'k \pmod{n''}.$$

Эти равенства можно переписать в эквивалентном виде

$$k' = ((N'' \mod n')\, k \mod n'), \quad k'' = ((N' \mod n'')\, k \mod n'').$$

Выходные индексы k вычисляются по правилу

$$k = n''k' + n'k'' \pmod{n}.$$

Для проверки этого равенства выпишем:

$$k = n''(N''k + Q_1 n') + n'(N'k + Q_2 n'') \pmod{n'n''} =$$
$$= k(n''N'' + n'N') \pmod{n} = k.$$

В этих новых индексных обозначениях формула

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i$$

преобразуется к виду

$$V_{n''k'+n'k''} = \sum_{i''=0}^{n''-1} \sum_{i'=0}^{n'-1} \omega^{(i'N''n''+i''N'n')(n''k'+n'k'')} v_{i'N''n''+i''N'n'}.$$

Выполним умножения в показателе степени. Поскольку порядок элемента ω равен $n'n''$, то члены с этим показателем пропадают. Тогда выписанное выше преобразование индексов для элементов входной и выходной таблиц дает

$$V_{k',k''} = \sum_{i'=0}^{n'-1} \sum_{i''=0}^{n''-1} \omega^{N''(n'')^2 i'k'} \omega^{N'(n')^2 i''k''} v_{i',i''} =$$
$$= \sum_{i'=0}^{n'-1} \sum_{i''=0}^{n''-1} \beta^{i'k'} \gamma^{i''k''} v_{i',i''},$$

где $\beta = \omega^{N''(n'')^2}$ и $\gamma = \omega^{N'(n')^2}$. Элементы β и γ являются простыми корнями из единицы степеней n' и n'', задающими n'-точечное преобразование Фурье и n''-точечное преобразование Фурье соответственно. Чтобы увидеть это, достаточно заметить, что $\beta = (\omega^{n''})^{N''n''}$. Так как $\omega^{n''} = e^{-j2\pi/n'}$ и $N''n'' = 1$ по модулю n', то $\beta = e^{-j2\pi/n'}$. Аналогичное утверждение справедливо и для элемента γ.

Уравнение теперь записано в форме двумерного $(n' \times n'')$-точечного преобразования Фурье. Число умножений и число сложений равно примерно $n(n' + n'')$. Преобразование Фурье по строкам и по столбцам, если соответствующая размерность задается составным числом, можно в свою очередь упростить, применяя алгоритм БПФ. Таким образом, если длина n преобразования разлагается в произведение простых множителей n_l, то описанная форма БПФ-алгоритма требует примерно $n \sum_l n_l$ умножений и столько же сложений.

Для вычисления преобразования Фурье можно пользоваться как БПФ-алгоритмом Кули—Тьюки, так и БПФ-алгоритмом Гуда—Томаса. Можно даже строить алгоритмы, содержащие БПФ-алгоритм Кули—Тьюки и БПФ-алгоритм Гуда—Томаса одновременно. Например, используя БПФ-алгоритм Гуда—Томаса, можно 63-точечное преобразование Фурье разбить на 7-точечное и 9-точечное; используя далее БПФ-алгоритм Кули—Тьюки, 9-точечное преобразование можно разбить на два 3-точечных преобразования. Вычисления при этом преобразуются в форму, аналогичную трехмерному $(3 \times 3 \times 7)$-преобразованию

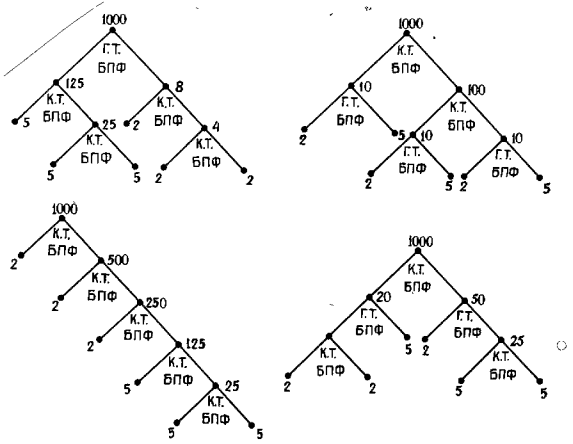


Рис. 4.10. Некоторые способы построения 1000-точечного преобразования Фурье.

Фурье. На рис. 4.10 показано несколько способов разбиения 1000-точечного преобразования Фурье на преобразования меньшего объема. Каждый из способов содержит 2-точечный модуль три раза и 5-точечный модуль три раза. Но все процедуры существенно различны, так как малые модули используются в разной последовательности. Число умножений и сложений, чувствительность алгоритма к погрешностям вычислений и легкость реализации алгоритма различны.

4.4. Алгоритм Герцеля

Значение одной компоненты преобразования Фурье можно вычислить по *правилу Горнера*, дающему один из способов вычисления многочлена

$$v(x) = v_{n-1}x^{n-1} + v_{n-2}x^{n-2} + \ldots + v_1 x + v_0$$

в некоторой точке β. Правило Горнера, записанное в виде

$$v(\beta) = (\ldots((v_{n-1}\beta + v_{n-2})\beta + v_{n-3})\beta + \ldots + v_1)\beta + v_0,$$

требует $n-1$ сложений и $n-1$ умножений в поле элемента β. Если все различные степени элемента β вычислены заранее,

то правило Горнера не дает никаких преимуществ по сравнению с прямым вычислением. Преимущество правила Горнера состоит в том, что оно не требует предварительного вычисления и запоминания этих степеней.

Если $\beta = \omega^k$, то правило Горнера вычисляет k-ю компоненту преобразования Фурье посредством $n-1$ комплексных умножений и $n-1$ комплексных сложений. Более эффективным для этого вычисления является алгоритм Герцеля.

Алгоритм Герцеля представляет собой процедуру вычисления дискретного преобразования Фурье. Он позволяет уменьшить число необходимых умножений, но на очень малый множитель. Он не принадлежит к алгоритмам БПФ, так как его сложность по-прежнему пропорциональна n^2. Алгоритм Герцеля полезен в тех случаях, когда требуется вычислить малое число компонент преобразования Фурье, — как правило, не более чем $\log_2 n$ из n компонент. Так как БПФ-алгоритмы вычисляют все компоненты преобразования, то в этих случаях приходится выбрасывать ненужные компоненты.

Для вычисления одной компоненты преобразования Фурье $V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i$ рассмотрим многочлен $p(x) = (x - \omega^k)(x - \omega^{-k})$. Он представляет собой многочлен наименьшей степени с вещественными коэффициентами, для которого элемент ω^{-k} является корнем. Это минимальный многочлен элемента ω^k над полем вещественных чисел, и он равен

$$p(x) = x^2 - 2\cos\left(\frac{2\pi}{n}k\right)x + 1.$$

Пусть

$$v(x) = \sum_{i=0}^{n-1} v_i x^i$$

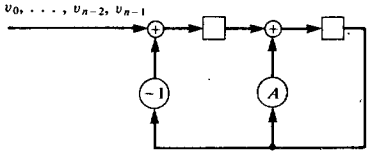
и запишем

$$v(x) = p(x)\,Q(x) + r(x).$$

Многочлен-частное $Q(x)$ и многочлен-остаток $r(x)$ могут быть найдены с помощью алгоритма деления многочленов. Тогда величина V_k может быть вычислена по остатку согласно равенству

$$V_k = v(\omega^k) = r(\omega^k),$$

так как по построению величина $p(\omega^k)$ равна нулю. Бо́льшая часть работы приходится на деление многочленов. Если коэффициенты многочлена $v(x)$ являются комплексными, то для его деления на многочлен $p(x)$ требуется $2(n-2)$ вещественных умножений; если же коэффициенты многочлена $v(x)$ являются вещественными, то требуется $n-2$ вещественных умножений. Аналогично, необходимое число сложений в комплексном случае равно $4(n-2)$, а в вещественном $2(n-2)$.



Замечания:
- Входные данные вещественные или комплексные.
- $A = 2\cos\frac{2\pi}{n}k$.
- $r(x)$ остается в регистре сдвига после завершения деления
- Для каждого V_k своя цепь деления.
- $V_k = \omega^k r_1 + r_0$.

Рис. 4.11. Блок-схема алгоритма Герцеля.

Так как степень многочлена $r(x)$ равна единице, то для вычисления $r(\omega^k)$ требуется только одно комплексное умножение и одно комплексное сложение. Таким образом, при комплексном входе для вычисления одной выходной компоненты алгоритму Герцеля требуется $2n-1$ вещественных умножений и $4n-1$ вещественных сложений.

Блок-схема алгоритма Герцеля показана на рис. 4.11. Эта схема имеет форму авторегрессионного фильтра. Это является следствием того, что схема деления многочленов имеет форму авторегрессионного фильтра. После ввода многочлена $v(x)$ цепь на рис. 4.11 будет содержать остаток $r(x)$ от деления многочлена $v(x)$ на многочлен $p(x)$. Частное $Q(x)$ интереса не представляет и поэтому отбрасывается.

4.5. Вычисление преобразования Фурье с помощью свертки

Одним из эффективных способов вычисления дискретного преобразования Фурье

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i$$

является сведение к вычислению свертки. Это может показаться странным, так как мы уже знаем, что хорошим методом вычисления свертки является использование БПФ и теоремы о свертке. И на самом деле, возможно поэтому, многие годы рассматриваемые в данном разделе методы привлекали мало внимания. Сейчас, однако, стало понятно, что иногда выгодно вычислять преобразование Фурье сведением к вычислению свертки, а иногда, наоборот, выгоднее вычислять свертку через преобразование Фурье. Что еще удивительнее, иногда выгодно вычислять свертку через преобразование Фурье, реализуя при этом преобразование Фурье через алгоритм вычисления свертки, хотя и на длине, отличной от исходной.

Два различных способа перехода от преобразования Фурье к свертке дают чирп-алгоритм Блюстейна и алгоритм Рейдера для простых чисел (см. рис. 4.12). Алгоритм Блюстейна переводит n-точечное преобразование Фурье в n-точечную свертку и $2n$ дополнительных умножений. Алгоритм Рейдера переводит n-точечное преобразование Фурье в $(n-1)$-точечную свертку, но только для простых чисел n.

Алгоритм Блюстейна менее полезен, но проще в описании, так что мы начнем с него. Он описывается равенством

$$V_k = \beta^{-k^2} \sum_{i=0}^{n-1} \beta^{(i-k)^2} (\beta^{-i^2} v_i),$$

где β равно квадратному корню из ω. В этом варианте преобразования Фурье производятся следующие вычисления:

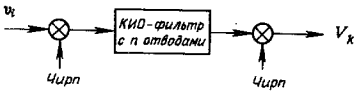
$$\beta^{-k^2} \sum_{i=1}^{n-1} \beta^{(i-k)^2} (\beta^{-i^2} v_i) = \sum_{i=0}^{n-1} \beta^{2ik} v_i = \sum_{i=0}^{n-1} \omega^{ik} v_i = V_k.$$

Чирп-алгоритм содержит n поточечных умножений v_i на β^{-i^2}, циклическую свертку с β^{i^2} в КИО-фильтре с n отводами и следующие за этим n поточечных умножений на β^{-k^2}. Поэтому полное число операций по-прежнему имеет порядок n^2, так что чирп-алгоритм асимптотически не эффективнее прямого вычисления преобразования Фурье. Однако в некоторых приложениях он допускает более простую аппаратурную реализацию. Кроме того,

Чирп-алгоритм Блюстейна

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i$$

$$= \beta^{k^2} \sum_{i=0}^{n-1} \beta^{-(i-k)^2} [\beta^{i^2} v_i]$$



Алгоритм Рейдера для простой длины

Длина n преобразования является простым числом

Использовать π — примитивный элемент поля GF(n)

$$\{1, 2, 3, \ldots, n-1\} = \{\pi^1, \pi^2, \pi^3, \ldots, \pi^{n-1} \mod n\}$$

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i \qquad i = \pi^{-[n-r(i)]} \qquad k = \pi^{r(k)}$$

$$V_0 = \sum_{i=0}^{n-1} v_i \qquad V_k = v_0 + \sum_{i=0}^{n-1} \omega^{ik} v_i \qquad k = 0, \ldots, n-1$$

$$= v_0 + \sum_{i=1}^{n-1} \omega^{\pi^{r(i)+r(k)}} v_i$$

$$V_l' = v_0 + \sum_{j=1}^{n-1} \omega^{\pi^{l-j}} v_j \qquad l = 0, \ldots, n-1$$

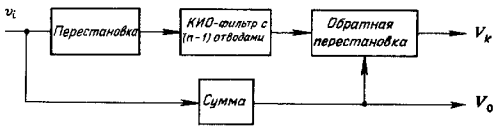


Рис. 4.12. Сведение преобразования Фурье к свертке.

прямое вычисление свертки можно заменить рассмотренными в гл. 3 алгоритмами быстрой свертки.

Алгоритм Блюстейна содержит $2n$ умножений и свертку длины n. Более предпочтителен следующий алгоритм — алгоритм Рейдера, — так как он не содержит $2n$ дополнительных умножений. Алгоритм Рейдера содержит только некоторые операции по переиндексации входных данных и циклическую свертку, длина которой теперь уже равна $n-1$.

Алгоритм Рейдера можно использовать для вычисления преобразования Фурье в любом поле F, если только длина преобразования n является простым числом. Так как n — простое число, то можно воспользоваться структурой поля $GF(n)$ для переиндексации компонент входного вектора. Поле $GF(n)$ не следует путать с полем F, в котором вычисляется преобразование Фурье.

Выберем примитивный элемент π простого поля $GF(n)$. Тогда каждое не превосходящее n целое число можно однозначно записать в виде степени элемента π. Преобразование Фурье

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \qquad k = 0, \ldots, n-1,$$

можно переписать, заменив индексы i и k соответствующими степенями элемента π. Индексы i и k принимают и нулевые значения; эти компоненты входного и выходного векторов удобнее рассматривать отдельно. Тогда

$$V_0 = \sum_{i=0}^{n-1} v_i,$$

$$V_k = v_0 + \sum_{i=1}^{n-1} \omega^{ik} v_i, \quad k = 1, \ldots, n-1.$$

Пусть $r(i)$ обозначает единственное для каждого i от 1 до $n-1$ целое число, такое что в поле $GF(n)$ выполняется равенство $\pi^{r(i)} = i$. Функция $r(i)$ является отображением множества $\{1, 2, \ldots, n-1\}$ на множество $\{1, 2, \ldots, n-1\}$; она задает перестановку на множестве $\{1, 2, \ldots, n-1\}$. Тогда V_k можно записать в следующем виде:

$$V_{\pi^{r(k)}} = v_0 + \sum_{i=1}^{n-1} \omega^{\pi^{r(i)+r(k)}} v_{\pi^{r(i)}}.$$

Так как $r(i)$ задает перестановку, то можно положить $l = r(k)$ и $j = n-1-r(i)$ и выбрать j в качестве индекса суммирования; это дает

$$V_{\pi^l} = v_0 + \sum_{j=1}^{n-1} \omega^{\pi^{l-j}} v_{\pi^{n-1-j}}$$

или

$$V'_l = v_0 + \sum_{j=0}^{n-2} \omega^{\pi^l - l} v'_j,$$

где $V'_l = V_{\pi^l}$ и $v'_j = v_{\pi^{n-1-j}}$ — соответственно переставленные компоненты последовательностей входных и выходных данных. Теперь мы получили уравнение для вычисления циклической свертки последовательностей $\{v'_j\}$ и $\{\omega^{\pi^l}\}$. Таким образом, переставляя индексы входных и выходных данных, мы записали преобразование Фурье в виде свертки. Однако для того вида, в котором эти формулы выписаны, необходимое число операций для вычисления свертки опять имеет порядок n^2. В следующем разделе мы скомбинируем алгоритм Рейдера для простой длины с алгоритмом Винограда для свертки и получим малый БПФ-алгоритм Винограда.

Построим, используя алгоритм Рейдера, двоичное пятиточечное преобразование Фурье, для которого

$$V_k = \sum_{i=0}^{4} \omega^{ik} v_i, \quad k = 0, \ \ldots, \ 4,$$

где $\omega = e^{-j2\pi/5}$. Сначала перепишем преобразование в виде

$$V_0 = \sum_{i=0}^{4} v_i,$$

$$V_k = v_0 + \sum_{i=1}^{4} \omega^{ik} v_i = V_0 + \sum_{i=1}^{4} (\omega^{ik} - 1)\, v_i, \quad k = 1, \ \ldots, \ 4,$$

и будем заниматься только членами $\sum_{i=1}^{4} (\omega^{ik} - 1)\, v_i$. Элемент 2 в поле $GF(5)$ является примитивным, и, следовательно, в этом поле выполняются равенства

$$2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 3,$$

$$2^0 = 1, \quad 2^{-1} = 3, \quad 2^{-2} = 4, \quad 2^{-3} = 2.$$

Таким образом,

$$V'_l - V_0 = \sum_{j=0}^{3} (\omega^{2^{l-j}} - 1)\, v'_j.$$

В этой сумме легко узнается 4-точечная циклическая свертка. Выделим постоянные члены и определим фильтр Рейдера, задавая его многочленом $g(x)$ над полем комплексных чисел, где

$$g(x) = (\omega^3 - 1)\, x^3 + (\omega^4 - 1)\, x^2 + (\omega^2 - 1)\, x + (\omega - 1)$$

(с коэффициентами $g_j = \omega^{2^j} - 1$). Вход и выход фильтра описываются многочленами, коэффициенты которых равны переставленным компонентам векторов $\mathbf{v}$ и $\mathbf{V}$. Итоговая форма 5-точечного алгоритма Рейдера задается уравнениями:

$$d(x) = v_2 x^3 + v_4 x^2 + v_3 x + v_1,$$

$$s(x) = (V_3 - V_0)\, x^3 + (V_4 - V_0)\, x^2 + (V_2 - V_0)\, x + (V_1 - V_0)$$

и

$$s(x) = g(x)\, d(x) \pmod{x^4 - 1}.$$

Многочлен $g(x)$ фиксирован. Многочлен $d(x)$ образуется перестановкой коэффициентов многочлена $v(x)$. Многочлен $V(x)$ получается обратной перестановкой коэффициентов многочлена $s(x)$. Схематически этот алгоритм представлен на рис. 4.13.

Поучительно переписать алгоритм Рейдера в матричной формулировке. Для 5-точечного преобразования Фурье такая формулировка имеет вид

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 \\ 1 & \omega^2 & \omega^4 & \omega & \omega^3 \\ 1 & \omega^3 & \omega & \omega^4 & \omega^2 \\ 1 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}.$$

Отсюда получаем $V_0 = v_0 + v_1 + v_2 + v_3 + v_4$ и

$$\begin{bmatrix} V_1 - V_0 \\ V_2 - V_0 \\ V_3 - V_0 \\ V_4 - V_0 \end{bmatrix} = \begin{bmatrix} \omega^1 - 1 & \omega^2 - 1 & \omega^3 - 1 & \omega^4 - 1 \\ \omega^2 - 1 & \omega^4 - 1 & \omega^1 - 1 & \omega^3 - 1 \\ \omega^3 - 1 & \omega^1 - 1 & \omega^4 - 1 & \omega^2 - 1 \\ \omega^4 - 1 & \omega^3 - 1 & \omega^2 - 1 & \omega^1 - 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}.$$

Правило перестановки алгоритма Рейдера приводит это равенство к эквивалентному равенству

$$\begin{bmatrix} V_1 - V_0 \\ V_2 - V_0 \\ V_4 - V_0 \\ V_3 - V_0 \end{bmatrix} = \begin{bmatrix} \omega^1 - 1 & \omega^3 - 1 & \omega^4 - 1 & \omega^2 - 1 \\ \omega^2 - 1 & \omega^1 - 1 & \omega^3 - 1 & \omega^4 - 1 \\ \omega^4 - 1 & \omega^2 - 1 & \omega^1 - 1 & \omega^3 - 1 \\ \omega^3 - 1 & \omega^4 - 1 & \omega^2 - 1 & \omega^1 - 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \\ v_4 \\ v_2 \end{bmatrix},$$

в котором легко распознать матричную форму записи циклической свертки

$$\begin{bmatrix} V_1 - V_0 \\ V_2 - V_0 \\ V_4 - V_0 \\ V_3 - V_0 \end{bmatrix} = \begin{bmatrix} g_0 & g_3 & g_2 & g_1 \\ g_1 & g_0 & g_3 & g_2 \\ g_2 & g_1 & g_0 & g_3 \\ g_3 & g_2 & g_1 & g_0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \\ v_4 \\ v_2 \end{bmatrix}.$$

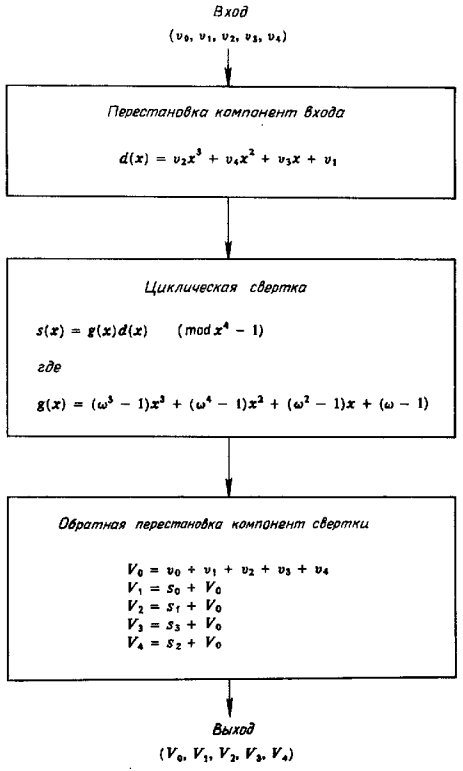


Рис. 4.13. Алгоритм Рейдера вычисления 5-точечного преобразования Фурье.

где $g_0 = \omega - 1$, $g_1 = \omega^2 - 1$, $g_2 = \omega^4 - 1$, $g_3 = \omega^3 - 1$.

Идея алгоритма Рейдера переносится на случай, когда длина n преобразования равна степени нечетного простого числа. В этом случае, подобно тому, как нулевые компоненты временного и частотного векторов обрабатывались отдельно, еще некоторое количество компонент временного и частотного векторов должны обрабатываться отдельно. Это объясняется тем, что в кольце $Z/(p^m)$ не существует элемента порядка $p^m - 1$. Теорема 5.1.8 (которая будет доказана в гл. 5) гарантирует, однако, что для простого нечетного p в этом кольце имеется элемент порядка $p^{m-1}(p-1)$, и мы этим воспользуемся.

При вычислении

$$V_k = \sum_{l=0}^{p^m-1} \omega^{lk} v_l, \qquad k = 0, \ldots, p^m - 1,$$

воспользуемся структурой кольца $Z/(p^m)$ для того, чтобы переупорядочить компоненты векторов. Эта структура описывается теоремой 5.1.8. Если q равно степени простого нечетного числа, то $Z/(p^m)$ содержит циклическую подгруппу порядка $p^{m-1}(p-1)$. Из $(p^m \times p^m)$-матрицы $\mathbf{W} = [\omega^{lk}]$ просто выбросим все вызывающие трудности строки и столбцы так, чтобы к оставшейся матрице размерности $p^{m-1}(p-1)$ была применима идея Рейдера. Выброшенные строки и столбцы будем обрабатывать отдельно. Как мы увидим в следующем разделе, обработку даже этих вызывающих трудности столбцов и строк можно организовать как вычисление еще меньших циклических сверток.

Таким образом, вычисление p^m-точечного преобразования Фурье, хотя и слишком нерегулярного для того, чтобы быть вычисляемым как единое целое, можно реализовать в виде всего нескольких разумных фрагментов.

Случай, когда длина преобразования равна степени двойки, является несколько более сложным и требует еще одного уровня вычислений.

Это объясняется тем, что множество индексов, взаимно простых с 2^m — множество нечетных индексов — не образует циклической группы по умножению. Как доказывается в теореме 5.1.8, это множество образует группу, изоморфную группе $Z_2 \times Z_{2^{m-2}}$. Идея организации процедуры вычислений состоит в следующем. Из $(2^m \times 2^m)$-матрицы $\mathbf{W} = [\omega^{lk}]$ выбрасываются все строки и столбцы с четными индексами, так что остается матрица размерности 2^{m-1}. Все оставшиеся индексы являются нечетными и по умножению образуют группу, изоморфную группе $Z_2 \times Z_{2^{m-2}}$. Этот изоморфизм используется для того, чтобы определить такую

перестановку строк и столбцов матрицы, которая переводит ее в матрицу вида

$$\overline{\mathbf{W}} = \left[\begin{array}{c|c} \mathbf{W}_1 & \mathbf{W}_2 \\ \hline \mathbf{W}_2 & \mathbf{W}_1 \end{array}\right],$$

где $\mathbf{W}_1$ и $\mathbf{W}_2$ представляют собой $(2^{m-2}\times 2^{m-2})$-матрицы, каждая из которых задает структуру циклической свертки.

Точнее, пусть $\pi = 3$ и $\sigma = 2^m - 1$. Тогда по модулю 2^m имеем $\sigma^2 = 1$. На самом деле группа нечетных целых чисел относительно умножения по модулю 2^m порождается элементами π и σ: каждый элемент группы может быть однозначно представлен в виде $\sigma^{l'}\pi^{l''}$, где $l' = 0,\ 1$ и $l'' = 0, \ldots, 2^{m-2}$. Следовательно, $\omega^{lk} = \omega^{\sigma^{l'}\pi^{l''}\sigma^{r'}\pi^{r''}}$, и подходящей перестановкой строк и столбцов матрица $\overline{\mathbf{W}}$ приводится к виду

$$\overline{\mathbf{W}} = \left[\begin{array}{cc} [\omega^{\pi^{l''}+r''}] & [\omega^{\sigma\pi^{l''}+r''}] \\ [\omega^{\sigma\pi^{l''}+r''}] & [\omega^{\pi^{l''}+r''}] \end{array}\right],$$

где l'' и r'' соответственно индексы строк и столбцов в каждой из подматриц. Каждая из четырех подматриц задает циклическую свертку длины 2^{m-2}.

После выполнения перестановки рассматриваемое вычисление приводится к виду матричной 2-точечной циклической свертки

$$\begin{bmatrix} \bar{\mathbf{v}}_1 \\ \bar{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 \\ \mathbf{W}_2 & \mathbf{W}_1 \end{bmatrix} \begin{bmatrix} \bar{\mathbf{v}}_1 \\ \bar{\mathbf{v}}_2 \end{bmatrix},$$

один из способов вычисления которой дается формулой

$$\begin{bmatrix} \bar{\mathbf{v}}_1 \\ \bar{\mathbf{v}}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{2}(\mathbf{W}_1 + \mathbf{W}_2) & 0 \\ 0 & \frac{1}{2}(\mathbf{W}_1 - \mathbf{W}_2) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \bar{\mathbf{v}}_1 \\ \bar{\mathbf{v}}_2 \end{bmatrix}.$$

Этим задача сводится к вычислению двух комплексных циклических сверток длины 2^{m-2}; мы увидим, что вычисления можно организовать несколько лучше.

В качестве примера рассмотрим две 4-точечные циклические свертки, образующие сердцевину 16-точечного преобразования Фурье. Пусть

$$V_k = \sum_{l=0}^{15} \omega^{lk} v_l, \quad k = 0, \ldots, 15,$$

где $\omega^{16} = 1$. Рассмотрим матрицу, образуемую в результате выбрасывания всех четных индексов в рассматриваемом диапазоне значений l и k. Получим

$$\begin{bmatrix} V'_1 \\ V'_3 \\ V'_5 \\ V'_7 \\ V'_9 \\ V'_{11} \\ V'_{13} \\ V'_{15} \end{bmatrix} = \begin{bmatrix} \omega^1 & \omega^3 & \omega^5 & \omega^7 & \omega^9 & \omega^{11} & \omega^{13} & \omega^{15} \\ \omega^3 & \omega^9 & \omega^{15} & \omega^5 & \omega^{11} & \omega^1 & \omega^7 & \omega^{13} \\ \omega^5 & \omega^{15} & \omega^9 & \omega^3 & \omega^{13} & \omega^7 & \omega^1 & \omega^{11} \\ \omega^7 & \omega^5 & \omega^3 & \omega^1 & \omega^{15} & \omega^{13} & \omega^{11} & \omega^9 \\ \omega^9 & \omega^{11} & \omega^{13} & \omega^{15} & \omega^1 & \omega^3 & \omega^5 & \omega^7 \\ \omega^{11} & \omega^1 & \omega^7 & \omega^{13} & \omega^3 & \omega^9 & \omega^{15} & \omega^5 \\ \omega^{13} & \omega^7 & \omega^1 & \omega^{11} & \omega^5 & \omega^{15} & \omega^9 & \omega^3 \\ \omega^{15} & \omega^{13} & \omega^{11} & \omega^9 & \omega^7 & \omega^5 & \omega^3 & \omega^1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \\ v_5 \\ v_7 \\ v_9 \\ v_{11} \\ v_{13} \\ v_{15} \end{bmatrix}.$$

Чтобы найти нужную перестановку, выпишем индексы в виде чисел $15^{l'}\cdot 3^{l''}$ для $l' = 0,\ 1$ и $l'' = 0,\ 1,\ 2,\ 3$. Степени тройки по модулю 16 имеют вид

$$3^0 = 1, \quad 3^1 = 3, \quad 3^2 = 9, \quad 3^3 = 11,$$

$$3^0 = 1, \quad 3^{-1} = 11, \quad 3^{-2} = 9, \quad 3^{-3} = 3,$$

и соответственно

$$15\cdot 3^0 = 15, \quad 15\cdot 3^1 = 13, \quad 15\cdot 3^2 = 7, \quad 15\cdot 3^3 = 5,$$

$$15\cdot 3^0 = 15, \quad 15\cdot 3^{-1} = 5, \quad 15\cdot 3^{-2} = 7, \quad 15\cdot 3^{-3} = 13.$$

Выполним перестановку входных индексов, записывая их в порядке $15^{-l'}\cdot 3^{-l''} \pmod{16}$, и перестановку выходных индексов, записывая их в порядке $15^{l'}\cdot 3^{l''} \pmod{16}$. Тогда

$$\begin{bmatrix} V'_1 \\ V'_{11} \\ V'_9 \\ V'_3 \\ V'_{15} \\ V'_5 \\ V'_7 \\ V'_{13} \end{bmatrix} = \left[\begin{array}{cccc|cccc} \omega^1 & \omega^3 & \omega^9 & \omega^{11} & \omega^{15} & \omega^{13} & \omega^7 & \omega^5 \\ \omega^{13} & \omega^1 & \omega^3 & \omega^9 & \omega^5 & \omega^{15} & \omega^{13} & \omega^7 \\ \omega^9 & \omega^{11} & \omega^1 & \omega^3 & \omega^7 & \omega^5 & \omega^{15} & \omega^{13} \\ \omega^3 & \omega^9 & \omega^{11} & \omega^1 & \omega^{13} & \omega^7 & \omega^5 & \omega^{15} \\ \hline \omega^{15} & \omega^{13} & \omega^7 & \omega^5 & \omega^1 & \omega^3 & \omega^9 & \omega^{11} \\ \omega^5 & \omega^{15} & \omega^{13} & \omega^7 & \omega^{11} & \omega^1 & \omega^3 & \omega^9 \\ \omega^7 & \omega^5 & \omega^{15} & \omega^{13} & \omega^9 & \omega^{11} & \omega^1 & \omega^3 \\ \omega^{13} & \omega^7 & \omega^5 & \omega^{15} & \omega^3 & \omega^9 & \omega^{11} & \omega^1 \end{array}\right] \begin{bmatrix} v_1 \\ v_3 \\ v_9 \\ v_{11} \\ v_{15} \\ v_{13} \\ v_7 \\ v_5 \end{bmatrix}.$$

Чтобы показать четыре образовавшиеся при этом циклические свертки, матрица разбита на блоки. Если преобразование Фурье вычисляется в поле комплексных чисел, то блоки связаны соот-

ношением комплексной сопряженности. Чтобы наглядно выявить эту связь, перепишем матричное уравнение в виде

$$\begin{bmatrix} V'_1 \\ V'_{11} \\ V'_9 \\ V'_3 \\ \\ V'_{15} \\ V'_5 \\ V'_7 \\ V'_{13} \end{bmatrix} = \begin{bmatrix} \omega^1 & \omega^3 & \omega^9 & \omega^{11} & \omega^{-1} & \omega^{-3} & \omega^{-9} & \omega^{-11} \\ \omega^{11} & \omega^1 & \omega^3 & \omega^9 & \omega^{-11} & \omega^{-1} & \omega^{-3} & \omega^{-9} \\ \omega^9 & \omega^{11} & \omega^1 & \omega^3 & \omega^{-9} & \omega^{-11} & \omega^{-1} & \omega^{-3} \\ \omega^3 & \omega^9 & \omega^{11} & \omega^1 & \omega^{-3} & \omega^{-9} & \omega^{-11} & \omega^{-1} \\ \omega^{-1} & \omega^{-3} & \omega^{-9} & \omega^{-11} & \omega^1 & \omega^3 & \omega^9 & \omega^{11} \\ \omega^{-11} & \omega^{-1} & \omega^{-3} & \omega^{-9} & \omega^{11} & \omega^1 & \omega^3 & \omega^9 \\ \omega^{-9} & \omega^{-11} & \omega^{-1} & \omega^{-3} & \omega^9 & \omega^{11} & \omega^1 & \omega^3 \\ \omega^{-3} & \omega^{-9} & \omega^{-11} & \omega^{-1} & \omega^3 & \omega^9 & \omega^{11} & \omega^1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \\ v_9 \\ v_{11} \\ v_{15} \\ v_{13} \\ v_7 \\ v_5 \end{bmatrix}$$

Отметим, что четыре верхние строки комплексно сопряжены с четырьмя нижними строками; выполнять надо только вычисления, связанные с первыми четырьмя строками. Следовательно, дальнейшие вычисления можно организовать в виде пары циклических сверток

$$V'_3x^3 + V'_9x^2 + V'_{11}x + V'_1 =$$
$$= (\omega^{11}x^3 + \omega^6x^2 + \omega^3x + \omega)(v_{11}x^3 + v_9x^2 + v_3x + v_1) +$$
$$+ (\omega^{-11}x^3 + \omega^{-9}x^2 + \omega^{-3}x + \omega^{-1})(v_5x^3 + v_7x^2 + v_{13}x + v_{15})$$
$$(\mathrm{mod}\ x^4 - 1).$$

Для комплексного поля можно использовать и альтернативный способ, основанный на выписанной выше 2-точечной циклической свертке блоков:

$$\begin{bmatrix} \bar{\mathbf{V}}_1 \\ \bar{\mathbf{V}}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{2}(\mathbf{W}_1 + \mathbf{W}_2) & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}(\mathbf{W}_1 - \mathbf{W}_2) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \bar{\mathbf{v}}_1 \\ \bar{\mathbf{v}}_2 \end{bmatrix},$$

где при $\theta = 2\pi/16$

$$\frac{1}{2}(\mathbf{W}_1 + \mathbf{W}_2) = \begin{bmatrix} \cos\theta & \cos 3\theta & \cos 9\theta & \cos 11\theta \\ \cos 11\theta & \cos\theta & \cos 3\theta & \cos 9\theta \\ \cos 9\theta & \cos 11\theta & \cos\theta & \cos 3\theta \\ \cos 3\theta & \cos 9\theta & \cos 11\theta & \cos\theta \end{bmatrix}$$

и

$$\frac{1}{2}(\mathbf{W}_1 - \mathbf{W}_2) = \begin{bmatrix} j\sin\theta & j\sin 3\theta & j\sin 9\theta & j\sin 11\theta \\ j\sin 11\theta & j\sin\theta & j\sin 3\theta & j\sin 9\theta \\ j\sin 9\theta & j\sin 11\theta & j\sin\theta & j\sin 3\theta \\ j\sin 3\theta & j\sin 9\theta & j\sin 11\theta & j\sin\theta \end{bmatrix}.$$

Заметим, что теперь мы получили две циклические свертки, одна из которых чисто вещественная, а другая — чисто мнимая. Для случая вещественного входного вектора необходимо вычислить только две вещественные свертки,

$$s(x) = [\cos 3\theta x^3 + \cos 9\theta x^2 + \cos 11\theta x + \cos\theta]\, d(x)$$
$$(\mathrm{mod}\ x^4 - 1)$$

и

$$s'(x) = [\sin 3\theta x^3 + \sin 9\theta x^2 + \sin 11\theta x + \sin\theta]\, d'(x)$$
$$(\mathrm{mod}\ x^4 - 1).$$

Используя китайскую теорему об остатках для разложения $x^4 - 1 = (x^2 - 1)(x^2 + 1)$, видим, что некоторые вычисления излишни, так как

$$\cos 3\theta x^3 + \cos 9\theta x^2 + \cos 11\theta x + \cos\theta = 0 \pmod{x^2 - 1}$$

и

$$\sin 3\theta x^3 + \sin 9\theta x^2 + \sin 11\theta x + \sin\theta = 0 \pmod{x^2 - 1}.$$

Следовательно, умножения, связанные с модулем $x^2 - 1$, не нужны. Вычеты по модулю $x^2 + 1$ требуют трех умножений каждый. Следовательно, для вычисления двух циклических сверток необходимо в общем только шесть умножений. Исходное 16-точечное преобразование требует, таким образом, всего 10 нетривиальных умножений.

4.6. Алгоритм Винограда для быстрого преобразования Фурье малой длины

Этот БПФ-алгоритм Винограда предназначен для эффективного вычисления дискретного преобразования Фурье малой длины. В основу алгоритма заложены две идеи: рассмотренный в предыдущем разделе алгоритм Рейдера для простых длин и рассмотренный в разд. 3.4 алгоритм Винограда для свертки. Рассмотрению подлежат три случая: (1) длина равна простому числу; (2) длина равна степени простого нечетного числа и (3) длина равна степени двойки. Наиболее популярными длинами являются 2, 3, 4, 5, 7, 8, 9 и 16. Характеристики БПФ-алгоритма Винограда для этих длин приведены на рис. 4.14. (Сами алгоритмы выписаны в приложении В.) Число умножений на этом рисунке упоминается дважды: один раз приведено число умножений без учета тривиальных умножений, а другой раз — полное число умножений, включающее умножения на (± 1) и $(\pm j)$. Если БПФ-алгоритм малой длины используется в качестве блока для гнездового алгоритма, который будет изложен в гл. 8, то умножения на (± 1) и $(\pm j)$ в малом алгоритме приводят к нетривиальным умно-

Длина	Число вещественных умножений *	Число нетривиальных вещественных сложений	
2	2	0	2
3	3	2	6
4	4	0	8
5	6	5	17
7	9	8	36
8	8	2	26
9	11	10	44
11	21	20	84
13	21	20	94
16	18	10	74
17	36	35	157
19	39	38	186

* Включая умножения на ± 1 или $\pm j$.

Рис. 4.14. Характеристики БПФ-алгоритмов Винограда малой длины.

жениям (и сложениям) в большем алгоритме. Поэтому мы выписываем как полное число умножений, так и число умножений без учета тривиальных.

В характеристики алгоритмов, приведенные на рис. 4.14, включены также некоторые тривиальные сложения. Сюда относятся чисто вещественные и чисто мнимые сложения, которые согласно принятым определениям не являются сложениями. Но при подсчетах сложности мы не будем различать тривиальные и нетривиальные сложения. Если заменить вещественный вектор комплексным, то все сложения становятся нетривиальными комплексными сложениями.

Длина преобразования равна простому числу. Первый шаг состоит в замене преобразования Фурье сверткой. Если n мало, то воспользуемся алгоритмом Рейдера для замены преобразования Фурье сверткой, которую вычислим затем, используя алгоритм Винограда для сверток малой длины. Длина n выбирается не слишком большой, так как необходимые уравнения выписываются вручную. Переход от преобразования Фурье к свертке с помощью алгоритма Рейдера осуществляется только перестановкой индексов; этот шаг не содержит ни умножений, ни сложений. Структура алгоритма свертки такова, что сначала выполняется некоторое множество сложений, затем некоторое множество умножений, а затем опять некоторое множество сложений.

Рассмотрим 5-точечный двоичный БПФ-алгоритм Винограда, вычисляющий

$$V_k = \sum_{i=0}^{4} \omega^{ik} v_i, \quad k = 0, \ldots, 4,$$

где $\omega = e^{-j2\pi/5}$. Сначала воспользуемся описанным в разд. 4.5 алгоритмом Рейдера, переводящим рассматриваемое преобразование Фурье в циклическую свертку

$$s(x) = g(x)\, d(x) \pmod{x^4 - 1},$$

где многочлен Рейдера

$$g(x) = (\omega^3 - 1)\, x^3 + (\omega^4 - 1)\, x^2 + (\omega^2 - 1)\, x + (\omega - 1)$$

имеет фиксированные коэффициенты. Вход и выход фильтра описываются соответственно многочленами

$$d(x) = v_2 x^3 + v_4 x^2 + v_3 x + v_1,$$
$$s(x) = (V_3 - V_0)\, x^3 + (V_4 - V_0)\, x^2 + (V_2 - V_0)\, x + (V_1 - V_0).$$

Коэффициенты многочлена $d(x)$ получаются перестановкой коэффициентов входного многочлена $v(x)$; коэффициенты многочлена $V(x)$ на выходе алгоритма (с точностью до слагаемого V_0) получаются обратной перестановкой коэффициентов многочлена $s(x)$ на выходе фильтра.

5-точечный БПФ-алгоритм Винограда получается, если произведение $g(x)\, d(x)$ вычислять с помощью алгоритма Винограда для малой свертки. Воспользуемся приведенным на рис. 3.14 и содержащим пять умножений алгоритмом 4-точечной циклической свертки. Его можно приспособить для вычисления преобразования Фурье, введя в матрицу свертки операции перестановок путем соответствующей перестановки ее строк и столбцов. Так как коэффициенты многочлена $g(x)$ фиксированы, то вычисление произведения вектора g на его матрицу также можно выполнить заранее. Если в алгоритме 4-точечной свертки произвести все эти изменения и внести в него члены v_0 и V_0, то и получится 5-точечный БПФ-алгоритм Винограда, стандартная матричная форма которого показана на рис. 4.15. Эта стандартная форма окажется очень полезной при построении в гл. 8 гнездовых методов. Заметим, что в алгоритме на рис. 4.15 матрицы предсложений и постсложений не являются квадратными. 5-точечный входной вектор дополняется до 6-точечного вектора, к компонентам которого применяется умножение. Верхние две строки связующей матрицы не содержат умножений, так как связывают v_0 и V_0. Другие пять строк соответетвуют алгоритму вычисления 4-точечной циклической свертки. Одна из констант умножения оказалась равной единице, так что на самом деле алгоритм содержит только пять нетривиальных умножений и одно тривиальное.

На рис. 4.15 видна также и другая важная деталь алгоритма. Хотя нет никаких причин ожидать этого, все диагональные элементы оказались чисто вещественными или чисто мнимыми [1]).

[1]) При желании множитель j можно перенести из диагональной матрицы в матрицу постсложений. Тогда элементы диагональной матрицы окажутся чисто вещественными.

Доказательство. Так как элемент π примитивен, то $\pi^{p-1} = 1$ и $\pi^{(p-1)/2} = -1$. Так как

$$g(x) = \sum_{k=0}^{p-2} (\omega^{\pi^k} - 1) x^k,$$

то коэффициенты многочлена $g(x) \pmod{x^{(p-1)/2} \pm 1}$ даются равенствами

$$g_k = (\omega^{\pi^k} - 1) \mp (\omega^{\pi^{k+(p-1)/2}} - 1).$$

Утверждение теоремы вытекает теперь из того, что $\pi^{(p-1)/2} = -1$. □

Мы показали, как строится БПФ-алгоритм Винограда малой длины для случая, когда длина преобразования n является простым числом. БПФ-алгоритм Винограда малой длины может быть построен и в случае, когда длина n преобразования равна степени простого числа. Эта конструкция также основана на аналогичной алгоритму Рейгера идее перехода от преобразования Фурье длины p^m, p простое, к свертке. Однако множество целых чисел по модулю p^m не образует поля, так что в этом случае отсутствует элемент π порядка $p^m - 1$.

Случаи $p = 2$ и нечетного простого p решаются двумя различными методами. Сначала рассмотрим случай нечетного простого p.

Длина преобразования равна степени простого нечетного числа. Конструкция несколько усложняется. Сначала для выделения циклической группы порядка $p^{r-1}(p-1)$ из множества $\{1, 2, \ldots, p^r - 1\}$ всех индексов удаляются целые числа, кратные p. Эта циклическая группа приводит к свертке длины $p^{r-1}(p-1)$, которая образует ядро алгоритма вычисления преобразования Фурье. Как и прежде, она вычисляется быстрым алгоритмом свертки. Компоненты свертки нужно переставить по правилу, обратному перестановке входных компонент свертки, и подправить членами, учитывающими выброшенные p^{r-1} строк и столбцов. Как мы увидим, эти поправочные члены можно вычислить алгоритмами еще меньших сверток, так как они представимы в виде алгоритмов преобразования Фурье малой длины.

Например, при $N = 9 = 3^2$ удаление номеров 0, 3 и 6 из множества всех индексов приводит к подмножеству $\{1, 2, 4, 5, 7, 8\}$, образующему относительно умножения по модулю 9 циклическую группу, изоморфную аддитивной группе Z_8 целых чисел $\{0, 1, 2, 3, 4, 5\}$ относительно сложения по модулю 6. Мультипликативная группа порождается степенями 2 по модулю 9, так как эти степени равны соответственно $\{1, 2, 4, 8, 7, 5\}$. Таким образом, 9-точечное преобразование Фурье содержит шесть строк, которые можно изолировать, переставить и вычислить в виде свертки. Нетрудно предвидеть, что остальные строки и столбцы (с индексами, равными 0, 3 и 6) имеют структуру, близкую к 3-точечному

$$V_k = \sum_{i=0}^{4} \omega^{ik} v_i \qquad \begin{matrix} i = 0, \ldots, 4 \\ \omega = e^{-j2\pi/5} \end{matrix}$$

$$\mathbf{V} = \mathbf{W}\mathbf{v} = \mathbf{CBA}\mathbf{v}$$

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & -1 \\ 1 & 1 & -1 & 1 & 1 & 0 \\ 1 & 1 & -1 & -1 & -1 & 0 \\ 1 & 1 & 1 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_0 & & & & & \\ & B_1 & & & & \\ & & B_2 & & & \\ & & & B_3 & & \\ & & & & B_4 & \\ & & & & & B_5 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & -1 & 1 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix},$$

где $B_0 = 1$

$$\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 2 & 0 & -2 & 0 \\ -2 & 2 & 2 & -2 \\ 2 & 2 & -2 & -2 \end{bmatrix} \begin{bmatrix} \omega - 1 \\ \omega^2 - 1 \\ \omega^4 - 1 \\ \omega^3 - 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(\cos\theta + \cos 2\theta) - 1 \\ \frac{1}{2}(\cos\theta - \cos 2\theta) \\ j \sin\theta \\ j(-\sin\theta + \sin 2\theta) \\ j(\sin\theta + \sin 2\theta) \end{bmatrix}$$

и $\theta = 2\pi/5$

Рис. 4.15. 5-точечный БПФ-алгоритм Винограда.

Это существенно потому, что означает, что в случае вещественного входного вектора каждому умножению отвечает одно вещественное умножение, а в случае комплексного входного вектора каждому умножению отвечают два вещественных умножения. Это явление оказывается довольно общим.

Если p нечетно, то можно записать

$$x^{p-1} - 1 = (x^{(p-1)/2} - 1)(x^{(p-1)/2} + 1).$$

Делители многочлена $x^{p-1} - 1$ делят один из множителей, стоящих справа в этом равенстве. Следовательно, когда алгоритм Винограда для вычисления свертки разбивается на подзадачи, соответствующие делителям многочлена $x^{p-1} - 1$, то ситуация описывается следующей теоремой.

Теорема 4.6.1. *Пусть $g(x)$ — многочлен Рейдера; тогда для каждого нечетного p коэффициенты многочлена $g(x) \pmod{x^{(p-1)/2} - 1}$ являются вещественными, а коэффициенты многочлена $g(x) \pmod{x^{(p-1)/2} + 1}$ являются мнимыми.*

преобразованию Фурье, так что некоторые из поправочных членов могут быть выражены в виде свертки малой длины.

Построение 9-точечного БПФ-алгоритма Винограда проводится следующим образом. Выпишем матричное уравнение

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 & \omega^8 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \omega & \omega^3 & \omega^5 & \omega^7 \\ 1 & \omega^3 & \omega^6 & 1 & \omega^3 & \omega^6 & 1 & \omega^3 & \omega^6 \\ 1 & \omega^4 & \omega^8 & \omega^3 & \omega^7 & \omega^2 & \omega^6 & \omega & \omega^5 \\ 1 & \omega^5 & \omega & \omega^6 & \omega^2 & \omega^7 & \omega^3 & \omega^8 & \omega^4 \\ 1 & \omega^6 & \omega^3 & 1 & \omega^6 & \omega^3 & 1 & \omega^6 & \omega^3 \\ 1 & \omega^7 & \omega^5 & \omega^3 & \omega & \omega^8 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^8 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{bmatrix}.$$

Посмотрев на эту матрицу, можно убедиться, что строки и столбцы с номерами 0, 3 и 6 являются для нас новыми, так как содержат повторяющиеся элементы. Переставим строки и столбцы матрицы так, чтобы эти строки и столбцы оказались в новой матрице первыми, остальные строки расположились в порядке степеней двойки по модулю 9, т. е. в порядке 1, 2, 4, 8, 7, 5, а столбцы — в порядке степеней 2^{-1} по модулю 9, т. е. в порядке 1, 5, 7, 8, 4, 2. Тогда вычисление преобразуется к виду

$$\begin{bmatrix} V_0 \\ V_1 \\ V_6 \\ V_4 \\ V_2 \\ V_5 \\ V_8 \\ V_7 \\ V_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & \omega^3 & \omega^6 & \omega^3 & \omega^6 & \omega^3 & \omega^6 \\ 1 & 1 & 1 & \omega^6 & \omega^3 & \omega^6 & \omega^3 & \omega^6 & \omega^3 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^5 & \omega^7 & \omega^8 & \omega^4 & \omega^2 \\ 1 & \omega^6 & \omega^3 & \omega^2 & \omega & \omega^5 & \omega^7 & \omega^8 & \omega^4 \\ 1 & \omega^3 & \omega^6 & \omega^4 & \omega^2 & \omega & \omega^5 & \omega^7 & \omega^8 \\ 1 & \omega^6 & \omega^3 & \omega^8 & \omega^4 & \omega^2 & \omega & \omega^5 & \omega^7 \\ 1 & \omega^3 & \omega^6 & \omega^7 & \omega^8 & \omega^4 & \omega^2 & \omega & \omega^5 \\ 1 & \omega^6 & \omega^3 & \omega^5 & \omega^7 & \omega^8 & \omega^4 & \omega^2 & \omega \end{bmatrix} \begin{bmatrix} v_0 \\ v_3 \\ v_6 \\ v_1 \\ v_5 \\ v_7 \\ v_8 \\ v_4 \\ v_2 \end{bmatrix}.$$

Пунктирные разбиения, сделанные в матрице, показывают сформировавшиеся циклические свертки: одну 6-точечную циклическую свертку и шесть 2-точечных циклических сверток. Те 2-точечные свертки, которые расположены во второй и третьей строках матрицы, содержат повторяющиеся вычисления, что позволяет переписать входящие в эти две строки вычисления в виде

$$\begin{bmatrix} V_3 \\ V_6 \end{bmatrix} = \begin{bmatrix} \omega^3 & \omega^6 \\ \omega^6 & \omega^3 \end{bmatrix} \begin{bmatrix} v_1 + v_7 + v_4 \\ v_5 + v_8 + v_2 \end{bmatrix} + \begin{bmatrix} v_0 + v_3 + v_6 \\ v_0 + v_3 + v_6 \end{bmatrix}.$$

Таким образом, подходящим способом выполненные перестановки и разбиения входных данных и матрицы преобразования позволили представить рассматриваемое преобразование Фурье в виде одной 6-точечной циклической свертки и двух 2-точечных циклических сверток. Можно было бы ожидать, что так организованное вычисление потребует 12 комплексных умножений. Однако, как показывает следующая теорема, происходят две вещи:

1. Все умножения являются чисто вещественными или чисто мнимыми, так что 12 комплексных умножений превращаются в 12 вещественных умножений.

2. Число необходимых умножений уменьшается до 10, так как два коэффициента оказываются равными нулю.

На самом деле мы выпишем алгоритм с 11 умножениями. Одно умножение на единицу необходимо добавить для того, чтобы строку с номером нуль записать в алгоритме в таком же виде, как и остальные строки.

Теорема 4.6.2. *Пусть $m > 1$ целое, а p — простое нечетное число. Пусть $b = (p-1)\,p^{m-1}$. Пусть $g(x)$ — обобщенный многочлен Рейдера, $g(x) = \sum_{k=0}^{b-1} \omega^{\pi^k} x^k$, где ω—корень из единицы степени p^m и π — целое число порядка b относительно умножения по модулю p^m. Тогда:*

(i) *Коэффициенты многочлена $g(x) \pmod{x^{b/2} - 1}$ являются вещественными числами;*

(ii) *Коэффициенты многочлена $g(x) \pmod{x^{b/2} + 1}$ являются мнимыми числами.*

(iii) *Коэффициенты многочлена $g(x) \pmod{x^{b/p} - 1}$ равны нулю.*

Доказательство. Доказательство первых двух утверждений теоремы аналогично доказательству теоремы 4.6.1. Так как порядок числа π равен b, а b четно, то $\pi^b = 1$ и $\pi^{b/2} = -1$. Тогда коэффициенты многочлена $g(x) \pmod{x^{b/2} \pm 1}$ равны $g_k = \omega^{\pi^k} \mp \omega^{\pi^{(k+b/2)}}$. Так как $\pi^{b/2} = -1$, то отсюда сразу вытекают первые два утверждения теоремы.

Для доказательства утверждения (iii) рассмотрим многочлен

$$g'(x) = g(x) \pmod{x^{b/p} - 1} = \sum_{k=0}^{b-1} \omega^{\pi^k} x^k \pmod{x^{b/p} - 1}.$$

Коэффициент g'_0 равен сумме тех коэффициентов многочлена $g(x)$, индекс k которых кратен числу b/p. Таких членов имеется p штук и

$$g'_0 = \sum_{t=0}^{p-1} \omega^{\pi^{tb/p}}.$$

В общем случае коэффициент g'_r дается равенством

$$g'_r = \sum_{i=0}^{p-1} \omega^{\pi^r + ib/p}, \qquad r = 0, \ldots, (b/p) - 1.$$

Требуется доказать, что g'_r равен нулю для всех таких r. Перепишем выражение для g'_r в виде

$$g'_r = \sum_{i=0}^{p-1} [\omega^{\pi^r}]^{ib/p}.$$

Так как ω^{π^r} опять является корнем степени p^m из единицы, то утверждение достаточно доказать только для r, равного нулю, т. е. для $g'_0 = \sum_{i=0}^{p-1} \omega^{\alpha i}$, где $\alpha = \pi^{b/p}$ представляет собой элемент порядка p. Последнее равенство можно переписать в виде

$$g'_0 = \omega' \sum_{h=0}^{p-1} (\omega')^h,$$

где ω' является корнем степени p из единицы. Следовательно, $g'_0 = 0$, и доказательство теоремы закончено. □

Таким образом, мы умеем строить БПФ-алгоритм Винограда малых длин, равных степени нечетного простого числа. Матрица преобразования Фурье размера $p^m \times p^m$ разбивается на $(p^m - p^{m-1})$-точечную циклическую свертку и $p^{m-1} + 1$ $(p-1)$-точечных циклических сверток. Согласно теореме 4.6.2, все эти свертки вычисляются с помощью алгоритма Винограда для циклических сверток, содержащего только чисто вещественные и чисто мнимые умножения.

Длина преобразования равна степени двойки. Преобразование Фурье, длина которого равна степени двойки, приходится рассматривать отдельно, так как целые числа, не превосходящие 2^m, взаимно простые с 2, не образуют относительно операции умножения по модулю 2^m циклической группы (за исключением случаев $m = 1$ и $m = 2$). Это множество чисел образует группу, изоморфную группе $Z_2 \times Z_{2^{m-2}}$. По этой причине конструкция БПФ-алгоритма Винограда длины 2^m содержит на один шаг больше. Прежде всего, отберем 2^{m-1} строк и столбцов матрицы с нечетными индексами, подобно тому как это было сделано в разд. 4.5. Это подмножество элементов матрицы будет переупорядочено теперь не в одну, а в четыре циклические свертки.

Строки и столбцы с четными индексами можно представить в виде преобразования Фурье длины 2^{m-1} несколькими способами. Эта часть вычислений представляет собой 2^{m-1}-точечное преобразование Фурье.

Разбиение в некотором смысле аналогично одному шагу алгоритма Кули—Тьюки по основанию 2. Предположим, что мы уже умеем строить 2^{m-1}-точечное быстрое преобразование Фурье. Для заданного $V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i$, $k = 0, \ldots, n-1$, компоненты с четными значениями индекса k можно записать в виде

$$V_{2k'} = \sum_{i'=0}^{n/2-1} (v_i + v_{i'+n/2})\, \omega^{2i'k'}, \quad k' = 0, \ldots, n/2 - 1.$$

Это 2^{m-1}-точечное преобразование Фурье может быть вычислено уже известным алгоритмом. Компоненты с нечетными значениями индекса k можно записать в виде

$$V_{2k'+1} = \sum_{i'=0}^{n/2-1} v_{2i'} \omega^{2i'(2k'+1)} + \sum_{i'=0}^{n/2-1} v_{2i'+1} \omega^{(2i'+1)(2k'+1)}.$$

Так как ω^4 является корнем из единицы степени 2^{m-2}, то небольшие вычисления позволяют первую сумму переписать в виде преобразования Фурье длины 2^{m-2}. Таким образом,

$$V_{2k'+1} = \sum_{i'=0}^{n/4-1} \left(\omega^{2i'} v_{2i'} - \omega^{2i'} v_{2i'+n/4}\right) \omega^{4i'k'} +$$
$$+ \sum_{i'=0}^{n/2-1} v_{2i'+1} \omega^{(2i'+1)(2k'+1)}, \quad k' = 0, \ldots, n/2 - 1,$$

где первая сумма подлежит вычислению только для первых $n/4$ значений индекса k', $k = 0, \ldots, n/4 - 1$, так как потом эти значения суммы повторяются. Если входная последовательность данных является вещественной, то эта часть вычислений состоит из $(n/2) - 6$ вещественных умножений и $(n/4)$-точечного преобразования Фурье, а если входная последовательность данных является комплексной, то к $(n/4)$-точечному преобразованию Фурье добавляется $(3/4)\, n - 8$ вещественных умножений.

Вторая сумма в уравнениях для $V_{2k'+1}$ может быть вычислена рассмотренным в предыдущем разделе обобщенным алгоритмом Рейдера. Он сводится к вычислению двух циклических сверток длины 2^{m-2}. Следующая теорема показывает, что если эти циклические свертки вычисляются на основании китайской теоремы об остатках, то часть умножений является умножением на нуль и может быть выброшена.

Теорема 4.6.3. *Пусть $m > 2$ есть целое число и $g(x, y)$ — обобщенный двумерный многочлен Рейдера вида*

$$g(x, y) = \sum_{i=0}^{n'-1} \sum_{i'=0}^{1} \omega^{3^i (-1)^{i'}} x^i y^{i'},$$

где ω является корнем степени 2^n из единицы. Тогда

(i) *Коэффициенты многочлена* $g(x, y) \pmod{y - 1}$ *являются вещественными числами.*

(ii) *Коэффициенты многочлена* $g(x, y) \pmod{y + 1}$ *являются мнимыми числами.*

(iii) *Коэффициенты многочлена* $g(x, y) \pmod{x^{n'/2} - 1}$ *равны нулю.*

Доказательство аналогично доказательству теоремы 4.6.2. □

Согласно первой части теоремы циклическая свертка по модулю $x^{n/4} - 1$ представляет собой две вещественные циклические свертки (точнее, одну чисто мнимую и одну чисто вещественную). Вторая часть теоремы утверждает, что вычисление циклической свертки сводится только к вычислениям, связанным с модулем $x^{n/8} + 1$, так как остальные вычеты обращаются в нуль. В силу неприводимости многочлена $x^{n/8} + 1$ на эти вычисления требуется $2(n/8)$ — вещественных умножений.

Теперь мы получили разбиение преобразования Фурье на следующие фрагменты: (1) 2^{m-1}-точечное преобразование Фурье; (2) 2^{m-2}-точечное преобразование Фурье, которому предшествует $2^{m-2} - 1$ комплексных умножений; (3) два произведения многочленов по модулю неприводимого многочлена $x^{n/8} + 1$.

8-точечное преобразование Фурье разбивается на две части, 4-точечное преобразование Фурье

$$\begin{bmatrix} V_0 \\ V_2 \\ V_4 \\ V_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{bmatrix} \begin{bmatrix} v_0 + v_4 \\ v_1 + v_5 \\ v_2 + v_6 \\ v_3 + v_7 \end{bmatrix}$$

и

$$\begin{bmatrix} V_1 \\ V_3 \\ V_5 \\ V_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & \omega^4 \\ 1 & 1 \\ 1 & \omega^4 \end{bmatrix} \begin{bmatrix} \omega^0(v_0 - v_4) \\ \omega^2(v_2 - v_6) \end{bmatrix} + \begin{bmatrix} \omega^1 & \omega^3 & \omega^5 & \omega^7 \\ \omega^3 & \omega^1 & \omega^7 & \omega^5 \\ \omega^5 & \omega^7 & \omega^1 & \omega^3 \\ \omega^7 & \omega^5 & \omega^3 & \omega^1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \\ v_5 \\ v_7 \end{bmatrix}.$$

Вторую часть сначала выпишем в следующем виде:

$$\begin{bmatrix} V_1 \\ V_3 \\ V_7 \\ V_5 \end{bmatrix} = \begin{bmatrix} \omega^1 & \omega^3 & \omega^7 & \omega^5 \\ \omega^3 & \omega^1 & \omega^5 & \omega^7 \\ \omega^7 & \omega^5 & \omega^1 & \omega^3 \\ \omega^5 & \omega^7 & \omega^3 & \omega^1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \\ v_7 \\ v_5 \end{bmatrix}.$$

Теперь преобразуем эти равенства несколько иначе, чем в общем случае, а именно, воспользуемся условием $\omega^4 = -1$ и перепишем уравнения в виде

$$\begin{bmatrix} V_1 \\ V_3 \\ V_7 \\ V_5 \end{bmatrix} = \begin{bmatrix} \omega^1 & -\omega^7 & \omega^7 & -\omega^1 \\ -\omega^7 & \omega^1 & -\omega^1 & \omega^7 \\ \omega^7 & -\omega^1 & \omega^1 & -\omega^7 \\ -\omega^1 & \omega^7 & -\omega^7 & \omega^1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \\ v_7 \\ v_5 \end{bmatrix}.$$

Тогда

$$\begin{bmatrix} V_1 \\ V_7 \end{bmatrix} = \begin{bmatrix} \omega^1 & \omega^7 \\ \omega^7 & \omega^1 \end{bmatrix} \begin{bmatrix} v_1 - v_5 \\ v_7 - v_3 \end{bmatrix},$$

$$\begin{bmatrix} V_5 \\ V_3 \end{bmatrix} = - \begin{bmatrix} \omega^1 & \omega^7 \\ \omega^7 & \omega^1 \end{bmatrix} \begin{bmatrix} v_1 - v_5 \\ v_7 - v_3 \end{bmatrix} = - \begin{bmatrix} V_1' \\ V_7' \end{bmatrix}.$$

Циклическую свертку можно вычислить по правилу

$$\begin{bmatrix} V_1' \\ V_7' \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 \\ 0 & j\sin\theta \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} v_1 - v_5 \\ v_7 - v_3 \end{bmatrix}.$$

Описанный 8-точечный БПФ-алгоритм Винограда содержит всего два нетривиальных умножения; кроме того, имеется шесть тривиальных умножений.

Если длина преобразования равна большой степени двух, то выписывать все необходимые для вычисления по БПФ-алгоритму Винограда уравнения становится скучно. Лучше представить этот алгоритм в рекуррентном виде. Мы отложим такое представление до гл. 10. В разд. 10.4 будет описан эффективный БПФ-алгоритм по основанию два, построенный на идее Винограда, выписанной в рекуррентной форме.

Задачи

4.1. Пусть задано устройство вычисления n-точечного комплексного преобразования Фурье; описать, как это устройство можно использовать для одновременного вычисления двух вещественных n-точечных преобразований Фурье.

4.2. Пусть задано устройство вычисления n-точечного преобразования Фурье; описать, как это устройство можно использовать для вычисления n-точечного обратного преобразования Фурье.

4.3. Приведите блок-схему устройства вычисления 5-точечного преобразования Фурье, основанного на чирп-алгоритме Блюстейна.

4.4. **а.** Доказать, что 2 является примитивным элементом поля $GF(11)$.

б. Используя алгоритм Рейдера, записать 11-точечное преобразование Фурье над полем комплексных чисел $V_k = \sum_{i=0}^{10} \omega^{ik} v_i$ в виде свертки

$s(x) = g(x)\,d(x)$, выписывая многочлены $d(x)$, $s(x)$ и $g(x)$ через компоненты векторов $\mathbf{v}$ и $\mathbf{V}$ и элемент ω.

4.5. Показать, что использование для вычисления линейной свертки двух последовательностей длины $n/2$ алгоритма Кука — Тоома с n точками $\beta_t = (e^{-t2\pi/n})^t$ приводит к тому же самому алгоритму, что и использование преобразования Фурье и теоремы о свертке.

4.6. Указать две различные причины того, что сочетание алгоритма Рейдера для простых чисел с алгоритмом Винограда вычисления свертки приводит к лучшему БПФ-алгоритму, чем сочетание алгоритма Блюстейна с алгоритмом Винограда вычисления свертки. Можете ли Вы указать третью причину?

4.7. Найти простые целые числа n и n', такие что $n < n'$, но n-точечный БПФ-алгоритм Винограда содержит больше умножений, чем n'-точечный БПФ-алгоритм Винограда.

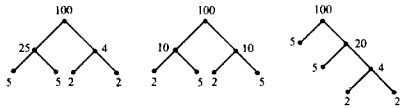
4.8. Сколько умножений содержит 25-точечный БПФ-алгоритм Винограда, если все составляющие свертки оптимальны? Как это соотносится с процедурой, основанной на алгоритме Кули — Тьюки, сочетающем два 5-точечных БПФ-алгоритма Винограда?

4.9. Показать, что связывая вход с выходом, любой БПФ-алгоритм можно использовать для вычисления обратного преобразования Фурье.

4.10. В алгоритме Гуда — Томаса вход и выход переиндексируются по-разному. Показать, что можно поменять местами схемы индексации входа и выхода без принципиальных изменений алгоритма.

4.11. Даны вычисляемые непосредственно 2-точечное и 5-точечное преобразования Фурье (с 4 и 25 умножениями соответственно).

а. Найти число умножений, необходимых для вычисления 100-точечного преобразования Фурье по каждой из нижеследующих схем:



Использовать всюду, где можно, алгоритм Гуда — Томаса; в других случаях использовать алгоритм Кули — Тьюки. Найти число умножений, включая тривиальные (на ± 1, $\pm j$).

б. Найти число умножений без учета тривиальных умножений на ± 1, $\pm j$.

в. Теперь предположим, что имеется подпрограмма вычисления 2-точечного преобразования Фурье без нетривиальных умножений и подпрограмма 5-точечного преобразования Фурье с пятью (нетривиальными) умножениями. Повторить п. б) задачи.

4.12. Из того, что возникающие в алгоритме Рейдера свертки вычисляются оптимальными алгоритмами, еще не вытекает оптимальность БПФ-алгоритма Винограда. Это следует и из того, что коэффициенты фильтра Рейдера не произвольные числа, а степени элемента ω, которые, вообще говоря, могут оказаться зависимыми, и эта зависимость может быть использована для уменьшения числа умножений. А именно, мы видели, что комплексные умножения сводятся к вещественным и иногда коэффициенты оказываются равными нулю. Доказать, что 5-точечный БПФ-алгоритм Винограда оптимален по критерию числа умножений.

4.13. 16-точечный БПФ-алгоритм Винограда содержит 18 умножений, 8 из которых тривиальны, и 74 сложения.

а. Построить 256-точечный БПФ-алгоритм, используя для построения 16-точечный БПФ-алгоритм Винограда и алгоритм Кули — Тьюки.

б. Сколько умножений потребуется, если входные данные вещественны?

в. Сколько умножений потребуется, если входные данные комплексны?

Замечания

В цифровой обработке сигналов алгоритмы быстрого преобразования Фурье начали широко использоваться в результате работы Кули и Тьюки [1] (1965) и близкой к ней работы Синглтона [2] (1969). Однако другие алгоритмы БПФ, основанные на китайской теореме об остатках, появились в более ранних работах Гуда [3] (1960) и Томаса [4] (1963). Различия в этих работах были описаны Гудом [5] (1971). Эффективная организация алгоритма Кули — Тьюки была предложена Рейдером и Бреннером [6] (1976) и модифицирована Преусом [7] (1982). Рейдер [8] (1968) и Блюстейн [9] (1970) описали метод сведения преобразования Фурье к свертке. Целью работы Рейдера было вычисление дискретного преобразования Фурье, длина которого равна большому простому числу; но, по иронии судьбы, этот метод оказался наиболее важным для длин, равных малым простым числам. Обобщение алгоритма Рейдера на случай длин, равных степени простого числа, было сделано Виноградом [11] (1978).

БПФ-алгоритм Винограда был анонсирован в работе [10] в 1976 г. и с подробностями опубликован в 1978 г. Наше изложение не отражает всей значимости оригинальной работы; развитие ее результатов включено в другие темы. Мы затабулировали те БПФ-модули для малых длин, которые представляются нам наиболее полезными. БПФ-модули для больших длин были построены Джонсоном и Баррасом [12] (1981). Другие методы вычисления преобразования Фурье изучались Герцелем [13] (1968) и Сервейтом [14] (1978).

Глава 5

ТЕОРИЯ ЧИСЕЛ И АЛГЕБРАИЧЕСКАЯ ТЕОРИЯ ПОЛЕЙ

В предыдущей главе при построении БПФ-алгоритмов мы уже пользовались некоторыми результатами теории чисел, которые, однако, доказаны не были. В настоящей главе, представляющей собой математическую интерлюдию, доказываются все использованные ранее результаты, а также результаты, которые понадобятся в дальнейшем.

Мы также вернемся к рассмотрению полей, поскольку нам понадобится более глубокое развитие идеи расширения полей. Алгебраическая структура поля будет играть важную роль в следующей главе при построении теоретико-числовых преобразований, а также в гл. 7 и 8 при построении многомерных алгоритмов свертки и преобразования Фурье.

5.1. Элементарная теория чисел

В кольце целых чисел Z_q по модулю q некоторые элементы взаимно просты с q, а некоторые делят q. Нам важно знать, сколько именно элементов относится к каждому типу.

Определение 5.1.1 (Эйлер). *Функция Эйлера*, обозначаемая $\varphi(q)$, для $q > 1$ определяется как число ненулевых элементов в Z_q, взаимно простых с q; $\varphi(q) = 1$ для $q = 1$.

Если $q = p$ простое, то все ненулевые элементы кольца Z_q взаимно просты с q, так что $\varphi(p) = p - 1$. Если q равно степени простого числа, $q = p^m$, то не взаимно простыми с q являются только те элементы, которые кратны p. Следовательно, $\varphi(p^m) = p^{m-1}(p-1)$. Все остальные значения функции Эйлера можно получить, используя следующую теорему.

Теорема 5.1.2. *Если* НОД $(q', q'') = 1$, *то* $\varphi(q'q'') = \varphi(q')\varphi(q'')$.

Доказательство. Занумеруем элементы кольца $Z_{q'q''}$ индексом i для $i = 0, \ldots, q'q'' - 1$, а затем выпишем их в виде двумерной таблицы, нумеруя каждый элемент кольца $Z_{q'q''}$ парой индексов по правилу

$$i' = i \pmod{q'}, \qquad i'' = i \pmod{q''},$$

так что i' и i'' являются соответственно элементами колец $Z_{q'}$ и $Z_{q''}$. Так как НОД $(q', q'') = 1$, то отображение индекса i в пару индексов (i', i'') является взаимно однозначным, и для некоторых Q' и Q'' выполняются равенства

$$i = q'Q' + i', \quad i = q''Q'' + i''.$$

Предположим, что i' не имеет общих делителей с q', а i'' не имеет общих делителей с q''. Тогда i не имеет общих делителей ни с q', ни с q'', и, следовательно, общих делителей с $q'q''$. Следовательно, $\varphi(q'q'') \geqslant \varphi(q')\varphi(q'')$.

Обратно, если i не имеет общих делителей с $q'q''$, то i не имеет общих делителей ни с q', ни с q''. Следовательно, i' не имеет общих делителей с q', а i'' не имеет общих делителей с q''. Таким образом, $\varphi(q'q'') \leqslant \varphi(q')\varphi(q'')$, и теорема доказана. □

Следствие 5.1.3. *Если разложение числа q на простые множители дается равенством $q = p_1^{c_1} p_2^{c_2} \ldots p_r^{c_r}$, то*

$$\varphi(q) = p_1^{c_1-1} p_2^{c_2-1} \ldots p_r^{c_r-1} (p_1 - 1)(p_2 - 1) \ldots (p_r - 1). \quad \square$$

Часто оказывается полезным другое важное свойство функции Эйлера. Предположим, что d делит q и обозначим через $f(q)$ некоторую функцию от q. Под символом $\sum_{d \mid q} f(d)$ будем понимать сумму значений $f(d)$ для всех d, делящих q. Очевидно, что

$$\sum_{d \mid q} \varphi(d) = \sum_{d \mid q} \varphi\left(\frac{q}{d}\right),$$

так как (q/d) делит q всякий раз, когда d делит q. Более удивительной является следующая теорема.

Теорема 5.1.4. *Функция Эйлера удовлетворяет равенству*

$$\sum_{d \mid q} \varphi(d) = q.$$

Доказательство. Для каждого d, делящего q, рассмотрим в Z_q множество $\{i \mid \text{НОД}(i, q) = d\}$. Каждый элемент кольца Z_q принадлежит точно одному такому множеству. Следовательно, суммируя числа элементов во всех этих множествах, мы должны получить q.

Теперь рассмотрим эквивалентное определение этого же множества, $\left\{i \mid \text{НОД}\left(\frac{i}{d}, \frac{q}{d}\right) = 1\right\}$. Это множество содержит

точно $\varphi(q/d)$ элементов. Суммируя по всем подмножествам, получаем

$$\sum_{d \mid q} \varphi\left(\frac{q}{d}\right) = q,$$

откуда вытекает утверждение теоремы. □

Элементы кольца Z_q относительно сложения образуют группу. Относительно умножения ненулевые элементы кольца Z_q также могут образовывать группу, хотя и не обязательно. Однако в Z_q всегда можно указать подмножество, образующее относительно умножения в кольце группу, и, следовательно, являющееся подгруппой группы ненулевых элементов кольца Z_q в том случае, когда ненулевые элементы также образуют подгруппу относительно умножения.

Пусть a — произвольный элемент кольца Z_q. Порождаемая элементом a циклическая группа равна множеству $\{a, a^2, a^3, \ldots, a^{m-1}, 1\}$. *Порядком элемента* a называется число элементов в циклической подгруппе, порождаемой элементом a. Пусть G_q обозначает множество положительных целых чисел, меньших q и взаимно простых с q. Относительно умножения по модулю q множество G_q образует группу с $\varphi(q)$ элементами. Следующая теорема дает информацию о порядке элементов из группы G_q.

Теорема 5.1.5 (теорема Эйлера). *Если* НОД $(a, q) = 1$, *то* $a^{\varphi(q)} = 1 \pmod q$, *так что порядок элемента* a *делит* $\varphi(q)$.

Доказательство. Утверждение вытекает из теоремы 2.1.5, так как группа G_q содержит $\varphi(q)$ элементов. □

Следствие 5.1.6 (теорема Ферма). *Если* p *простое, то для каждого* a *выполняется равенство* $a^{p-1} = 1 \pmod p$, *или, эквивалентно,* $a^p = a \pmod p$.

Доказательство является тривиальным следствием теоремы Эйлера, так как $\varphi(p) = p - 1$ для любого простого p. □

Теорема Ферма является также простым следствием следующей теоремы.

Теорема 5.1.7. *Если* p *просто, то относительно умножения по модулю* p *ненулевые элементы кольца* $Z/(p)$ *образуют циклическую группу, порождаемую примитивным элементом* π.

Доказательство сразу вытекает из теоремы 5.6.2, которую мы оставляем на конец главы. □

Согласно теореме Эйлера, если a и q взаимно просты, то порядок элемента a делит $\varphi(q)$, но не обязательно равен $\varphi(q)$. Следующая теорема дает более подробную информацию об этом для

случая q, равного степени простого числа. Это весьма сложная теорема теории чисел, и ее доказательство занимает всю оставшуюся часть настоящего раздела. Оно опирается на еще не доказанную теорему 5.1.7, что, однако, не создает порочного круга, так как в доказательстве теоремы 5.1.7 не используется теорема 5.1.8.

Прежде чем переходить к этой теореме, приведем два примера. Пусть $q = 9$; тогда $G_9 = \{1, 2, 4, 5, 7, 8\}$. Легко проверить, что порядок элемента 2 равен 6, так что его можно использовать в качестве образующей циклической группы G_9.

Пусть $q = 16$. Тогда $G_{16} = \{1, 3, 5, 7, 9, 11, 13, 15\}$. Перебирая все элементы, можно убедиться, что наибольший порядок всех элементов равен 4. (Таким элементом порядка 4 является элемент 3.) Следовательно, группа G_{16} не является циклической.

Так как в следующей теореме q равно степени простого числа p, $q = p^m$, то $\varphi(p^m) = p^m - p^{m-1} = p^{m-1}(p - 1)$.

Теорема 5.1.8. *Пусть число* p *простое; тогда:*

(i) *Если* p *нечетно, то группа* G_{p^m} *является циклической и изоморфной группе* $Z_{p^{m-1}(p-1)}$.

(ii) *Если* $p = 2$ *и* $p^m \geqslant 8$, *то группа* G_{p^m} *не является циклической и изоморфна группе* $Z_2 \times Z_{2^{m-2}}$.

(iii) *Если* $p^m = 4$, *то группа* G_{p^m} *изоморфна группе* Z_2.

Доказательство. Утверждение п. (iii) теоремы тривиально, так как существует только одна группа с двумя элементами. Доказательство остальных утверждений теоремы разбивается на пять шагов. Шаг 1 и шаг 2 содержат доказательство п. (i). На шаге 1 доказывается, что для нечетного p найдется элемент порядка p^{m-1}; на шаге 2 доказывается, что для нечетного p найдется элемент порядка $p - 1$. Так как p^{m-1} и $(p - 1)$ взаимно просты, то произведение этих двух элементов является элементом порядка $p^{m-1}(p - 1)$, что и доказывает п. (i) теоремы.

Доказательство п. (ii) теоремы дается шагами 3 и 4. На шаге 3 доказывается, что если $p = 2$, то порядок каждого элемента делит 2^{m-2}. На шаге 4 доказывается, что найдется элемент порядка 2^{m-2}.

Начнем с доказательства на шаге 0 одного полезного соотношения.

Шаг 0. Для всех целых чисел a и b и произвольной степени p^m простого числа p выполняется сравнение

$$(a + bp)^{p^{m-1}} \equiv a^{p^{m-1}} \pmod{p^m}.$$

Доказательство этого шага проведем индукцией по m. Для $m = 1$ утверждение проверяется непосредственной проверкой.

Предположим, что для некоторого m справедливо

$$(a+bp)^{p^{m-1}} \equiv a^{p^{m-1}} \pmod{p^m}.$$

Тогда для некоторого целого k

$$(a+bp)^{p^{m-1}} = a^{p^{m-1}} + kp^m.$$

Возведем это выражение в p-ю степень:

$$\left((a+bp)^{p^{m-1}}\right)^p = \left(a^{p^{m-1}} + kp^m\right)^p,$$

$$(a+bp)^{p^m} = a^{p^m} + pkp^m a^{(p-1)\,p^{m-1}} + \sum_{i=2}^{p} \binom{p}{i} k^i p^{im} a^{(p-i)\,p^{m-1}}.$$

Второй член справа, так же как и все члены суммы, делится на p^{m+1}; следовательно,

$$(a+bp)^{p^m} \equiv a^{p^m} \pmod{p^{m+1}},$$

так что утверждение справедливо при переходе от m к $m+1$, что и завершает индукцию.

Шаг 1. Положим в утверждении шага 0 числа a и b равными 1. Тогда

$$(1+p)^{p^{m-1}} \equiv 1 \pmod{p^m},$$

так что порядок элемента $(1+p)$ делит p^{m-1}. Покажем теперь, что при нечетном p порядок этого элемента в точности равен p^{m-1}, для чего докажем, что этот порядок не делит числа p^{m-2}. А именно, для завершения доказательства шага 1 докажем, что если $m > 1$, то

$$(1+p)^{p^{m-2}} \equiv 1 + p^{m-1} \pmod{p^m}.$$

Непосредственно проверяется справедливость утверждения для $m = 2$. Предположим, что оно выполняется для некоторого m. Тогда для некоторого k выполняется равенство

$$(1+p)^{p^{m-2}} = 1 + p^{m-1} + kp^m.$$

Следовательно,

$$(1+p)^{p^{m-1}} = 1 + p\left(p^{m-1} + kp^m\right) + \sum_{i=2}^{p-1} \binom{p}{i} \left(p^{m-1} + kp^m\right)^i + \left(p^{m-1} + kp^m\right)^p.$$

Если p нечетно, то каждое из чисел $\binom{p}{2}, \binom{p}{3}, \ldots, \binom{p}{p-1}$ делится на p. Каждый член в сумме делится по меньшей мере на $p^{2(m-1)+1}$, и, следовательно, делится на p^{m+1} при $m \geqslant 2$. Последний член в правой части равенства делится на $p^{p(m-1)}$, и, следовательно, при $p \geqslant 3$ и $m \geqslant 2$ делится на p^{m+1}. (Заметим, что доказательство не проходит при $p = 2$.) Следовательно, порядок элемента $(1+p) \pmod{p^m}$ не делит p^{m-2}. Это завершает доказательство того, что порядок элемента $(1+p)$ равен p^{m-1}.

Шаг 2. Так как группа G_{p^m} содержит $p^{m-1}(p-1)$ элементов, то порядок каждого ее элемента делит $p^{m-1}(p-1)$. Чтобы найти элемент группы порядка $(p-1) \pmod{p^m}$, выберем π равным примитивному элементу кольца $Z/(p)$. Тогда порядок π равен $(p-1) \pmod{p}$. Покажем, что $\alpha = \pi^{p^{m-1}}$ представляет собой искомый элемент порядка $(p-1) \pmod{p^m}$. Так как $\pi^{p^{m-1}(p-1)} = 1$, то порядок элемента α должен делить $p-1$; поэтому надо только показать, что порядок элемента α не меньше, чем $p-1$.

Так как π примитивен в кольце $Z/(p)$, то $p-1$ степеней π^i для $i = 0, \ldots, p-2$, могут быть записаны в виде целых чисел $\pi^i = a_i + b_i p$, где a_i пробегает все различные ненулевые целые числа, не превосходящие $p-1$. Следовательно, используя шаг 0, имеем

$$(\pi^i)^{p^{m-1}} = (a_i + b_i p)^{p^{m-1}} = a_i^{p^{m-1}} \pmod{p^m},$$

или

$$\alpha^i = a_i^{p^{m-1}} \pmod{p^m},$$

где a_i пробегает все различные ненулевые целые числа, не превосходящие $p-1$. Следовательно, если мы докажем, что для любых двух целых чисел a и b, не превосходящих $p-1$, выполняется соотношение

$$a^{p^{m-1}} \not\equiv b^{p^{m-1}} \pmod{p^m},$$

то отсюда следует, что порядок элемента α больше, чем $p-2$. Но $a^p = a$ для любого элемента кольца $Z/(p)$, так что $a^{p^{m-1}} = a \pmod{p}$. Следовательно, если

$$a^{p^{m-1}} = b^{p^{m-1}} \pmod{p^m},$$

то, конечно,

$$a^{p^{m-1}} = b^{p^{m-1}} \pmod{p},$$

и, таким образом,

$$a = b \pmod{p},$$

что противоречит предположению о том, что a и b различные не превосходящие $p-1$ целые числа. Следовательно, порядок элемента α равен $p-1$ и п. (i) теоремы доказан.

Шаг 3. Порядок каждого элемента группы G_{2^m} делит 2^{m-2}. Чтобы доказать это, сначала докажем, что для любых целых a и b выполняется сравнение

$$(a+4b)^{2^{m-2}} \equiv a^{2^{m-2}} \pmod{2^m}.$$

Поскольку элементами группы G_{2^m} являются только нечетные числа, то, полагая $a = \pm 1$ и варьируя b, сразу получаем из этого сравнения нужное утверждение.

Сравнение доказывается индукцией. Для $m = 2$ оно проверяется непосредственно. Предположим, что оно справедливо для некоторого положительного целого m. Тогда для некоторого целого k

$$(a + 4b)^{2^{m-2}} = a^{2^{m-2}} + k2^m,$$

и, следовательно,

$$(a + 4b)^{2^{m-1}} = \left(a^{2^{m-2}} + k2^m\right)^2 = a^{2^{m-1}} + ka^{2^{m-2}}2^{m+1} + k^2 2^{2m}.$$

Таким образом,

$$(a + 4b)^{2^{m-1}} \equiv a^{2^{m-1}} \pmod{2^{m+1}}.$$

Шаг 4. Нам осталось найти элемент порядка 2^{m-2} при $m \geqslant 3$. Покажем, что $3^{2^{m-3}} \not\equiv 1 \pmod{2^m}$, так что 3 должно быть элементом порядка 2^{m-2}. При $m = 3$ это утверждение проверяется непосредственно. Доказательство утверждения при $m \geqslant 4$ состоит в доказательстве сравнения

$$(1 + 2)^{2^{m-3}} \equiv 1 + 2^{m-1} \pmod{2^m}.$$

Это сравнение просто проверяется для $m = 4$. Предположим, что оно справедливо для некоторого $m > 4$. Тогда для некоторого k

$$(1 + 2)^{2^{m-3}} = 1 + 2^{m-1} + k2^m.$$

Следовательно,

$$(1 + 2)^{2^{m-2}} = (1 + 2^{m-1} + k2^m)^2 =$$
$$= 1 + 2^m + 2^{2(m-1)} + k2^{m+1} + k2^{2m} + k^2 2^{2m},$$

и, таким образом, так как m больше трех,

$$(1 + 2)^{2^{m-2}} \equiv 1 + 2^m \pmod{2^{m+1}}.$$

Следовательно, при всех $m > 3$ выполняется сравнение

$$(1 + 2)^{2^{m-3}} \equiv 1 + 2^{m-1} \pmod{2^m}.$$

Таким образом, порядок 3 (mod 2^m) не делит 2^{m-3}, и, следовательно, он равен 2^{m-2}, что и завершает доказательство. □

5.2. Конечные поля, основанные на кольце целых чисел

Имеется очень важная конструкция, позволяющая по заданному кольцу построить новое кольцо, называемое его *факторкольцом*. В случае произвольного кольца построение его факторкольца опирается на смежные классы. В случае кольца целых чисел факторкольцо строится просто: оно представляет собой кольцо целых чисел по модулю q, с которым мы уже встречались ранее, и обозначается через $Z/(q)$ (или, более кратко, Z_q); в этом случае его называют кольцом вычетов.

Пусть q — положительное целое число. *Кольцом вычетов* $Z/(q)$ называется множество $\{0, 1, \ldots, q-1\}$ с операциями сложения и умножения, определяемыми равенствами $a + b = R_q[a + b]$, $a \cdot b = R_q[ab]$.

Элементы, обозначенные через $\{0, 1, \ldots, q-1\}$, принадлежат как Z, так и $Z/(q)$. Элементы кольца $Z/(q)$ названы так же, как и первые q элементов кольца Z. Дело вкуса, подразумевать ли под ними те же самые математические объекты или некоторые другие объекты с теми же именами.

Два элемента a и b кольца Z, отображаемые в один и тот же элемент кольца $Z/(q)$, называются сравнимыми по модулю q, и $a = b + mq$ для некоторого целого m.

Теорема 5.2.1. *Кольцо вычетов $Z/(q)$ является кольцом.*

Доказательство. Непосредственная проверка выполнения свойств кольца. □

Теорема 5.2.2. *Ненулевой элемент s кольца $Z/(q)$ обратим относительно умножения тогда и только тогда, когда s и q взаимно просты.*

Доказательство. Пусть s — такой ненулевой элемент кольца, что s и q взаимно просты. Тогда согласно следствию 2.6.4 найдутся целые числа a и b, такие, что $1 = aq + bs$. По модулю q соответственно имеем

$$1 = R_q[1] = R_q[aq + bs] = R_q\{R_q[aq] + R_q[bs]\} =$$
$$= R_q[bs] = R_q\{R_q[b]\,R_q[s]\} = R_q\{R_q[b] \cdot s\}.$$

Следовательно, относительно умножения по модулю q мультипликативный обратный к s равен $R_q[b]$.

Пусть теперь s такой элемент кольца, что s и q не взаимно просты. Сначала рассмотрим случай, когда s делит q: $q = s \cdot r$. Пусть у элемента s есть обратный s^{-1}. Тогда

$$r = R_q[r] = R_q[s^{-1} \cdot s \cdot r] = R_q[s^{-1} \cdot q] = 0.$$

Но $r \not\equiv 0 \pmod{q}$, так что получаем противоречие. Таким образом, если s является делителем q, то он не имеет обратного.

Теперь рассмотрим случай, когда s и q не взаимно просты, но s не делит q. Пусть $d = \text{НОД}[s, q]$. Тогда $s = ds'$ для некоторого s', взаимно простого с q, и делителя d числа q. Если элемент s обратим, то и элемент d также обратим: $d^{-1} = s^{-1}s'$, что

противоречит только что доказанному утверждению. Следовательно, s не может быть обратимым, и теорема доказана. □

Поскольку обращение в кольце $Z/(q)$ возможно не всегда, то в общем случае ненулевые элементы кольца $Z/(q)$ группы по умножению не образуют. Однако в $Z/(q)$ можно найти подмножества, образующие подгруппы. Выберем в кольце $Z/(q)$ ненулевой элемент β и сформируем подмножество $\{\beta, \beta^2, \ldots, \beta^r = 1\}$, остановив процесс, как только получим единичный элемент. Мы уже видели, что единичный элемент всегда достижим и что конструкция приводит к циклической группе. Она является подгруппой кольца $Z/(q)$, и целое число r называется порядком элемента β в $Z/(q)$.

Мы уже видели в примерах разд. 2.3, что арифметика полей GF (2) и GF (3) может быть задана как сложение и умножение по модулю 2 и 3 соответственно, но арифметика поля GF (4) так описана уже быть не может. В символической записи $GF(2) = Z/(2)$, $GF(3) = Z/(3)$, $GF(4) \neq Z/(4)$. Общий результат описывается следующей теоремой.

Теорема 5.2.3. *Кольцо вычетов $Z/(q)$ является полем тогда и только тогда, когда q равно простому числу.*

Доказательство. Предположим, что q просто. Тогда каждый элемент кольца $Z/(q)$ взаимно прост с q и, следовательно (по теореме 5.2.2), обратим относительно умножения.

Предположим теперь, что q составное число. Тогда $q = r \cdot s$ и согласно теореме 5.2.2, r и s не могут иметь обратных элементов. □

Если кольцо вычетов $Z/(q)$ является полем, то чтобы подчеркнуть этот факт, оно обозначается через $GF(q)$. Конечные поля, построенные как кольца вычетов целых чисел, не исчерпывают всего множества полей, но конечные поля с простым числом элементов всегда могут быть построены как кольца вычетов.

Если поле $GF(p)$ не содержит квадратного корня из -1, то его можно расширить до $GF(p^2)$ точно таким же способом, как поле вещественных чисел расширяется до поля комплексных чисел. Например, в поле $GF(7)$ выполняется равенство $6 = -1$, но, как легко проверить, поле не содержит элемента, квадрат которого равен 6. Следовательно, поле $GF(49)$ можно задать множеством элементов вида

$$GF(49) = \{a + jb : a,\ b \in GF(7)\},$$

определив на нем операции сложения и умножения так же, как для комплексных чисел:

$$(a + jb) + (c + jd) = (a + c) + j(b + d),$$

$$(a + jb)(c + jd) = (ac - bd) + j(ab + cd),$$

где во всех скобках справа операции обозначают сложение и умножение в поле $GF(7)$. При таком определении множество $GF(49)$ является полем. Целыми поля $GF(49)$ являются элементы поля $GF(7)$, так что характеристика поля $GF(49)$ равна 7.

Теорема 5.2.4. *Каждое поле содержит однозначно определяемое наименьшее подполе, которое либо изоморфно полю рациональных чисел, либо содержит конечное число элементов. Следовательно, характеристика каждого поля Галуа либо бесконечна, либо равна простому числу.*

Доказательство. Каждое поле содержит 0 и 1. Для построения подполя рассмотрим подмножество $G = \{0, 1, 1 + 1, 1 + 1 + 1, \ldots\}$, обозначая его элементы соответственно $\{0, 1, 2, 3, \ldots\}$. Это подмножество содержит либо бесконечное множество элементов, либо конечное множество, скажем p. Мы покажем, что если это число конечно, то оно просто и $G = GF(p)$. Так как G является циклической группой, то сложением в нем служит сложение по модулю p. В силу наличия в поле дистрибутивного закона умножение в G также является умножением по модулю p, так как

$$\alpha \cdot \beta = (1 + 1 + \ldots + 1)\beta = \beta + \beta + \ldots + \beta,$$

где сумма содержит α копий элемента β и сложение выполняется как сложение по модулю p. Следовательно, умножение также является умножением по модулю p. Относительно умножения каждый ненулевой элемент β имеет обратный, так как последовательность $\beta, 2\beta, 3\beta, \ldots$ образует в G циклическую подгруппу. Поскольку $\alpha \cdot \beta \neq 0$ для любых ненулевых α и β из исходного поля, то $\alpha\beta \neq 0$ по модулю p для всех целых поля, и, следовательно, p должно быть простым. Таким образом, подмножество G образует в точности простое поле, описываемое теоремой 5.2.3.

Альтернативным является случай, когда множество G бесконечно. В этом случае оно изоморфно кольцу целых чисел. Наименьшее содержащее G подполе F изоморфно наименьшему полю, содержащему кольцо целых чисел, каковым является поле рациональных чисел. □

В поле Галуа $GF(q)$ всегда можно найти простое подполе $GF(p)$. В частности, если то q, с которого мы начинаем, само является простым числом, то это простое подполе можно интерпретировать как поле вычетов целых чисел по модулю q. Следовательно, на самом деле имеется только одно поле с заданным числом элементов, которое, конечно, допускает много различных представлений.

5.3. Поля, основанные на кольцах многочленов

Поля можно строить, отталкиваясь от колец многочленов, таким же образом, как были построены конечные поля из кольца целых чисел. Такая конструкция приводит как к конечным, так и к бесконечным полям в зависимости от того, было ли исходное кольцо многочленов определено над конечным или бесконечным полем. Эти расширения оказались полезными в нескольких направлениях дискретной обработки сигналов. Они были использованы для построения теоретико-числовых преобразований, рассматриваемых в гл. 6, и для построения алгоритмов многомерной свертки, рассматриваемых в гл. 7.

Пусть имеется кольцо многочленов $F[x]$ над полем F. Так же как для кольца Z были построены кольца вычетов, можно построить и для кольца $F[x]$ его факторкольцо. Выбирая из $F[x]$ произвольный многочлен $p(x)$, можно определить такое кольцо, используя $p(x)$ в качестве модуля для задания арифметики этого кольца. Мы ограничим рассмотрение только приведенными многочленами, так как это ограничение снимает ненужную неопределенность построения.

Определение 5.3.1. Для произвольного приведенного многочлена $p(x)$ ненулевой степени над полем F *кольцом многочленов по модулю* $p(x)$ называется множество всех многочленов над F, степень которых не превосходит степени многочлена $p(x)$, с операциями сложения и умножения многочленов по модулю $p(x)$. Это кольцо принято обозначать через $F[x]/(p(x))$.

Произвольный многочлен $r(x)$ из $F[x]$ можно отобразить в $F[x]/(p(x))$, отображая $r(x)$ в $R_{p(x)}[r(x)]$. Два сравнимых по модулю $p(x)$ элемента $a(x)$ и $b(x)$ из $F(x)$ отображаются в один и тот же многочлен из $F[x]/(p(x))$.

Теорема 5.3.2. *Кольцо многочленов по модулю* $p(x)$ *является кольцом.*

Доказательство. Доказательство проводится прямой проверкой выполнения свойств кольца. □

В качестве примера рассмотрим кольцо многочленов над $GF(2)$, выбирая модуль равным $p(x) = x^3 + 1$. Тогда кольцо многочленов по модулю $p(x)$ равно $GF(2)[x]/(x^3 + 1)$. Оно состоит из элементов $\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1,\}$ и умножение в этом кольце выполняется, например, следующим образом:

$$(x^2+1)\cdot(x^2) = R_{x^3+1}[(x^2+1)x^2] = R_{x^3+1}[x(x^3+1)+x^2+x] = \\ = x^2+x,$$

где использована редукция по правилу $x^4 = x(x^3+1)+x$.

В качестве другого примера рассмотрим кольцо многочленов над полем рациональных чисел $\mathbb{Q}$, выбирая $p(x) = x^2 + 1$. Тогда кольцо многочленов по модулю $p(x)$ есть $\mathbb{Q}[x]/(x^2+1)$. Оно содержит многочлены вида $a + xb$, где a и b — рациональные числа. Умножение выполняется по правилу

$$(a + xb)(c + xd) = (ac - bd) + x(ad + bc).$$

Умножение имеет ту же структуру, что и комплексное умножение; на самом деле рассматриваемое кольцо является полем. Это вытекает из следующей общей теоремы.

Теорема 5.3.3. *Кольцо многочленов по модулю приведенного многочлена* $p(x)$ *является полем тогда и только тогда, когда многочлен* $p(x)$ *прост* [1]).

Доказательство. Пусть многочлен $p(x)$ прост. Чтобы доказать, что рассматриваемое кольцо образует поле, достаточно показать, что каждый ненулевой элемент имеет мультипликативный обратный. Пусть $s(x)$ — некоторый ненулевой элемент кольца. Тогда $\deg s(x) < \deg p(x)$. Так как многочлен $p(x)$ прост, то НОД $[s(x), p(x)] = 1$. Согласно следствию 2.7.7,

$$1 = a(x)\,p(x) + b(x)\,s(x)$$

для некоторых многочленов $a(x)$ и $b(x)$. Следовательно,

$$1 = R_{p(x)}[1] = R_{p(x)}[a(x)\,p(x) + b(x)\,s(x)] = \\ = R_{p(x)}[b(x)]\cdot R_{p(x)}[s(x)] = R_{p(x)}[b(x)]\,s(x).$$

Таким образом, в кольце многочленов по модулю многочлена $p(x)$ многочлен $R_{p(x)}[b(x)]$ является мультипликативным обратным к $s(x)$.

Теперь предположим, что многочлен $p(x)$ не прост. Тогда $p(x) = r(x)\,s(x)$ для некоторых $r(x)$ и $s(x)$. Если кольцо является полем, то многочлен $r(x)$ имеет обратный $r^{-1}(x)$, и поэтому

$$s(x) = R_{p(x)}[s(x)] = R_{p(x)}[r^{-1}(x)\cdot r(x)\cdot s(x)] = \\ = R_{p(x)}[r^{-1}(x)\cdot p(x)] = 0.$$

Но $s(x) \neq 0$, и мы получаем противоречие. Следовательно, такое кольцо не может быть полем. □

Теорема описывает один из способов построения поля комплексных чисел $\mathbb{C}$ как расширения поля вещественных чисел $\mathbb{R}$. Так как многочлен $x^2 + 1$ является простым над полем вещественных

[1]) Напомним что простой многочлен является одновременно приведенным и неприводимым. Для построения поля достаточно только неприводимости $p(x)$, но мы условились рассматривать только приведенные многочлены, так что дальнейшие результаты носят менее общий характер.

чисел, то $\mathbb{R}$ можно расширить до множества $\{a + bx\}$, где a и b вещественны, задавая сложение и умножение по модулю $x^2 + 1$. Тогда

$$(a + bx)(c + dx) = (ac - bd) + (ad + bc)x,$$

$$(a + bx) + (c + dx) = (a + c) + (b + d)x.$$

Эти формулы будут более привычны, если вместо x поставить j:

$$(a + jb) + (c + jd) = (a + c) + j(b + d),$$

$$(a + jb) \quad (c + jd) = (ac - bd) + j(ad + bc).$$

Эта же самая конструкция может быть использована для расширения любого поля, над которым многочлен $x^2 + 1$ простой. Таким полем является, например, GF (7), и его можно расширить до поля GF (49) точно так же, как поле вещественных чисел до поля комплексных чисел, что и было формально проделано перед доказательством последней теоремы.

С другой стороны, многочлен $x^2 + 1$ не является простым многочленом над полем GF (5) и не может быть использован для его расширения. Чтобы построить расширение GF (25) поля GF (5), надо воспользоваться многочленом $x^2 + x + 1$, который является простым над GF (5). При этом модуле правило умножения приобретает вид

$$(a + xb)(c + xd) = (ac - bd) + x(ad + bc - bd).$$

Умножение в этом поле отлично от умножения в комплексном поле. Умножение всегда будет задаваться этим правилом, если поле $GF(p^2)$ получается как расширение поля $GF(p)$ с помощью многочлена $x^2 + x + 1$, и такое расширение всегда возможно, если $x^2 + x + 1$ является простым над $GF(p)$. В частности, этот многочлен можно использовать при расширении GF (2) до GF (4).

5.4. Минимальные многочлены и сопряжения

Элемент некоторого поля может быть корнем многочлена, коэффициенты которого принадлежат некоторому меньшему полю. Если элемент является корнем хотя бы одного такого многочлена, то с ним оказывается связанным один специальный многочлен, представляющий особый интерес.

Определение 5.4.1. Пусть F — некоторое поле, а элемент β принадлежит некоторому расширению поля F. *Минимальным многочленом* $f(x)$ элемента β над полем F называется ненулевой приведенный многочлен наименьшей степени, такой что коэффициенты его принадлежат основному полю F, а элемент β является корнем.

Минимальный многочлен над полем F существует не для всякого элемента расширения — элементы вещественного поля, не имеющие минимальных многочленов с рациональными коэффициентами, называются *трансцендентными* числами — но если для данного элемента минимальный многочлен существует, то он единственен. Действительно, если $f^{(1)}(x)$ и $f^{(2)}(x)$ оба являются минимальными многочленами элемента β, то они оба приведенные, имеют одну и ту же степень и β является их корнем. Тогда многочлен $f(x) = f^{(1)}(x) - f^{(2)}(x)$ имеет меньшую степень и β является его корнем. Следовательно, $f(x) = 0$ и $f^{(1)}(x)$ равен $f^{(2)}(x)$.

Теорема 5.4.2. *Каждый минимальный многочлен является единственным и простым. Если $f(x)$ — минимальный многочлен элемента β, а $g(x)$ — некоторый многочлен, для которого β является корнем, то $f(x)$ делит $g(x)$.*

Доказательство. Единственность минимального многочлена мы уже доказали. По определению многочлен $f(x)$ является приведенным. Предположим, что $f(x)$ разлагается в произведение двух многочленов, $f(x) = a(x)\,b(x)$. Тогда $f(\beta) = a(\beta)\,b(\beta) = 0$, так что хотя бы один из многочленов $a(x)$ и $b(x)$ имеет степень, меньшую чем степень $f(x)$, и элемент β служит его корнем. Следовательно, минимальный многочлен является простым.

Для доказательства второй части теоремы запишем

$$g(x) = f(x)\,h(x) + s(x).$$

Степень многочлена $s(x)$ меньше степени $f(x)$, так что $s(x)$ не может иметь элемент β своим корнем. Но

$$0 = g(\beta) = f(\beta)\,h(\beta) + s(\beta) = s(\beta),$$

так что многочлен $s(x)$ равен нулю, и теорема доказана. □

Каждый элемент β поля F имеет над F минимальный многочлен первой степени, равный $f(x) = x - \beta$. Если элемент β не принадлежит полю F, то степень его минимального многочлена равна двум или больше. Следовательно, минимальный многочлен может иметь и другие корни. Но тогда минимальный многочлен элемента β является минимальным многочленом и для других, отличных от β, элементов.

Определение 5.4.3. Два элемента некоторого расширения поля F, имеющие общий минимальный многочлен над F, называются *сопряженными* (относительно F).

В общем случае один элемент может иметь более одного сопряженного элемента, на самом деле столько, какова степень его минимального многочлена. Подчеркнем, что отношение сопряжен-

ности двух элементов зависит от основного поля. Два элемента комплексного поля могут быть сопряженными относительно поля рациональных чисел, но не быть сопряженными относительно поля вещественных чисел. Например, минимальный многочлен над Q комплексного числа $j\sqrt[4]{2}$ равен $x^4 - 2$, а над R равен $x^2 + \sqrt{2}$. Множество сопряженных с $j\sqrt[4]{2}$ элементов, содержащее этот элемент, относительно поля Q равно $\{\sqrt[4]{2}, -\sqrt[4]{2}, j\sqrt[4]{2}, -j\sqrt[4]{2}\}$, а относительно поля R равно $\{j\sqrt[4]{2}, -j\sqrt[4]{2}\}$.

5.5. Круговые многочлены

Над произвольным полем F можно выписать разложение многочлена $x^n - 1$ в произведение его простых делителей:

$$x^n - 1 = p_1(x)\, p_2(x) \ldots p_K(x).$$

Если F — поле рациональных чисел, то простые делители находятся сравнительно легко. Они представляют собой многочлены, которые даются следующим определением.

Определение 5.5.1. Над произвольным полем F для каждого n *круговой многочлен* определяется равенством

$$Q_n(x) = \prod_{\text{нод}(i,\, n)=1} (x - \omega_n^i),$$

где ω_n равно корню [1]) степени n из единицы в некотором расширении поля F.

Главным образом нас будут интересовать круговые многочлены над полем рациональных чисел Q. В этом случае $\omega_n = e^{-j2\pi/n}$ принадлежит полю комплексных чисел, а для $n < 105$ коэффициентами круговых многочленов являются -1, 0 или $+1$.

В достаточно большом расширении поля F — для поля рациональных чисел таковым является комплексное поле — многочлен $x^n - 1$ может быть разложен в виде

$$x^n - 1 = \prod_{i=0}^{n-1} (x - \omega_n^i),$$

где ω_n — корень степени n из единицы. Следовательно, каждый круговой многочлен можно выразить в виде произведения некоторых из этих линейных множителей. Определение кругового многочлена специально построено так, чтобы все его коэффициенты были рациональными.

[1]) Имеется в виду, конечно, первообразный корень степени n из единицы. — *Прим. перев.*

При малых n круговые многочлены можно легко вычислить, разлагая многочлен $x^n - 1$. Очевидно, что $Q_1(x) = x - 1$. Для выяснения общей ситуации найдем несколько начальных многочленов. Пусть $n = 2$. Тогда

$$x^2 - 1 = (x - 1)(x + 1) = Q_1(x)\, Q_2(x).$$

Пусть $n = 3$. Тогда

$$x^3 - 1 = (x - 1)(x^2 + x + 1) = Q_1(x)\, Q_3(x).$$

Пусть $n = 4$. Тогда

$$x^4 - 1 = (x - 1)(x + 1)(x^2 + 1) = Q_1(x)\, Q_2(x)\, Q_4(x).$$

Отметим, что на каждом шаге появляется только один новый множитель. Пусть $n = 5$. Тогда

$$x^5 - 1 = (x - 1)(x^4 + x^3 + x^2 + x + 1) = Q_1(x)\, Q_5(x).$$

Пусть $n = 6$. Тогда

$$x^6 - 1 = (x - 1)(x + 1)(x^2 + x + 1)(x^2 - x + 1) = Q_1(x)\, Q_2(x)\, Q_3(x)\, Q_6(x).$$

Пусть $n = 7$. Тогда

$$x^7 - 1 = (x - 1)(x^6 + x^5 + x^4 + x^3 + x^2 + x + 1) = Q_1(x)\, Q_7(x).$$

Пусть $n = 8$. Тогда

$$x^8 - 1 = (x - 1)(x + 1)(x^2 + 1)(x^4 + 1) = Q_1(x)\, Q_2(x)\, Q_4(x)\, Q_8(x).$$

Пусть $n = 9$. Тогда

$$x^9 - 1 = (x - 1)(x^2 + x + 1)(x^6 + x^3 + 1) = Q_1(x)\, Q_3(x)\, Q_9(x).$$

Общий случай описывается следующей теоремой.

Теорема 5.5.2. *Для каждого* n

$$\deg Q_n(x) = \varphi(n),$$

где $\varphi(n)$ *— функция Эйлера.*

Доказательство. $\varphi(n)$ определяется как число целых чисел, меньших n и взаимно простых с n, и это в точности совпадает с числом линейных множителей, входящих в определение $Q_n(x)$. □

Теорема 5.5.3. *Для каждого* n

$$x^n - 1 = \prod_{k \mid n} Q_k(x).$$

Доказательство. Непосредственные вычисления дают:

$$x^n - 1 = \prod_{i=1}^{n} (x - \omega_n^i) = \prod_{k \mid n} \prod_{\text{НОД}\,(i,\ n)=n/k} (x - \omega_n^i) =$$

$$= \prod_{k \mid n} Q_k(x).$$

Теорема 5.5.4. *Над произвольным полем коэффициенты кругового многочлена всегда являются целыми этого поля.*

Доказательство. Воспользуемся индукцией по n. Коэффициенты многочлена $Q_1(x) = x - 1$ являются целыми. Согласно теореме 5.5.3,

$$Q_n(x) = \frac{x^n - 1}{\prod\limits_{\substack{k \mid n \\ k < n}} Q_k(x)}.$$

Деление приведенного многочлена с целыми коэффициентами на приведенный многочлен с целыми коэффициентами дает в виде частного приведенный многочлен с целыми коэффициентами. Следовательно, утверждение теоремы верно для любого n, если оно верно для всех меньших положительных целых n. □

Теорема 5.5.5. *Над полем рациональных чисел все круговые многочлены являются простыми и, следовательно, минимальными многочленами для соответствующих корней из единицы.*

Доказательство. Приведенность круговых многочленов очевидна. Доказать надо только неприводимость круговых многочленов над полем рациональных чисел. Пусть $f(x)$ обозначает минимальный многочлен элемента ω_n над полем рациональных чисел. Пусть h — произвольное целое число, взаимно простое с n, и пусть $g(x)$ — минимальный многочлен элемента ω_n^h над полем рациональных чисел. Предположим, что $g(x) = f(x)$ для каждого такого h. Тогда многочлен $f(x)$ должен быть кратным многочлену $Q_n(x)$, и так как $Q_n(x)$ приведен и имеет рациональные коэффициенты, $f(x) = Q_n(x)$. Тогда наша задача сводится к доказательству того, что для каждого взаимно простого с n числа h минимальный многочлен $g(x)$ элемента ω_n^h равен многочлену $f(x)$.

Шаг 1. Докажем сначала это утверждение для случая, когда h равно простому числу p, не являющемуся делителем n. Пусть $g(x)$ — минимальный многочлен элемента ω_n^p и предположим, что он не равен многочлену $f(x)$. Тогда

$$x^n - 1 = f(x)\, g(x)\, t(x),$$

для некоторого многочлена $t(x)$, коэффициенты которого целые числа, так как оба многочлена $f(x)$ и $g(x)$ являются делителями

многочлена $x^n - 1$ и имеют целые коэффициенты. Так как элемент ω_n является корнем многочлена $g(x^p)$, то многочлен $g(x^p)$ кратен многочлену $f(x)$:

$$g(x^p) = f(x)\, k(x)$$

для некоторого многочлена $k(x)$ с целыми коэффициентами. Заменим теперь оба равенства вычетами по модулю p:

$$x^n - 1 = f(x)\, g(x)\, t(x) \pmod p,$$

$$g(x^p) = f(x)\, k(x) \pmod p.$$

Согласно теореме 2.2.4, второе равенство записывается в виде

$$(g(x))^p = f(x)\, k(x) \pmod p.$$

Рассмотрим разложение этого уравнения над полем $GF(p)$. Каждый простой делитель $p(x)$ многочлена $f(x)$ должен быть простым делителем многочлена $(g(x))^p$, и, следовательно, многочлена $g(x)$. Следовательно, $(x^n - 1)$ должен делиться на $(p(x))^2$. Но тогда формальная производная nx^{n-1} от многочлена $(x^n - 1)$ делится над $GF(p)$ на $p(x)$. По предположению p не делит n, а x не может быть делителем $f(x) \pmod p$, так как $f(x)$ делит $x^n - 1$. Полученное противоречие показывает, что $g(x)$ не может быть не равным $f(x)$.

Шаг 2. Теперь докажем утверждение для произвольного h, взаимно простого с n. Запишем разложение h на простые множители:

$$h = p_1 p_2 \ldots p_s.$$

Каждый из этих множителей взаимно прост с n, так как h взаимно просто с n. Согласно утверждению шага 1, если ω_n — корень многочлена $f(x)$, то $\omega_n^{p_1}$ также является корнем этого многочлена. Теперь опять воспользуемся шагом 1: если $\omega_n^{p_1}$ — корень многочлена $f(x)$, то и $\omega_n^{p_1 p_2}$ является корнем этого многочлена. Используя индукцию, получаем, что и ω_n^h является корнем $f(x)$. □

5.6. Примитивные элементы

Согласно данному в разд. 2.3 определению, примитивным элементом поля Галуа называется такой элемент α, что каждый ненулевой элемент поля можно однозначно записать в виде степени этого элемента. Например, в поле $GF(7)$ выполняются равенства $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, $3^6 = 1$, так что 3 является примитивным элементом поля $GF(7)$. Примитивные элементы полезны при построении полей, так как, коль скоро один из них найден, то, перемножая степени этого примитивного элемента, можно построить таблицу умножения в этом поле. В полях Галуа

примитивные элементы играют роль основания логарифмов. В данном разделе будет доказано, что каждое конечное поле содержит примитивный элемент.

Конечное поле образует абелеву группу двояким способом. Множество всех элементов поля образует абелеву группу по сложению, и множество всех элементов поля, исключая нуль, образует абелеву группу по умножению. Мы будем работать с группой по умножению. Согласно теореме 2.1.5, порядок этой группы делится на порядок каждого ее элемента.

Теорема 5.6.1. *Пусть* $\beta_1, \beta_2, \ldots, \beta_{q-1}$ — *ненулевые элементы поля* $GF(q)$; *тогда*:

$$x^{q-1} - 1 = (x - \beta_1)(x - \beta_2) \ldots (x - \beta_{q-1}).$$

Доказательство. Множество ненулевых элементов поля $GF(q)$ образует конечную группу по умножению. Пусть β — какой-либо элемент из $GF(q)$, и пусть h — мультипликативный порядок этого элемента. Тогда, согласно теореме 2.1.5, h делит $q-1$. Следовательно,

$$\beta^{q-1} = (\beta^h)^{(q-1)/h} = 1^{(q-1)/h} = 1,$$

так что β является корнем многочлена $x^{q-1} - 1$. □

Теорема 5.6.2. *Группа ненулевых элементов поля* $GF(q)$ *по умножению является циклической.*

Доказательство. Если число $q-1$ простое, то теорема тривиальна, ибо порядок всех элементов, за исключением нуля и единицы, равен $q-1$, так что каждый такой элемент примитивен. Доказывать теорему надо только для составного числа $q-1$. Рассмотрим разложение числа $q-1$ на простые множители:

$$q - 1 = \prod_{i=1}^{s} p_i^{\nu_i}.$$

Так как $GF(q)$ — поле, то среди его $q-1$ ненулевых элементов должен найтись хотя бы один, не являющийся корнем многочлена $x^{(q-1)/p_i} - 1$, поскольку этот многочлен может иметь не более $(q-1)/p_i$ корней. Следовательно, для каждого i можно найти такой элемент a_i поля $GF(q)$, что $a_i^{(q-1)/p_i} \neq 1$. Пусть $b_i = a_i^{(q-1)/p_i^{\nu_i}}$, и пусть $b = \prod_{i=1}^{s} b_i$. Покажем, что порядок b равен $q-1$, и, следовательно, группа является циклической.

Шаг 1. Порядок элемента b_i равен $p_i^{\nu_i}$. *Доказательство.*

Очевидно, что $b_i^{p_i^{\nu_i}} = 1$, так что порядок b_i делит $p_i^{\nu_i}$. Он равен числу вида $p_i^{n_i}$. Если n_i меньше ν_i, то $b_i^{p_i^{\nu_i - 1}} = 1$. Но $b_i^{p_i^{\nu_i - 1}} = a_i^{(q-1)/p_i} \neq 1$, и, следовательно, порядок элемента b_i равен $p_i^{\nu_i}$.

Шаг 2. Порядок элемента b равен $q-1$. *Доказательство.* Предположим, что $b^n = 1$. Покажем сначала, что из этого вытекает равенство $n \equiv 0 \pmod{p_i^{\nu_i}}$ для всех $i = 1, \ldots, s$. Для каждого i можно записать

$$b^{\left(n \prod_{j \neq i} p_j^{\nu_j}\right)} = 1.$$

Заменив b на $\prod_{i=1}^{s} b_i$ и используя равенство $b_i^{p_i^{\nu_i}} = 1$, находим

$$b_i^{\left(n \prod_{j \neq i} p_j^{\nu_j}\right)} = 1.$$

Таким образом,

$$n \prod_{j \neq i} p_j^{\nu_j} \equiv 0 \pmod{p_i^{\nu_i}}.$$

Поскольку p_i представляют собой различные простые числа, то $n \equiv 0 \pmod{p_i^{\nu_i}}$ для каждого i. Следовательно, $n \equiv 0 \pmod{q-1}$. Теорема доказана. □

Теорема 5.6.3. *В каждом поле Галуа имеется примитивный элемент.*

Доказательство. Так как ненулевые элементы поля $GF(q)$ образуют циклическую группу, то среди них имеется элемент порядка $q-1$. Этот элемент является примитивным. □

Задачи

§ 1. а. Приводим ли над полем рациональных чисел многочлен $p(x) = x^4 + 2x^3 + x^2 - 2$? Если приводим, то разложить его.
б. Приводим ли над полем вещественных чисел многочлен $p(x) = x^4 + 2x^3 + x^2 - 2$? Если приводим, то разложить его.
в. Приводим ли многочлен $p(x) = x^4 + 2x^3 + x^2 - 2$ над полем комплексных чисел? Если приводим, то разложить его.

§ 2. Над полем рациональных чисел пусть

$$p_1(x) = x^3 + 1, \quad p_2(x) = x^4 + x^3 + x + 1.$$

а. Найти НОД $[p_1(x), p_2(x)]$.
б. Найти НОК $[p_1(x), p_2(x)]$.
в. Найти многочлены $A(x)$ и $B(x)$, удовлетворяющие равенству

$$\text{НОД}\,[p_1(x), p_2(x)] = A(x)\,p_1(x) + B(x)\,p_2(x).$$

5.3. Пусть $m_1(x) = x^3 + x + 1$, $m_2(x) = x^2 + 1$, $m_3(x) = x^2 - 1$. Пусть даны $c_1(x) = c(x) \pmod{m_1(x)}$,

$$c_2(x) = c(x) \pmod{m_2(x)},$$

$$c_3(x) = c(x) \pmod{m_3(x)},$$

где степень многочлена $c(x)$ не превосходит 5. Найти все многочлены, необходимые для вычисления многочлена $c(x)$ по этим вычетам.

5.4. Сколько круговых многочленов делят $x^{15} - 1$? Найти их.

5.5. Пусть $p(x) = \sum_{i=0}^{59} x^i$. Используя соотношение

$$x^{60} - 1 = (x - 1)\,p(x),$$

найти $R_{p(x)}[x^{120} + x^{70} + x^{20}]$.

5.6. Доказать, что для формальной производной многочлена выполняется равенство

$$[r(x)\,s(x)]' = r'(x)\,s(x) + r(x)\,s''(x),$$

и что если $a^2(x)$ делит $r(x)$, то $a(x)$ делит $r'(x)$.

5.7. Доказать, что в кольце $Z/(15)$ целых чисел по модулю 15 многочлен $p(x) = x^2 - 1$ имеет более двух корней. Этот же многочлен над полем имеет только два корня. В каком месте в случае кольца не проходит доказательство теоремы?

5.8. Многочлен $p(x) = x^4 + x^3 + x^2 + x + 1$ неприводим над полем $GF(2)$. Следовательно, кольцо многочленов по модулю $p(x)$ является полем $GF(16)$.
а. Доказать, что в этой конструкции элемент поля, соответствующий многочлену x, не является примитивным.
б. Показать, что многочлену $x + 1$ соответствует примитивный многочлен.
в. Найти минимальный многочлен элемента $x + 1$.

5.9. Над полем $GF(16)$:
а. Сколько имеется различных приведенных многочленов второй степени вида $x^2 + ax + b$, $b \neq 0$?
б. Сколько имеется различных многочленов вида $(x - \beta)(x - \gamma)$, $\beta, \gamma \neq 0$?
в. Доказывает ли это существование неприводимых многочленов второй степени? Сколько простых многочленов второй степени имеется над полем $GF(16)$?

5.10. а. Построить расширение поля вещественных чисел $\mathbb{R}$, используя простой многочлен $x^4 + 1$. Объяснить, почему это расширение является полем комплексных чисел $\mathbb{C}$, которое можно построить также, используя простой многочлен $x^2 + 1$.
б. Построить расширение поля рациональных чисел $\mathbb{Q}$, используя простой многочлен $x^4 + 1$. Объяснить, почему это расширение отличается от расширения поля рациональных чисел с помощью простого многочлена $x^2 + 1$.

5.11. Построить поле $GF(7)$, задавая его таблицами сложения и умножения.

5.12. Вычислить $3^{200} \pmod 7$.

5.13. Доказать, что кольцо вычетов Z_q является кольцом.

5.14. Найти элементы порядка $p^{m-1}(p-1)$ в каждом из следующих полей: $GF(9)$, $GF(25)$, $GF(27)$, $GF(49)$.

5.15. Доказать, что многочлен $x^6 + x^3 + 1$ неприводим над полем рациональных чисел.

Замечания

Материал данной главы обычен для математической литературы. Хорошее изложение теории чисел можно найти у Оре [1] (1948) и Харди и Райта [2] (1960). Свойства полей Галуа описываются во многих книгах по абстрактной алгебре, например, в книгах Биркгофа и Маклейна [3] (1941), а также Ван дер Вардена [4] (1949). В большинстве материала, однако, это стандартное изложение является формальным, уделяет основное внимание абстрактным свойствам и содержит мало примеров и приложений. Большее внимание вычислительным аспектам полей Галуа уделил Берлекэмп [5] (1968). Более детальное изложение колец многочленов и круговых многочленов можно найти в книге Маклеллана и Рейдера [6] (1979).

Глава 6

ВЫЧИСЛЕНИЯ В СУРРОГАТНЫХ ПОЛЯХ

Часто задачу вычисления в заданном поле можно вложить в другое поле, произвести там необходимые вычисления, а затем ответ вернуть в заданное поле. Целесообразность такого перехода может мотивироваться различными причинами. Может оказаться, что в новом поле необходимые вычисления выполнить легче, и это уменьшает объем работ или упрощает реализацию вычислений. Иногда электронные компоненты для вычислений в некотором поле доступнее, чем для заданного поля, и тогда, если требуемые вычисления допускают подходящую переформулировку, резоннее перейти в это поле. Возможны ситуации, когда желательно разработать стандартный вычислительный модуль обработки больших массивов данных и использовать его при решении разнообразных задач для обработки дискретных сигналов. Подгонка одного типа вычислений к другому может осуществляться тогда с целью стандартизации.

Другой важной причиной использования суррогатных полей является улучшение точности вычислений. Вычисления в полях Галуа не содержат погрешностей округления и поэтому всегда точны. Если задачу вычисления в поле вещественных или комплексных чисел удастся погрузить в поле Галуа, то это может привести к увеличению точности результата. В большинстве задач обработки дискретных сигналов можно ограничиться вычислениями с фиксированной запятой; обычно оказывается достаточной 16-битовая арифметика с фиксированной запятой. Если поле Галуа достаточно велико, чтобы содержать 16-битовые целые числа, то в его пределах можно правильно осуществлять многие целочисленные операции, если только результаты не выходят за пределы 16-битовых целых чисел.

6.1. Свертка в суррогатных полях

Свертка в поле вещественных чисел часто может быть заменена сверткой в подходящем поле Галуа. В реальных задачах вещественные числа записываются в виде конечного числа десятичных или двоичных знаков; в цифровой обработке сигналов

обычно используется запись с фиксированной запятой. Двоичное представление с конечным числом знаков может ограничиваться 12 или 16 битами. При вычислениях можно предполагать, что двоичная точка находится справа — все числа являются целыми. Сдвиг двоичной точки во входных данных с целью включения остальных случаев является очень простым делом.

Итак, линейную свертку

$$s(x) = g(x)\, d(x)$$

в поле вещественных чисел мы заменяем сверткой в кольце целых чисел. Для этого нужно только входные данные интерпретировать как целые числа. Внешний вид уравнения, описывающего свертку, не меняется, но операции приобретают смысл стандартной целочисленной арифметики. Затем уравнение в кольце целых чисел можно погрузить в подходящее поле Галуа, скажем, простое поле Галуа, $GF(p)$, если простое число p достаточно велико. Если все входящие в свертку целые числа положительны, то задающее простое поле $GF(p)$ простое число p должно быть столь большим, чтобы выходные коэффициенты свертки не превосходили $p - 1$. Тогда выполняется сравнение

$$s(x) = g(x)\, d(x) \pmod{p},$$

и условие вычислений по модулю p излишне. Если s_i могут принимать и отрицательные значения, то задающее простое поле $GF(p)$ простое число p надо выбирать так, чтобы s_i попадали в диапазон $-(p-1)/2 < s_i \leqslant (p-1)/2$. Тогда отрицательные целые числа могут быть представлены положительными целыми числами в интервале от $(p+1)/2$ до $p-1$.

Свертка в поле Галуа может быть вычислена любым из пригодных в данном поле методов. Некоторые прямые методы рассматриваются в следующем разделе. Можно также воспользоваться преобразованием Фурье в данном поле и теоремой о свертке. На рис. 6.1 приводится сравнение такого вычисления в поле Галуа и в поле комплексных чисел. Мы видим, что на обеих сторонах рисунка выписаны одинаковые уравнения, хотя, конечно, заложенные в них арифметики различны.

Так как циклическая свертка в поле Галуа приводит к тому же самому ответу, что и циклическая свертка в кольце целых чисел, то использование преобразования Фурье данного поля Галуа также приводит к правильному ответу. Так как арифметикой поля $GF(p)$ является арифметика по модулю p, то переполнения по модулю p в промежуточных вычислениях роли не играют, но могут оказаться существенными в выходных вычислениях; если p выбрано достаточно большим, то переполнения в выходных данных невозможны.

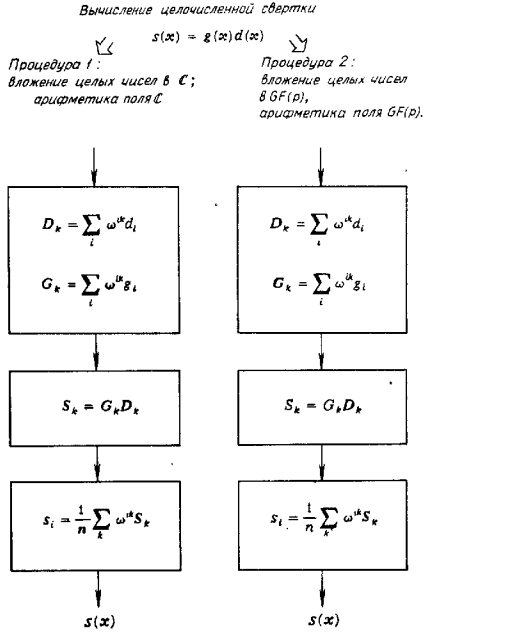


Рис. 6.1. Сравнение двух процедур свертки.

В качестве примера рассмотрим вычисление 5-точечной циклической свертки,

$$s(x) = g(x)\, d(x) \pmod{x^5 - 1}.$$

Выберем простое p равным 31 и будем работать в поле $GF(31)$. Это поле подходит к рассматриваемой задаче постольку, поскольку нам известно, что коэффициенты свертки $s(x)$ не превосходят 30. Для реально возникающих задач поле $GF(31)$ слишком мало и мы его выбрали только для примера.

Так как 5 делит $p - 1 = 30$, то в данном поле существует преобразование Фурье длины 5. Ядром преобразования можно

выбрать элемент 2, так как его порядок равен 5. Пусть

$$d_0 = 2,\ d_1 = 6,\ d_2 = 4,\ d_3 = 4,\ d_4 = 0,$$
$$g_0 = 3,\ g_1 = 0,\ g_2 = 2,\ g_3 = 1,\ g_4 = 2.$$

5-точечное преобразование вектора $\mathbf{d}$ задается равенствами

$$D_k = \sum_{i=0}^{4} 2^{ik} d_i, \quad k = 0, \ldots, 4;$$

для вектора $\mathbf{g}$ имеют место аналогичные соотношения. В матричной записи соответственно имеем

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ D_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 4 & 16 & 2 & 8 \\ 1 & 8 & 2 & 16 & 4 \\ 1 & 16 & 8 & 4 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 6 \\ 4 \\ 4 \\ 0 \end{bmatrix}.$$

Таким образом, компоненты векторов-преобразований даются равенствами

$$D_0 = 16,\ D_1 = 0,\ D_2 = 5,\ D_3 = 29,\ D_4 = 22,$$
$$G_0 = 8,\ G_1 = 20,\ G_2 = 22,\ G_3 = 0,\ G_4 = 27.$$

Перемножая D_i и G_i, для S_i получаем:

$$S_0 = 4,\ S_1 = 0,\ S_2 = 17,\ S_3 = 0,\ S_4 = 5.$$

Так как $5 \cdot 25 = 1$ в поле $GF(31)$, то $5^{-1} = 25$. Следовательно, обратное преобразование Фурье задается равенством

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = 25 \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 16 & 8 & 4 & 2 \\ 1 & 8 & 2 & 16 & 4 \\ 1 & 4 & 16 & 2 & 8 \\ 1 & 2 & 4 & 8 & 16 \end{bmatrix} \begin{bmatrix} 4 \\ 0 \\ 17 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 30 \\ 30 \\ 24 \\ 26 \\ 18 \end{bmatrix},$$

определяющим окончательный результат. Непосредственное вычисление этой свертки в кольце целых чисел дает тот же результат. Если мы выберем большие входные компоненты, то некоторые компоненты свертки окажутся больше 30. В этом случае вычисление свертки в $GF(31)$ приведет к переполнению выходных компонент и даст неправильный результат. Тогда нужно использовать большее поле; в практических приложениях полезны поля, содержащие по меньшей мере 2^{16} элементов.

Если длина слова слишком велика для поля желаемой мощности, то можно воспользоваться китайской теоремой об остатках и работать с вычетами целых чисел, разбивая таким образом длинные слова на короткие подслова. Это использование китайской тео-

ремы об остатках отличается от ее предыдущего использования при построении переупорядочивания линейного потока данных в таблицу. Теперь она используется только для разбиения самих чисел, не затрагивая индексацию данных: данные следуют в своем естественном порядке. Новые свертки, конечно, можно вычислять, используя опять китайскую теорему об остатках, но на сей раз, в отличие от предыдущего, она будет использоваться для вновь сформированных многочленов и их индексов.

Пусть $s_i^{(l)}$ обозначает вычет $s_i \pmod{m_l}$ для r взаимно простых целых чисел $m_1, m_2, \ldots, m_r$. Тогда

$$s_l^{(l)} = \sum_{k=0}^{N-1} g_{i-k}^{(l)} d_k^{(l)} \pmod{m_l}, \qquad \begin{matrix} i = 0, \ldots, L+N-1, \\ l = 0, \ldots, r, \end{matrix}$$

где $g_i^{(l)} = g_i \pmod{m_l}$ и $d_i^{(l)} = d_i \pmod{m_l}$. По китайской теореме об остатках величины s_i восстанавливаются по вычетам $s_i^{(l)}$. Следовательно, вычисление свертки, содержащей большие целые числа, мы свели к вычислению r сверток, содержащих малые целые числа. Для вычисления этих составляющих сверток можно пользоваться уже описанными способами или быстрыми алгоритмами вычисления сверток в кольце целых чисел.

6.2. Числовые преобразования Ферма

Простейший вид алгоритмы вычисления преобразования Фурье имеют в полях Галуа вида $GF(2^m + 1)$, которые действительно являются полями, если $p = 2^m + 1$ — простое число. Известно, что число $2^m + 1$ не является простым, если m не равно степени двух. Однако обратное утверждение неверно, так как число $2^{32} + 1$ является составным. Но при $m = 2, 4, 8$ и 16 число $2^m + 1$ является простым, равным соответственно 5, 17, 257 и 65 537. Это множество целых чисел известно под названием простых чисел Ферма. Если q — простое число Ферма, то в поле $GF(q)$ число $q - 1$ и любой его делитель равны некоторой степени двойки. Преобразование Фурье

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \quad k = 0, \ldots, n-1,$$

определено, если n делит 2^m и существует элемент ω порядка n [1]). Таким образом, в поле $GF(2^{16} + 1)$ определены преобразования Фурье длин $2^{16}, 2^{15}, 2^{14}, \ldots, 2^2, 2$. Используя алгоритм Кули—Тьюки, преобразование Фурье в поле $GF(2^m + 1)$ можно представить в виде последовательности преобразований по основанию два, для реализации которой требуется всего лишь примерно $(n/2)\log_2 n$ умножений и $(n/2)\log_2 n$ сложений.

[1]) Такое преобразование Фурье называют часто *числовым преобразованием Ферма*. — *Прим. перев.*

Преобразования длины 32 и меньше в поле $GF(2^{16} + 1)$ делаются на самом деле намного проще. Это обусловлено тем, что порядок элемента 2 в этом поле равен 32. Чтобы увидеть это, достаточно заметить, что $2^{16} + 1 = 0$ в поле $GF(2^{16} + 1)$. Следовательно, $2^{16} = -1$ и $2^{32} = 1$. Преобразование Фурье записывается в виде

$$V_k = \sum_{i=0}^{31} 2^{ik} v_i, \quad k = 0, \ldots, 31,$$

и умножение в данной арифметической системе сводится просто к циклическому сдвигу, так как представляет собой умножение на степени двойки. Такое преобразование вычисляется чрезвычайно просто, но длина 32 слишком мала для многих приложений.

В общем случае простого числа Ферма $2^m + 1$ порядок элемента 2 в поле $GF(2^m + 1)$ равен $2m$, так что 2 может быть использовано в качестве ядра преобразования длины $2m$. Для построения преобразования Фурье длины $4m$ можно воспользоваться ядром $\sqrt{2}$. В этих полях, как легко проверить возведением в квадрат и использованием соотношения $2^m = -1$, выполняется равенство $\sqrt{2} = 2^{m/4}(2^{m/2} - 1)$. Поэтому каждая степень элемента $\sqrt{2}$ имеет вид $2^a \pm 2^b$.

Например, преобразование Фурье длины 64 в поле $GF(2^{16} + 1)$ записывается в виде

$$V_k = \sum_{i=0}^{63} (2^{12} - 2^4)^{ik} v_i, \qquad k = 0, \ldots, 63.$$

Если для этого вычисления воспользоваться БПФ-алгоритмом Кули—Тьюки по основанию два, то все умножения на константы будут умножениями на числа вида $2^a \pm 2^b$ и, следовательно, могут быть реализованы как пара циклических сдвигов и вычитание. В общем случае поля $GF(2^m + 1)$ преобразование Фурье длины большей чем 2^m всегда содержит нетривиальные умножения. Число этих умножений можно уменьшить сведением БПФ-алгоритма Кули—Тьюки в форму многомерного преобразования, в каждом из измерений которого используется не более чем $2m$-точечное преобразование.

Например, рассмотрим 1024-точечное преобразование в поле $GF(2^{16} + 1)$:

$$V_k = \sum_{i=0}^{1023} \omega^{ik} v_i, \quad k = 0, \ldots, 1023,$$

где ω — элемент порядка 1024, который можно полагать равным π^{64}, где π — примитивный элемент поля. Элемент 3 в поле

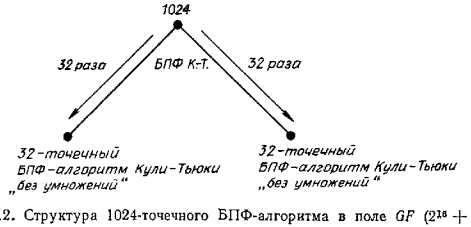


Рис. 6.2. Структура 1024-точечного БПФ-алгоритма в поле $GF\ (2^{16}+1)$.

$GF\ (2^{16}+1)$ является примитивным, но, возможно, такой выбор не является лучшим. Мы хотим ω выбрать так, чтобы выполнялось равенство $\omega^{32}=2$; следовательно, π надо выбрать так, чтобы $\pi^{2048}=2$. БПФ-алгоритм Кули—Тьюки приводит преобразование к виду

$$V_{n''k'+k''}=\sum_{i'=0}^{31}2^{i'k'}\left[\omega^{i'k''}\sum_{i''=0}^{31}2^{i''k''}v_{i'+n'i''}\right].$$

Для каждого значения i' внутренняя сумма является 32-точечным преобразованием Фурье, и для каждого k'' внешняя сумма является 32-точечным преобразованием Фурье. Каждое 32-точечное преобразование Фурье в свою очередь может быть разбито по БПФ-алгоритму Кули—Тьюки по основанию два и вычислено с помощью только циклических сдвигов и сложений. Умножения на $\omega^{i'k''}$ представляют собой нетривиальные умножения, но таких умножений имеется всего 1024, и все они являются целочисленными. На самом деле при i' или k'', равном нулю, эти умножения тривиальны, и остается только 931 нетривиальных умножений. Структура этого БПФ-алгоритма показана на рис. 6.2. В общем случае преобразование Фурье в поле $GF\ (2^{16}+1)$ может быть вычислено с помощью примерно $n\ (\lceil\log_{32}n\rceil-1)$ умножений в поле $GF\ (2^{16}+1)$, $(1/2)\,n\log_2 n$ сложений в этом поле и $(1/2)\,n\log_2 n$ циклических сдвигов.

Арифметикой поля $GF\ (2^{16}+1)$ являются обычные целочисленные сложение и умножение с последующим приведением результата по модулю $2^{16}+1$. Но так как $2^{16}=-1$, то $2^{16+i}=-2^i$, так что биты порядка больше чем 16 переводятся в младшие разряды и вычитаются.

6.3. Числовые преобразования Мерсенна

Полями Галуа, в которых умножение выглядит наиболее естественным образом, являются поля вида $GF\ (2^m-1)$, которые являются на самом деле полем только для простых m, так как

если m — число составное, то $2^{ab}-1$ делится на 2^a-1. Простые числа вида 2^m-1 называются простыми числами Мерсенна. Наименьшие значения m, для которых число 2^m-1 является простым, равны 3, 5, 7, 13, 17, 19 и 31; соответствующие им простые числа Мерсенна равны 7, 31, 127, 8191, 131 071, 524 287 и 2 147 483 647.

Арифметика поля $GF\ (2^m-1)$ очень хорошо согласуется с представлением целых чисел в виде m-битовых чисел, так как в этом поле $2^m=1$. Следовательно, арифметикой поля является обычная целочисленная арифметика, в которой биты переполнения переносятся слагаемыми в соответствующие низшие разряды. Это и есть арифметика 1-дополнений [1]).

Поле $GF\ (2^m-1)$ существует для всех простых чисел 2^m-1. В поле Галуа $GF\ (q)$ преобразование Фурье существует для всех длин n, которые делят число $q-1$. Следовательно, в простом поле $GF\ (2^m-1)$ преобразование Фурье существует для всех длин n, которые делят число 2^m-2. Эти преобразования иногда называют *числовыми преобразованиями Мерсенна*.

Так как число $(2^m-1)-1$ не равно степени двух, то числовые преобразования Мерсенна нельзя вычислять с помощью БПФ-алгоритма Кули—Тьюки по основанию два. Этим они существенно отличаются от числовых преобразований Ферма, для которых БПФ-алгоритм Кули—Тьюки оказался вполне подходящим. Для вычисления числового преобразования Мерсенна можно воспользоваться любым БПФ-алгоритмом по смешанному основанию.

Например, поле $GF\ (2^{13}-1)$ можно рассматривать как поле, элементы которого заданы в виде 13-битовых представлений целых чисел. Свертка двух векторов, компоненты которых заданы 12-битовыми числами, может быть вычислена в этом поле, если только не происходит переполнения выходных коэффициентов свертки. Таким образом, линейная свертка $s(x)=g(x)\,d(x)$, где коэффициентами многочленов $g(x)$ и $d(x)$ служат 12-битовые положительные целые числа (что эквивалентно 13-битовому представлению целых чисел в обратном коде), может быть выражена в виде

$$s(x)=g(x)\,d(x)\pmod{2^{13}-1},$$

если известно, что коэффициенты многочлена $s(x)$ меньше чем $2^{13}-1$.

Для длин n, которые делят число $2^{13}-2$, в поле $GF\ (2^{13}-1)$ существует преобразование Фурье. В данном случае выбор воз-

[1]) Напомним, что речь идет о двоичном представлении целых чисел в обратном коде, в котором взятое с обратным знаком число записывается в виде дополнения его двоичной записи. Например, при $m=3$ и $p=7$ число 2 записывается в виде 010, а число $-2=5$ в виде 101. — *Прим. перев.*

можной длины n преобразования дается разложением $2^{13} - 2 = 2 \cdot 5 \cdot 7 \cdot 9 \cdot 13$. В качестве ядра преобразования Фурье можно выбрать элемент -2, так как $2^{13} - 1 = 0 \pmod{2^{13} - 1}$, и, следовательно, порядок -2 равен 26. Таким образом, преобразование Фурье записывается в виде

$$V_k = \sum_{i=0}^{25} (-2)^{ik} v_i, \quad k = 0, \ldots, 25.$$

В этом преобразовании Фурье все умножения исчерпываются умножениями на степени двух, и, следовательно, вычисление можно реализовать одними циклическими сдвигами, без умножений. С другой стороны, делители числа 26 не очень пригодны в качестве длин БПФ-алгоритмов; на этой конкретной длине трудно придумать что-то лучшее, чем вычислять преобразование Фурье как оно записано посредством 25^2 циклических сдвигов и $26 \cdot 25$ сложений.

Для построения преобразований на других длинах приходится допустить наличие умножений. Алгоритм на большой длине, скажем 70, строится из алгоритмов малых длин, в данном случае 2, 5 и 7, подобно тому, как рассматриваемый в гл. 8 БПФ-алгоритм Винограда большой длины для поля комплексных чисел строится из БПФ-алгоритмов Винограда малых длин. Эти алгоритмы полностью аналогичны, за исключением того, что по-разному определяются константы умножения, связанные с различным определением ядра ω.

В качестве примера построим 5-точечный БПФ-алгоритм Винограда в поле $GF(2^{13} - 1)$. Непосредственной проверкой можно установить, что 3 является примитивным элементом поля $GF(2^{13} - 1)$; следовательно, порядок элемента $3^{2 \cdot 7 \cdot 9 \cdot 13}$ (что равно 4 794) равен 5. Далее, порядок элемента 1904 также равен 5, так как $1904 = (4794)^3$. Тогда 5-точечное преобразование Фурье в поле $GF(2^{13} - 1)$ имеет вид:

$$V_k = \sum_{i=0}^{4} (1904)^{ik} v_i, \qquad k = 0, \ldots, 4.$$

Сначала воспользуемся алгоритмом Рейдера для замены этого вычисления сверткой. Так как алгоритм Рейдера оперирует только с индексами компонент данных, то эта конструкция полностью повторяет конструкцию для поля комплексных чисел. Этим задача сводится к вычислению циклической свертки $s(x) = g(x)\, d(x) \pmod{x^4 - 1}$, в которой многочлен Рейдера равен

$$g(x) = (\omega^3 - 1)\, x^3 + (\omega^4 - 1)\, x^2 + (\omega^2 - 1)\, x + (\omega - 1) =$$

и

$$= 3001x^3 - 1511x^2 + 4793x + 1903$$

$$d(x) = v_2 x^3 + v_4 x^2 + v_3 x + v_1,$$

$$s(x) = (V_3 - V_0)\, x^3 + (V_4 - V_0)\, x^2 + \\ + (V_2 - V_0)\, x + (V_1 - V_0).$$

Теперь воспользуемся быстрым алгоритмом вычисления 4-точечной циклической свертки. Так как разложение многочлена $x^4 - 1$ в поле $GF(2^{13} - 1)$ дается равенством $x^4 - 1 = (x - 1)(x + 1)(x^2 + 1)$, то этот алгоритм циклической свертки имеет тот же самый вид, что и для полей комплексных и вещественных чисел. Это позволяет воспользоваться выписанным на рис. 4.13 алгоритмом, интерпретируя арифметические операции как операции в поле $GF(2^{13} - 1)$. Итак,

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -2 & -2 & 2 \\ 2 & 2 & -2 & -2 \\ 2 & 0 & -2 & 0 \end{bmatrix} \begin{bmatrix} 1903 \\ 4793 \\ -1511 \\ 3001 \end{bmatrix} = \begin{bmatrix} 6142 \\ 2245 \\ 811 \\ 2603 \\ 1707 \end{bmatrix}$$

и 5-точечный БПФ-алгоритм Винограда в поле $GF(2^{13} - 1)$ записывается в виде

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 1 \\ 1 & 1 & -1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & & & & & \\ & 6142 & & & & \\ & & 2245 & & & \\ & & & 811 & & \\ & & & & 2603 & \\ & & & & & 1707 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}.$$

В качестве второго примера рассмотрим поле $GF(2^{17} - 1)$, элементы которого записаны как 17-битовые представления целых чисел в обратном коде. Линейная свертка последовательностей 16-битовых целых чисел может быть вычислена как свертка в поле $GF(2^{17} - 1)$: здесь не происходит переполнения выходных компонент свертки. Для вычисления свертки можно воспользоваться БПФ-алгоритмами в поле $GF(2^{17} - 1)$ и теоремой о свертке.

Длины преобразования Фурье в поле $GF(2^{17} - 1)$ исчерпываются делителями числа $2^{17} - 2$, которые полностью определяются разложением $2^{17} - 2 = 2 \cdot 3 \cdot 5 \cdot 17 \cdot 257$. Можно выбрать, например, $n = 510$, и строить 510-точечное БПФ-преобразование из 2-точечного, 3-точечного, 5-точечного и 17-точечного БПФ-алгоритмов. Модули для 2-точечного, 3-точечного и 5-точечного преобразований очень просты и даются малыми БПФ-алгоритмами Винограда. Хотя эти малые алгоритмы рассматриваются над полем $GF(2^{17} - 1)$, записываются они точно так же, как и в случае поля комплексных чисел.

17-точечное преобразование тоже представляет собой простой модуль, но строится он другим способом. А именно, элемент 2

поля $GF(2^{17}-1)$ имеет порядок 17, так как $2^{17}=1$. Следовательно, 17-точечное преобразование в этом поле задается равенствами

$$V_k = \sum_{l=0}^{16} 2^{lk} v_l, \qquad k = 0, \ldots, 16,$$

и может быть вычислено с помощью одних циклических сдвигов и сложений. Умножения не нужны.

Если необходимо вычислить свертку, длина которой больше 510, то надо использовать также множитель 257. 257-точечное преобразование Фурье с помощью алгоритма Рейдера сводится к 256-точечной циклической свертке, которая затем вычисляется с помощью БПФ-алгоритма Кули—Тьюки по основанию два и алгоритма свертки.

6.4. Алгоритмы свертки в конечных полях

В полях Галуа вместо того, чтобы пользоваться преобразованием Фурье и теоремой о свертке, можно строить прямые методы вычисления свертки. Для построения алгоритмов свертки применимы все описанные в гл. 3 методы. Другой путь построения прямых алгоритмов свертки состоит в модификации уже построенных алгоритмов свертки для поля вещественных чисел.

Для преобразования алгоритма свертки, построенного для поля вещественных чисел, в алгоритм свертки в поле Галуа $GF(q)$ характеристики p начнем с алгоритма свертки, записанного в виде

$$\mathbf{s} = \mathbf{C}[(\mathbf{Ag})\cdot(\mathbf{Bd})],$$

где $\mathbf{A}$, $\mathbf{B}$ и $\mathbf{C}$ — матрицы с рациональными элементами. Умножим обе части равенства на наименьшее целое число L, такое, чтобы ликвидировать все знаменатели во всех компонентах; тогда равенство перепишется в виде

$$L\mathbf{s} = \mathbf{C}'[(\mathbf{A}'\mathbf{g})\cdot(\mathbf{B}'\mathbf{d})],$$

где L — целое число и $\mathbf{A}'$, $\mathbf{B}'$, $\mathbf{C}'$ — целочисленные матрицы. Это равенство можно рассматривать как алгоритм вычисления целочисленной свертки; следовательно, его можно рассматривать и как уравнение по модулю p:

$$L\mathbf{s} = \mathbf{C}'[(\mathbf{A}'\mathbf{g})\cdot(\mathbf{B}'\mathbf{d})] \pmod p.$$

Если $L \not\equiv 0 \pmod p$, то, разделив обе части равенства на L по модулю p, получаем алгоритм свертки в $GF(p)$, и, более того, алгоритм в любом расширении поля $GF(p)$.

Если L сравнимо с нулем по модулю p, то алгоритм свертки для поля вещественных чисел не переносится в поле Галуа, и надо строить такой алгоритм прямо в самом поле Галуа. Эта задача возникает даже тогда, когда L не равно нулю по модулю p, так как может оказаться, что алгоритм свертки, построенный специально для данного поля Галуа, лучше алгоритма, перенесенного из другого поля. Для построения такого алгоритма можно пользоваться разработанными в гл. 3 методами; конечно, при этом требуется, чтобы разложение

$$m(x) = m^{(0)}(x) \ldots m^{(k)}(x)$$

было разложением в том поле, в котором вычисляется свертка.

В большей части данной главы мы будем интересоваться полями, характеристика p которых велика. Так как для таких полей L будет намного меньше p, то все алгоритмы свертки для поля рациональных чисел могут быть перенесены в $GF(p)$ и будут выглядеть точно так же. В оставшейся части этого раздела мы займемся противоположным случаем — случаем полей малой характеристики, или, еще конкретнее, случаем полей характеристики два. Поля Галуа характеристики два играют важную роль в нескольких приложениях.

Начнем с построения алгоритма 3-точечной циклической свертки для полей характеристики два. В поле бесконечной характеристики этот алгоритм записывается уравнением

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 2 \\ 1 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{3}(g_0+g_1+g_2) & & & \\ & (g_0-g_2) & & \\ & & (g_1-g_2) & \\ & & & \frac{1}{3}(g_0+g_1-2g_2) \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}.$$

Ни один из знаменателей не кратен двум. Следовательно, алгоритм может быть перенесен в поля характеристики 2. Арифметикой целых чисел в полях характеристики два является арифметика по модулю два, так что все целые числа равны нулю или единице. Соответственно новый алгоритм записывается в виде

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} (g_0+g_1+g_2) & & & \\ & (g_0+g_2) & & \\ & & (g_1+g_2) & \\ & & & (g_0+g_1) \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}.$$

С другой стороны, алгоритм 2-точечной свертки содержит элементы с четным знаменателем и, следовательно, не может быть перенесен в поля характеристики два, так как знаменатели обратятся в нуль. Лучший алгоритм 2-точечной циклической свертки в полях характеристики два задается равенством

$$\begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_1 & \\ & & g_0+g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix},$$

и содержит три умножения. Причина необходимости трех умножений кроется в том, что над полем GF (2) многочлен $x^2 - 1$ не разлагается в произведение двух взаимно простых множителей, так как в полях характеристики два $-1 = 1$, и, следовательно, $x^2 + 1 = (x + 1)^2$; следовательно, в конструкцию алгоритма в качестве модуля входит многочлен $x^2 + 1$ степени два.

Такая ситуация, когда число умножений в конечном поле больше числа умножений в рациональном поле, встречается для циклических сверток многих длин. Это происходит потому, что несколько простых делителей многочлена $x^n - 1$ оказываются равными. С другой стороны, некоторые циклические свертки в полях конечной характеристики требуют меньше умножений, чем свертки тех же длин в рациональном поле. Это происходит потому, что круговые многочлены в конечных полях могут распадаться в произведение простых множителей. Например, над полем рациональных чисел разложение на неприводимые множители многочлена $x^7 - 1$ имеет вид

$$x^7 - 1 = (x - 1)(x^6 + x^5 + x^4 + x^3 + x^2 + x + 1),$$

а над полем характеристики два это разложение можно продолжить:

$$x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Следовательно, согласно выписанной в разд. 3.8 общей границе, оптимальный (относительно числа умножений) алгоритм 7-точечной циклической свертки над полем рациональных чисел должен содержать 12 умножений, а оптимальный алгоритм 7-точечной циклической свертки над полем характеристики два должен содержать 11 умножений. Для первого случая известен хороший с практической точки зрения алгоритм, содержащий 16 умножений, а для второго — хороший алгоритм с 13 умножениями.

Последний алгоритм строится методами главы 3 и задается равенством

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{bmatrix} = \begin{bmatrix} 1&1&1&1&1&1&1 \\ 0&1&0&0&1&1&1 \\ 1&0&0&1&1&1&0 \\ 1&1&0&1&0&0&1 \\ 1&0&1&0&1&0&0 \\ 0&0&1&1&1&0&1 \\ 0&1&1&1&0&0&1 \\ 0&1&0&1&1&1&0 \\ 0&0&1&0&1&1&1 \\ 1&1&0&0&1&0&1 \\ 1&0&0&1&0&1&1 \\ 1&1&1&0&0&1&0 \end{bmatrix} \left\{ \begin{bmatrix} 1&1&1&1&1&1&1 \\ 0&1&1&1&0&0&1 \\ 1&0&1&1&1&0&0 \\ 1&1&0&0&1&0&1 \\ 1&0&0&1&0&1&1 \\ 0&1&0&1&1&1&0 \\ 0&0&1&0&1&1&1 \\ 1&1&0&1&0&0&1 \\ 1&0&1&0&0&1&1 \\ 1&0&0&1&1&1&0 \\ 0&1&0&0&1&1&1 \\ 0&0&1&1&1&0&1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \end{bmatrix} \cdot \begin{bmatrix} 1&1&1&1&1&1&1 \\ 1&0&1&1&1&0&0 \\ 0&1&0&1&1&1&0 \\ 0&0&1&0&1&1&1 \\ 1&1&1&0&0&1&0 \\ 1&1&0&0&1&0&1 \\ 1&0&0&1&0&1&1 \\ 0&1&1&0&1&0&0 \\ 1&0&0&1&1&1&0 \\ 0&0&1&1&1&0&1 \\ 1&1&0&1&0&0&0 \\ 1&0&1&0&0&0&1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \right\}.$$

Такого сорта улучшения можно найти и в полях большей характеристики. Например, в поле GF (11) разложение на простые множители того же многочлена $x^7 - 1$ имеет вид

$$x^7 - 1 = (x - 1)(x^3 + 5x^2 + 4x - 1)(x^3 - 4x^2 - 5x - 1).$$

Следовательно, в полях характеристики одиннадцать алгоритм вычисления 7-точечной циклической свертки содержит 11 умножений. Таким алгоритмом является

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{bmatrix} = \begin{bmatrix} 1&4&-1&0&-4&1&-5&3&0&2&1 \\ 1&-1&2&4&2&4&-1&4&-5&-1&-5 \\ 1&-5&4&-5&0&-1&4&2&4&-2&-1 \\ 1&1&3&0&-5&-5&1&-1&0&5&4 \\ 1&0&1&1&4&1&0&1&1&4&1 \\ 1&0&1&-1&2&0&0&1&-1&2&0 \\ 1&1&1&1&1&0&1&1&1&1&0 \end{bmatrix} \left\{ \begin{bmatrix} 1&1&1&1&1&1&1 \\ 1&0&0&1&-5&-1&4 \\ 1&1&1&3&4&2&-1 \\ 1&-1&1&0&-5&4&0 \\ 1&2&4&-5&0&2&-4 \\ 0&0&1&-5&-1&4&1 \\ 1&0&0&1&4&-1&-5 \\ 1&1&1&-1&2&4&3 \\ 1&-1&1&0&4&-5&0 \\ 1&2&4&5&-2&-1&2 \\ 0&0&1&4&-1&-5&1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \end{bmatrix} \cdot \begin{bmatrix} -3&-3&-3&-3&-3&-3&-3 \\ 5&0&3&1&5&-4&1 \\ -4&-4&4&3&-2&2&1 \\ 3&-3&2&5&-3&-3&-1 \\ 0&0&1&-5&-1&4&1 \\ 0&-5&2&-1&3&2&-1 \\ -5&0&-3&5&5&-2&0 \\ -4&-4&-1&5&0&2&2 \\ -3&3&-2&4&-2&-1&1 \\ 0&0&-1&-4&1&5&-1 \\ 0&5&-1&-1&-4&0&1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \right\}.$$

В алгоритм входит только 11 умножений общего вида, но умножений на малые фиксированные константы (± 2, ± 3, ± 4, ± 5) он содержит достаточно много. Каждое из таких умножений можно заменить несколькими сложениями, но тогда мы получим большое число сложений. Возможности улучшения этого алгоритма и являются открытой задачей. Ее решение зависит от стоимости сложений и умножений. В достаточно больших расширениях поля GF (11^m) умножения обходятся существенно дороже сложений, так что такой алгоритм может оказаться вполне пригодным. В простом поле GF (11), по-видимому, замена умножений сложениями не дает преимуществ.

6.5. Комплексная свертка в суррогатных полях

В последних двух разделах мы видели, как свертку в вещественном поле можно вложить в поле Галуа, где ее удобнее вычислять. Теперь мы займемся этой же задачей для свертки комплексных чисел. В зависимости от наличия в поле $GF(p)$ элемента $\sqrt{-1}$ здесь имеются два различных случая, которые приводят к совершенно разным действиям. Мы рассмотрим только два класса простых чисел: простые числа Мерсенна и простые числа Ферма. В случае простых чисел Мерсенна поле $GF(p)$ не содержит элемента $\sqrt{-1}$; в случае простых чисел Ферма элемент $\sqrt{-1}$ принадлежит полю $GF(p)$. Мы будем рассматривать лишь эти два класса простых чисел, но к любому другому простому числу можно применить один из этих методов.

Мы начнем с простых чисел Мерсенна, $p = 2^m - 1$, где m — нечетное простое число. Это поле не содержит элемента $\sqrt{-1}$. Расширим поле $GF(2^m - 1)$ до поля $GF((2^m - 1)^2)$ подобно тому, как поле вещественных чисел $\mathbb{R}$ расширяется до поля комплексных чисел $\mathbb{C}$. Тогда преобразование Фурье в поле $GF((2^m - 1)^2)$ можно использовать для вычисления свертки в поле комплексных чисел.

В поле вещественных чисел многочлен $x^2 + 1$ не имеет корней. Обозначим через j некоторый новый элемент и присоединим его к полю вещественных чисел, образуя множество $\mathbb{C} = \{a + bj\}$, где a и b — вещественные числа, а сложение и умножение задаются правилами

$$(a + bj) + (c + dj) = (a + c) + (b + d)j,$$

$$(a + bj)(c + dj) = (ac - bd) + (ad + bc)j.$$

Легко проверить, что это множество образует поле.

Аналогично, в поле Галуа $GF(2^m - 1)$ многочлен $x^2 + 1$ не имеет корней. Расширим это поле, присоединив к нему некоторый обозначаемый через j элемент и сформировав множество $GF((2^m - 1)^2) = \{a + bj\}$, где a и b — произвольные элементы поля $GF(2^m - 1)$. Сложение и умножение опять зададим правилами

$$(a + bj) + (c + dj) = (a + c) + (b + d)j,$$

$$(a + bj)(c + dj) = (ac - bd) + (ad + bc)j,$$

где операции справа являются операциями в исходном поле $GF(2^m - 1)$. Легко проверить, что такое определение задает поле, содержащее $(2^m - 1)^2$ элементов.

Подытожим сказанное в виде двух теорем.

Теорема 6.5.1. *Если $2^m - 1$ — простое число Мерсенна, то поле $GF(2^m - 1)$ не содержит среди своих элементов квадратного корня из -1; следовательно, многочлен $x^2 + 1$ не имеет в этом поле корней.*

Доказательство отложим до конца раздела.

Теорема 6.5.2. *Относительно определенных выше операций сложения и умножения множество $GF((2^m - 1)^2)$ образует поле.*

Доказательство. Утверждение вытекает непосредственно из того, что многочлен $x^2 + 1$ прост. □

Теперь рассмотрим преобразование Фурье в поле $GF((2^m - 1)^2)$. Длина преобразования равна $(2^m - 1)^2 - 1$ или делит это число. Так как имеет место разложение $(2^m - 1)^2 - 1 = 2^{m+1}(2^{m-1} - 1)$, то возможной длиной преобразования Фурье в поле $GF((2^m - 1)^2)$ является 2^{m+1}, и тогда это преобразование можно вычислять с помощью БПФ-алгоритма Кули—Тьюки по основанию два. Другие возможные длины преобразования Фурье задаются делителями числа $2^{m-1} - 1$.

Выберем, например, $m = 17$. Тогда $(2^{17} - 1)^2 - 1 = 2^{18} \times 3 \cdot 5 \cdot 17 \cdot 257$. В качестве длины преобразования можно взять любую степень двойки вплоть до 2^{18}, а большие длины получаются присоединением остальных делителей.

Пусть n делит 2^{18} и пусть

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \qquad k = 0, \ldots, n-1,$$

где ω — элемент поля $GF((2^{17} - 1)^2)$ порядка n. (В табл. 6.1 приведены найденные на ЭВМ соответствующие элементы ω.) Так как n равно некоторой степени двух, то для вычисления преобразования Фурье можно воспользоваться БПФ-алгоритмом Кули—Тьюки по основанию два.

Возможности выбора длины преобразования Фурье в поле $GF((2^{17} - 1)^2)$ существенно шире, чем в поле $GF(2^{17} - 1)$; в частности, это относится к длинам, равным степени двойки. Поэтому для вычисления циклической свертки в $GF(2^{17} - 1)$ можно перейти в поле $GF((2^{17} - 1)^2)$, рассматривая исходную свертку как свертку целых чисел поля $GF((2^{17} - 1)^2)$.

В случае, когда p является простым числом Ферма, $\sqrt{-1}$ является элементом простого поля, и построение расширения $GF(p^2)$ с операцией умножения, аналогичной умножению в поле комплексных чисел, невозможно. Действительно, для $p = 2^m + 1$, где m равно степени двух, возведением в квадрат легко убедиться, что

Таблица 6.1

n	$GF(2^{2^1}-1)^2$	$GF(2^{2^2}-1)^2$
256	$(2^7\cdot 2^4-2^3)+j(2^{11}+2^7-2^4+2^3)$	$(2^8+2^5)+j(2^{15}+2^{11}+2^7-2^5+2^4+2^3)$
512	$(2^{11}\cdot 2^7-2^4+2^3)+j(2^8-2^3)$	$(2^{14}+2^9)+j(2^{21}-2^{11}+2^8+2^5+2^3)$
1024	$(2^{12}-2^8+2^3)+j(2^9-2)$	$(2^{11}+2^8-2^3)+j(2^{15}+2^{14}+2^7-2^3)$
2048	$(2^{16}+2^7+2^4+2)+j(2^{10}+2^3)$	$(2^7+2^4)+j(2^{13}+2^{11}-2^8+2^7+2^4-2^3)$
4096	$(2^{11}+2^7+2^4+2^3)+j(2^8+2)$	$(2^{10}-2^7+2^3)+j(2^{14}-2^3)$

n	$GF(2^{2^3}-1)^2$	$GF(2^{2^4}-1)^2$
256	$(2^{15}+2^{14}-2^7+2^3)$ $+j(2^{15}+2^{13}+2^{11}+2^7+2^4+2^3-2^0)$	$(2^{27}+2^{25}+2^{21}+2^{19}+2^{16}-2^{11}-2^{10}+2^4-2^3)$ $+j(2^{27}+2^{21}-2^{19}-2^{17}+2^{14}+2^{13}+2^8+2^3)$
512	$(2^{17}+2^{15}+2^{11}+2^7-2^4+2^3)$ $+j(2^{14}-2^{10}-2^3)$	$(2^{30}-2^{28}+2^{25}+2^{23}+2^{18}-2^{15}+2^{13}+2^{10}+2^3)$ $+j(2^{30}-2^{28}+2^{25}+2^{24}-2^{18}+2^8+2^3)$
1024	$(2^{12}+2^{10}-2^7)$ $+j(2^{18}-2^{14}+2^{13}-2^7-2^3)$	$(2^{31}-2^{28}+2^{25}-2^{18}-2^{13}+2^{10}+2^3)$ $+j(2^{30}-2^{25}-2^{19}+2^{16}-2^{13}-2^7+2^4-2^3)$
2048	$(2^{19}-2^{17}+2^{15})$ $+j(2^{16}+2^{11}-2^8+2^4+2^3)$	$(2^{28}+2^{24}+2^{18}-2^{16}+2^{14}-2^4+2^3)$ $+j(2^{27}-2^{24}+2^{17}-2^{14}+2^{10}+2^{18}-2^3)$
4096	$(2^{11}+2^8-2^4+2^3)$ $+j(2^{11}+2^4+2^3+2^3)$	$(2^{25}+2^{21}+2^{17}-2^{16}+2^{11}+2^3)$ $+j(2^{30}+2^{24}-2^{23}+2^{14}+2^8+2^3)$

$\sqrt{-1}=2^{m/4}(2^{m/2}-1)$. Для вычисления свертки $s(x)=g(x)\,d(x)$, где

$$g(x)=g_R(x)+jg_I(x) \text{ и } d(x)=d_R(x)+jd_I(x),$$

приходится вычислять четыре свертки $g_R(x)\,d_R(x)$, $g_R(x)\,d_I(x)$, $g_I(x)\,d_R(x)$ и $g_I(x)\,d_I(x)$ и пользоваться равенствами

$$s_R(x)=g_R(x)\,d_R(x)-g_I(x)\,d_I(x),$$
$$s_I(x)=g_R(x)\,d_I(x)+g_I(x)\,d_R(x).$$

Лучшая процедура, содержащая вдвое меньше умножений, основана на определении в кольце $GF(p)[x]/(x^n-1)$ многочленов

$$a(x)=\frac{1}{2}\left(g_R(x)-2^{m/2}g_I(x)\right)\left(d_R(x)-2^{m/2}d_I(x)\right),$$

$$b(x)=\frac{1}{2}\left(g_R(x)+2^{m/2}g_I(x)\right)\left(d_R(x)+2^{m/2}d_I(x)\right),$$

для вычисления которых надо сделать только две свертки. Все вычисления проводятся в кольце $GF(p)[x]/(x^n-1)$, и свертка $s(x)$ вычисляется по правилу

$$s_R(x)=(a(x)+b(x)),$$
$$s_I(x)=2^{m/2}(a(x)-b(x)).$$

Для завершения раздела нам осталось провести доказательство теоремы 6.5.1. Оно достаточно длинно и опирается на понятие квадратичного вычета. Те элементы простого поля $GF(p)$, для которых в этом поле существуют квадратные корни, называются *квадратичными вычетами* (поскольку по модулю p они равны квадратам своих квадратных корней). Если p нечетно, то ровно половина элементов поля $GF(p)$ имеют квадратные корни. Чтобы это показать, заметим сначала, что каждая четная степень примитивного элемента α имеет квадратный корень. С другой стороны, каждый элемент, равный квадратному корню, может быть записан в виде α^i для некоторого i, так что его квадрат равен $\alpha^{((2i))}$, где двойные скобки в показателе степени использованы для обозначения того, что вычисления проводятся по модулю $p-1$, так как мультипликативная группа поля является циклической порядка $p-1$. Но число $p-1$ четно, и, следовательно, $((2i))$ также четно. Таким образом, только четные степени элемента α могут иметь квадратный корень.

Теорема 6.5.3. *Для нечетных p в поле $GF(p)$ элемент r является квадратичным вычетом тогда и только тогда, когда* $r^{(p-1)/2}=1$.

Доказательство. Предположим, что $r^{(p-1)/2}\neq 1$. Тогда $\sqrt{r}$ не может существовать в данном поле, так как в противном случае

должно бы было выполняться равенство $(\sqrt{r})^{p-1} = 1$, которое не выполняется.

Предположим, что $r^{(p-1)/2} = 1$, и пусть α — примитивный элемент поля $GF(p)$. Очевидно, что все четные степени элемента α являются квадратичными вычетами, а все нечетные степени элемента α квадратичными вычетами не являются. Надо только показать, что r равно четной степени элемента α. Допустим противное: $r = \alpha^{2i+1}$; тогда, так как порядок элемента α равен $p-1$, имеем

$$r^{(p-1)/2} = \left(\alpha^{2i+1}\right)^{(p-1)/2} = \alpha^{i(p-1)}\alpha^{(p-1)/2} = \alpha^{(p-1)/2} \neq 1.$$

Таким образом, из равенства $r^{(p-1)/2} = 1$ вытекает, что r равно четной степени элемента α и, следовательно, является квадратичным вычетом. □

Теперь у нас уже все готово для доказательства теоремы 6.5.1.

Теорема 6.5.1. *Элемент* -1 *поля* $GF(2^m - 1)$, *где* $2^m - 1$ — *простое число Мерсенна, не имеет в этом поле квадратного корня. Следовательно, многочлен* $x^2 + 1$ *не имеет в этом поле корней.*

Доказательство. Предположим, что -1 имеет квадратный корень, равный r. Тогда $r^2 = -1$ и, согласно теореме 6.5.3, $r^{(p-1)/2} = 1$. Так как $p = 2^m - 1$, то $r^{(2^m-2)/2} = 1$ или $r^{2^{m-1}-1} = 1$. Но $r^2 = -1$ и $m - 1$ четно. Следовательно, $r^{2^{m-1}} = 1$ и $r^{-1} = 1$. Но тогда r^2 не равно -1. Полученное противоречие показывает, что в данном поле не существует квадратного корня из -1. □

6.6. Преобразования в числовом кольце

Если поле F содержит элемент порядка n, то в этом поле существует преобразование Фурье длины n. Если преобразование Фурье существует, то оно обладает всеми элементарными свойствами такого преобразования.

Часто возможно также определить преобразование в кольце, но при этом ситуация не является столь простой. В настоящем разделе рассматриваются преобразования в кольце $Z/(q)$ целых чисел по модулю q. Если q — простое число, то $Z/(q)$ является полем, и, как мы уже видели, в нем существует преобразование Фурье со всеми его основными свойствами. Поэтому надо рассмотреть только случай составного числа q. Как мы увидим, даже для составных q можно дать осмысленное определение преобразования Фурье. Однако структура этих преобразований представляет собой точную копию преобразований Фурье для простых делителей числа q, так что переход к составному q прибавляет мало возможностей.

Мы хотим определить в целочисленном кольце преобразование

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \qquad k = 0, \ldots, n-1,$$

вектора $\mathbf{v}$ над кольцом $Z/(q)$ так, чтобы существовало обратное преобразование

$$v_l = n^{-1} \sum_{k=0}^{n-1} \omega^{-ik} V_k, \qquad i = 0, \ldots, n-1,$$

и была справедлива теорема о свертке. Для того, чтобы такое определение преобразования Фурье было возможным, необходимо выполнение двух условий: (1) должен существовать элемент ω порядка n; (2) элемент n в кольце $Z/(q)$ должен быть обратимым. Нужна, конечно, еще и обратимость элемента ω, но из условия $\omega^n = 1$ автоматически следует, что $\omega^{-1} = \omega^{n-1}$.

Итак, надо определить те значения n, для которых существует элемент ω порядка n. Начнем с простейшего случая q, равного степени простого нечетного числа p, $q = p^m$. Кольцо $Z/(p^m)$ не образует поля, так как его умножением является умножение по модулю p^m. Согласно теореме 5.1.8, в этом кольце имеются элементы порядка $(p-1)p^{m-1}$, и согласно теореме 5.1.5 (теореме Эйлера), порядок каждого взаимно простого p^m элемента делит $(p-1)p^{m-1}$. Согласно этому условию, для каждого делителя n числа $(p-1)p^{m-1}$ можно выбрать элемент ω порядка n. Однако теорема 5.2.2 утверждает, что число n обратимо тогда и только тогда, когда n и p^m взаимно просты. Следовательно, n не может делиться на p. Таким образом, в качестве ω можно выбрать только элементы, порядок которых n делит $p-1$. Следующая теорема показывает, что для каждого такого n существует соответствующее преобразование Фурье. Однако длины такого преобразования в $Z/(p^m)$ в точности совпадают с теми длинами, которые допустимы для преобразования Фурье в поле $GF(p)$. Изменения касаются только разрядности используемых чисел, которая увеличивается примерно с $\log_2 p$ до $m \log_2 p$. Но обычно разрядность является достаточно большой и для преобразований в поле $GF(p)$, что еще больше уменьшает преимущества перехода к $Z/(p^m)$.

Ситуация для произвольного q аналогична и описывается следующей теоремой.

Теорема 6.6.1. *Обратимое преобразование Фурье длины* n *над кольцом* $Z/(q)$ *существует тогда и только тогда, когда* n *делит* $p-1$ *для всех простых делителей* p *числа* q.

Доказательство. Сначала мы приведем доказательство для случая, когда q равно степени простого числа. Затем используем

китайскую теорему об остатках, чтобы связать этот случай со случаем произвольного числа q.

Проще всего доказывается обратная часть теоремы. Длина n преобразования обратима, только если n и q взаимно просты, так как

$$nn^{-1} = 1 + Qq,$$

и, следовательно, любой делитель чисел n и q должен быть и делителем единицы, так что он должен равняться единице. Далее, теорема 5.1.5 утверждает, что если порядок n элемента ω взаимно прост с q, то он делит $\varphi(q) = (p-1)\,p^{m-1}$. Следовательно, преобразование Фурье в кольце $Z/(p^m)$ существует только на длинах n, которые делят $p-1$.

Теперь докажем прямое утверждение теоремы. Полагать p равным 2 смысла нет, так как тогда n равно 1 и утверждение тривиально. Следовательно, p — нечетное простое число и согласно теореме 5.1.8 существует элемент π порядка $(p-1)\,p^{m-1}$. Для любого делителя b числа $p-1$ положим $\omega = \pi^{bp^{m-1}}$; тогда порядок элемента ω равен $(p-1)/b$. Таким образом, для любого делителя b числа $p-1$ имеется элемент ω этого порядка, и остается доказать существование обратного преобразования Фурье. Для этого запишем

$$\frac{1}{n}\sum_{k=0}^{n-1}\omega^{-tk}V_k = \frac{1}{n}\sum_{k=0}^{n-1}\omega^{-tk}\sum_{t'=0}^{n-1}\omega^{t'k}v_{t'} =$$

$$= \frac{1}{n}\sum_{t'=0}^{n-1}v_{t'}\left[\sum_{k=0}^{n-1}\omega^{-k(t'-t)}\right].$$

Если $i' = i$, то сумма по k равна n. Для $i \neq i'$ имеем

$$\sum_{k=0}^{n-1}\left(\omega^{-(t'-t)}\right)^k = \frac{1-\omega^{-(t'-t)n}}{1-\omega^{-(t'-t)}}.$$

Так как оба индекса i и i' меньше n и $i'-i \neq 0 \pmod n$, то выражение справа равно нулю. Таким образом,

$$\frac{1}{n}\sum_{k=0}^{n-1}\omega^{-tk}V_k = \frac{1}{n}\sum_{t'=0}^{n-1}v_{t'}(n\delta_{tt'}) = v_t,$$

что и требовалось доказать.

Теперь предположим, что $q = p_1^{m_1}p_2^{m_2}\ldots p_r^{m_r}$. Воспользуемся китайской теоремой об остатках и алгоритмом Гуда—Томаса для

отображения кольца $Z/(q)$ в прямое произведение $Z/(p_1^{m_1}) \times \times Z/(p_2^{m_2}) \times \ldots \times Z/(p_r^{m_r})$. Условия существования преобразования Фурье в кольце $Z/(q)$ связаны с условиями существования преобразований Фурье в каждом из $Z/(p_i^{m_i})$, так что условие делимости $p_i - 1$ на n должно выполняться для всех i. □

По-видимому, описываемое теоремой 6.6.1 преобразование Фурье в целочисленном кольце $Z/(q)$ при составном q может иметь весьма ограниченные приложения. Тем не менее они могут в некоторых специальных случаях оказаться весьма подходящими.

Рассмотрим, например, разложение

$$2^{40} + 1 = (257)\,(4278255361) = p_1p_2.$$

В кольце $Z/(2^{40}+1)$ существует преобразование Фурье длины 256, так как 256 делит и $p_1 - 1$ и $p_2 - 1$. Это преобразование при разрядности чисел в 41 бит обеспечивает точность вычислений в 40 бит. Кольцо допускает БПФ-алгоритм Кули—Тьюки по основанию два. Арифметика переполнения легко учитывается равенством $2^{40} = -1$. Элемент 2 нельзя в этом случае использовать в качестве ядра преобразования, так как порядок 2 равен 80, а порядок ядра должен быть равен 256. Следовательно, умножения не могут быть реализованы в виде циклических сдвигов, а являются умножением 40-битовых чисел общего вида. Таким образом, такая конструкция приводит к процедуре 256-точечной циклической свертки, содержащей примерно $256 + 256\log_2 256$ умножений общего типа для 40-разрядных двоичных чисел с точностью в 40 битовых разрядах. По сравнению с комплексным БПФ-алгоритмом для 40-разрядных двоичных чисел эта процедура является и более точной и содержит меньшее число умножений.

6.7. Числовые преобразования Шевилла

Числа Шевилла (как правило, простые, но не всегда) в нестрогом определении задаются как числа q, для которых в кольце $Z/(q)$ существует преобразование Фурье по одному основанию для больших длин преобразования. Числа Шевилла связаны только с хорошими преобразованиями Фурье и не имеют специального теоретико-числового значения. Таблица этих чисел, построенных с помощью ЭВМ, приведена на рис. 6.3.

Если элемент ω кольца $Z/(q)$ имеет порядок, равный степени малого простого числа, то преобразование Фурье

$$V_k = \sum_{i=0}^{n-1}\omega^{ik}v_k, \qquad k = 0, \ldots, n-1,$$

в этом кольце удобно вычислять с помощью БПФ-алгоритма Кули—Тьюки по одному основанию. В приведенной на рис. 6.3 таблице для каждого q указан порядок, определяющий длину преобразования n, равную степени малого простого числа.

Длина слова	q	Эффективная длина слова	Максимальная длина преобразования	Максимальная длина БПФ по одному основанию
8	163	7.3	162	81
	193	7.6	192	64
	197	7.6	196	49
	241	7.9	240	16
	251	8.0	250	125
9	257	8.0	256	256
	401	8.6	400	25
	449	8.8	448	64
	487	8.9	486	243
	491	8.9	490	49
10	751	9.6	750	125
	769	9.6	768	256
	883	9.8	882	49
	919	9.8	918	27
	929	9.9	928	32
	1009	10.0	1008	16
11	1373	10.4	1372	343
	1409	10.5	1408	128
	1459	10.5	1458	729
	1471	10.5	1470	49
	1601	10.6	1600	64
	1783	10.8	1782	81
	1951	10.9	1950	25
	1999	11.0	1998	27
	2017	11.0	2016	32
12	2917	11.5	2916	729
	3329	11.7	3328	256
	3457	11.8	3456	128
	3889	11.9	3888	243
	4001	12.0	4000	32, 125
	4019	12.0	4018	49
	4049	12.0	4048	16
	4051	12.0	4050	81, 25
	5347	12.4	5346	243
	7001	12.8	7000	125
	7547	12.9	7546	343
	7681	12.9	7680	512
	7841	12.9	7840	49
	7937	13.0	7936	256
	8101	13.0	8100	81, 25
	8161	13.0	8160	32

Рис. 6.3. Таблица алгоритмов Шевилла для преобразования

Длина слова	q	Эффективная длина слова	Максимальная длина преобразования	Максимальная длина БПФ по одному основанию
14	8263	13.0	8262	243
	13751	13.7	13750	625
	14407	13.8	14406	2401
	15361	13.9	15360	1024
	16001	14.0	16000	128, 125
	16073	14.0	16072	49
	16193	14.0	16192	64
	16301	14.0	16300	25
	16363	14.0	16362	81
	16369	14.0	16368	16
15	17497	14.1	17496	2187
	18433	14.2	18432	2048
	25601	14.6	25600	1024
	28751	14.8	28750	625
	28813	14.8	28812	2401
	30871	14.9	30870	343
	32077	15.0	32076	729
	32251	15.0	32250	125
	32257	15.0	32256	512
	32401	15.0	32400	25
	32537	15.0	32536	49
	32563	15.0	32562	243
	32609	15.0	32608	32
	32689	15.0	48	16
16	39367	15.3	39366	19683
	40961	15.3	40960	8192
	52489	15.7	52488	6561
	61441	15.9	61440	4096
	62501	15.9	62500	15625
	63001	15.9	250	125
	64153	16.0	64152	729
	64513	16.0	64512	1024
	65089	16.0	65088	64
	65101	16.0	65100	25
	65171	16.0	65170	343
	65269	16.0	65268	49
	65281	16.0	96	32
	65449	16.0	65448	81
	65521	16.0	65520	16

Фурье по одному основанию.

Для вычислений в кольце $Z/(q)$ необходимо знать вычеты по модулю q, которые в общем случае вычисляются не столь просто, как в случае простых чисел Ферма или Мерсенна. Это является одним из основных недостатков числовых преобразований Шевилла; возможным выходом является предварительное вычисление и табулирование вычетов по модулю простых чисел.

6.8. Алгоритм Препараты—Сервейта

Так же как вычисления в поле комплексных чисел вкладываются в поля Галуа, вычисления в полях Галуа можно вложить в поле комплексных чисел. Для вычисления свертки в $GF(q)$ можно воспользоваться комплекснозначным БПФ-алгоритмом. Предположим, что в поле $GF(q)$, где q равно степени простого числа p, требуется вычислить произведение $s(x) = g(x)\,d(x)$. Представим элементы поля $GF(q)$ в виде многочленов; тогда произведение элементов поля в $GF(q)$ можно интерпретировать как свертку многочленов по модулю неприводимого многочлена $p(x)$. Вычисление всех вычетов по модулю этого неприводимого многочлена можно отложить до тех пор, пока не будут вычислены все свертки. Исходная свертка при этом превращается в двумерную свертку.

Запишем элементы g_i и d_i поля $GF(q)$ над простым полем $GF(p)$ в виде многочленов

$$g_i = \sum_{l=0}^{m-1} g_{il} z^l \quad \text{и} \quad d_i = \sum_{l=0}^{m-1} d_{il} z^l,$$

где $q = p^m$, а g_{il} и d_{il} обозначают неотрицательные целые числа, не превосходящие $p-1$. Тогда линейная свертка векторов $\mathbf{g}$ и $\mathbf{d}$ равна

$$s_i = \sum_{k=0}^{n-1} g_k d_{i-k} =$$

$$= \sum_{k=0}^{n-1} \sum_{l=0}^{m-1} \sum_{l'=0}^{m-1} g_{kl} d_{(i-k)\,l'} z^{l+l'} \pmod{p}\,(\mathrm{mod}\ p(x)),$$

где p — характеристика поля, а $p(x)$ — простой многочлен степени m. Определим двумерную целочисленную свертку равенством

$$s_{ii'} = \sum_{k=0}^{n-1} \sum_{k'=0}^{m-1} g_{kk'} d_{(i-k)\,(i'-k')}, \qquad \begin{aligned} i &= 0, \ldots, n-1, \\ i' &= 0, \ldots, 2m-1. \end{aligned}$$

Каждый элемент такой двумерной таблицы представляет собой число между 0 и $p-1$. Тогда

$$s_i = \sum_{i'=0}^{2m-1} s_{ii'} z^{i'} \pmod{p} \pmod{p(x)}.$$

Вычисления вычетов делаются на последнем этапе вычислений, причем сначала вычисляются вычеты целых чисел по модулю p, а затем вычеты многочленов $s_i(z) = \sum_{i'=0}^{2m-1} s_{ii'} z^{i'}$ по модулю многочлена $p(x)$. Сложностью вычисления вычетов можно пренебречь по сравнению со сложностью вычисления двумерной свертки. Двумерную свертку можно вычислять в любом подходящем суррогатном поле, например в поле вещественных или поле комплексных чисел.

Вместо поля комплексных чисел можно использовать как суррогатное поле любое подходящее конечное поле, даже с характеристикой, отличной от характеристики исходного поля. Например, для свертки последовательностей над $GF(2)$, длина n которых меньше, чем половина простого числа Ферма 2^m+1 (при $m = 2, 4, 8, 16$ соответственно $2^m+1 = 5, 17, 257, 65537$), можно использовать поле $GF(2^m+1)$. Для этого достаточно проинтерпретировать последовательность над $GF(2)$ как последовательность целых чисел, которые принимают только значения, равные 0 и 1. Длина линейной свертки таких целочисленных последовательностей, хотя и больше n, но не превосходит 2^m, и, следовательно, эту линейную свертку можно вычислять как циклическую свертку в поле $GF(2^m+1)$ с последующим приведением целых чисел по модулю 2.

Для вычисления циклической свертки в поле $GF(2^m+1)$ можно воспользоваться любым допустимым быстрым алгоритмом. Так как в поле $GF(2^m+1)$ существует преобразование Фурье длины n, равной любому делителю числа 2^m, то одной из возможных схем вычислений является использование преобразования Фурье и теоремы о свертке. Преобразование Фурье при этом можно вычислять по БПФ-алгоритму Кули—Тьюки, содержащему $(n/2)\log_2 n$ умножений и столько же сложений в поле $GF(2^m+1)$. Далее, так как элементы поля $GF(2^m+1)$ допускают запись в виде $(m+1)$-разрядных двоичных чисел, то сложность такого алгоритма свертки сравнима со сложностью в поле комплексных чисел.

Задачи

6.1. а. Построить логические схемы, реализующие операции умножения и сложения в поле Мерсенна $GF(7)$. Позволяет ли представление 0 в двух видах получить какое-то преимущество?

б. Построить логические схемы, реализующие операции умножения и сложения в поле Ферма GF (5). Позволяет ли представление чисел 0, 1 и 2 получить какие-то преимущества?

6.2. **а.** Проверить, что если ядро преобразования Фурье в поле $GF(2^{2^m}+1)$ равно 2, то все необходимые умножения на самом деле являются умножениями на степени двойки и могут быть заменены циклическими сдвигами.
б. Элемент $\omega = 8$ в поле GF (193) имеет порядок, равный 32, и, следовательно, может быть выбран в качестве ядра 32-точечного преобразования Фурье в этом поле. Можно ли как-то осмысленно утверждать, что все умножения такого преобразования являются умножениями на степени двойки и могут быть заменены циклическими сдвигами?

6.3. Порядок элемента 2 в кольце $Z/(15)$ равен четырем. Выписать (4×4)-матрицы 4-точечного преобразования Фурье и ему обратного преобразования. Показать, что эти матрицы не ведут себя стандартным для матриц преобразования Фурье образом. Какие свойства оказываются неверными? Почему?

6.4. **а.** Чему равен порядок элемента 2 в простом поле Ферма GF (17)?
б. Описать 8-точечный БПФ-алгоритм Кули — Тьюки по основанию два для поля GF (17). Сколько умножений содержит этот алгоритм?
в. Описать 4-точечный БПФ-алгоритм Кули — Тьюки по основанию два, ядром которого является элемент **4**.
г. В поле GF (17) имеет место равенство $\sqrt{2} = 11$. Является ли 11 примитивным элементом поля? Описать 16-точечный БПФ-алгоритм Кули — Тьюки для поля GF (17) с минимальным числом умножений.

6.5. Пусть $q = (2^8+1)(2^{16}+1)$. Чему равен порядок элемента 2 в кольце $Z/(q)$?

6.6. Число 31 является простым числом Мерсенна.
а. Чему равен порядок элемента 2 в поле GF (31)?
б. Выписать 5-точечный БПФ-алгоритм Винограда в поле GF (31).
в. В поле GF (31) элемент -1 не имеет квадратного корня. Чему равен порядок элемента $1+j$ в расширении $GF(31^2)$ поля?
г. Чему равна наибольшая допустимая длина преобразования Фурье в поле $GF(31^2)$?
д. Описать структуру 8-точечного БПФ в поле $GF(31^2)$.

6.7. Показать, что порядок элемента $1+j$ в поле комплексных чисел Мерсенна $GF((2^{17}-1)^2)$ равен 136. Это означает, что в данном поле существует преобразование Фурье длины 136. Описать структуру быстрого алгоритма вычисления этого преобразования.

6.8. Построить 5-точечный БПФ-алгоритм Винограда для поля GF (41). (*Указание*: чему равен порядок элемента 2 в поле GF (41)?)

6.9. В поле $GF((2^{17}-1)^2)$ существует 17-точечное преобразование Фурье.
а. Описать алгоритм вычисления этого преобразования с помощью 16-точечного преобразования Фурье в поле комплексных чисел. Выписать блок-схему такого вычисления.
б. Сколько потребуется вещественных умножений и сложений, если 16-точечное преобразование Фурье в поле комплексных чисел вычислять с помощью малого БПФ-алгоритма Винограда?
в. Каковы должны быть разрядность и процедура округления для того, чтобы комплексные вычисления приводили к правильному ответу в поле $GF((2^{17}-1)^2)$?

6.10. В поле комплексных чисел имеется 17-точечное преобразование Фурье.
а. Описать, как это преобразование может быть вычислено с помощью двух 16-точечных преобразований в поле комплексных чисел Мерсенна.
б. Сколько умножений и сложений в суррогатном поле потребуется при таком вычислении?

6.11. **а.** Показать, что многочлен $x^{32}-1$ имеет в поле $GF(2^{16}+1)$ 32 различных корня.

б. Сколько умножений общего типа содержит 32-точечный алгоритм Винограда вычисления циклической свертки в расширении $GF((2^{16}+1)^m)$ поля $GF(2^{16}+1)$?
в. Сколько умножений общего типа требуется для вычисления 32-точечной циклической свертки в $GF((2^{16}+1)^2)$ при использовании БПФ и теоремы о свертке?
г. Объяснить взаимосвязь между п. (б) и (в) задачи.

6.12. Выпишите детально 16-точечный БПФ-алгоритм Винограда для поля GF (17) и его расширений. Сколько требуется нетривиальных умножений?

Замечания

Преобразования Фурье рассматривались в произвольных полях, но тот факт, что вещественную свертку можно вычислять, переходя в поле целых чисел по модулю простых чисел Мерсенна или простых чисел Ферма, впервые отметил Рейдер [1] (1972). Это погружение было разработано им в связи с приложением к цифровой обработке сигналов, как существенно упрощающее необходимые вычисления. Агарвал и Баррас [2] (1973) также описали применение числовых преобразований Ферма и разработали их структуру [3, 4] (1974, 1975). Шевилла [11] (1978) рассмотрел возможности использования других простых полей. Некоторые вопросы реализации этих числовых преобразований были рассмотрены Макклелланом [5] (1976) и Лейбовицем [6] (1976). Предложение использовать комплексные расширения полей Галуа для вычисления комплексных сверток сделали Рид и Труонг [7] (1975) и Нуссбаумер [8] (1976). Приведенная нами табл. 6.1 основана на работе Рида и Труонга.

Алгоритмы свертки в полях Галуа малой характеристики могут отличаться от алгоритмов свертки в поле вещественных чисел. Алгоритмы свертки в полях малой характеристики рассмотрел Райс [9] (1980), и мы воспользовались некоторыми его примерами. Идея использования суррогатных полей принадлежит Препарате и Сервейту [10] (1977), которые вложили вычисление свертки в полях Галуа в поле комплексных чисел.

Использование китайской теоремы об остатках для представления данных (в противоположность использованию для переупорядочивания индексов) хорошо известно; можно сослаться на учебник Сабо и Танаки [12] (1967) или более недавнюю работу Дженкинса и Леона [13] (1977).

Глава 7

БЫСТРЫЕ АЛГОРИТМЫ И МНОГОМЕРНЫЕ СВЕРТКИ

Подобно тому, как определяется одномерная свертка, можно определить многомерную свертку. Для многомерных таблиц данных можно дать полезные во многих отношениях определения линейной и циклической сверток для любого представляющего интерес поля вычислений. Многомерные таблицы данных и многомерные свертки создаются искусственным образом как часть алгоритмов обработки одномерных векторов данных. Многомерные таблицы данных возникают также естественным образом во многих задачах цифровой обработки сигналов, в частности, в задачах обработки изображений, например фотографий, полученных со спутников, и медицинских изображений, в том числе рентгенограмм, в задачах сейсмографии и электронной микроскопии.

Глава начинается с гнездового метода построения быстрых алгоритмов вычисления многомерных сверток из быстрых алгоритмов для одномерных сверток. Затем рассматривается способ построения быстрых алгоритмов вычисления одномерной свертки путем временного отображения в многомерную свертку, — процедура, известная под названием алгоритм Агарвала—Кули вычисления свертки. Алгоритм Агарвала—Кули для одномерной циклической свертки представляет собой мощное дополнение к развитым в гл. 3 методам, которые становятся чрезмерно громоздкими для длинных сверток. Он позволяет строить алгоритмы для длинных сверток из коротких сверток. Наконец, последняя половина главы содержит методы, развитые специально для вычисления двумерных сверток.

7.1. Гнездовые алгоритмы свертки

Двумерная свертка представляет собой операцию, задаваемую на паре двумерных таблиц. Ее можно представлять себе как операцию фильтрации, в которой одна двумерная таблица представляет собой двумерный фильтр, через который пропускается вторая двумерная таблица и на выходе которого формируется выходной двумерный сигнал. Такого сорта операции возникают в задачах обработки изображений.

Заданную $(N' \times N'')$-таблицу

$$\mathbf{d} = \{d_{i',\,i''},\quad i' = 0, \ldots, N' - 1;\quad i'' = 0, \ldots, N'' - 1\}$$

назовем таблицей данных, а $(L' \times L'')$-таблицу

$$\mathbf{g} = \{g_{i',\,i''},\quad i' = 0, \ldots, L' - 1;\quad i'' = 0, \ldots, L'' - 1\}$$

назовем таблицей фильтра, и вычислим $((L' + N' - 1) \times (L'' + N'' - 1))$-таблицу сигнала

$$\mathbf{s} = \left\{ s_{i',\,i''},\quad \begin{matrix} i' = 0, \ldots, L' + N' - 2, \\ i'' = 0, \ldots, L'' + N'' - 2 \end{matrix} \right\},$$

задавая правила вычисления ее элементов равенствами

$$s_{i',\,i''} = \sum_{k'=0}^{N'-1} \sum_{k''=0}^{N''-1} g_{i'-k',\,i''-k''} d_{k',\,k''} \qquad \begin{matrix} i' = 0, \ldots, L' + N' - 2, \\ i'' = 0, \ldots, L'' + N'' - 2. \end{matrix}$$

Это определение можно непосредственно перенести и на свертки большей размерности. Однако мы ограничим рассмотрение в основном двумерными [1]) сверточками, так как все идеи легко обобщаются на случай произвольной размерности.

Двумерная свертка может быть выписана в терминах многочленов; полиномиальные обозначения можно вводить как по любому из измерений, так и по обоим измерениям одновременно. Таким образом, таблицу $\mathbf{d}$ можно записать или как вектор многочленов,

$$d_{i'}(y) = \sum_{i''=0}^{N''-1} d_{i',\,i''} y^{i''},$$

или как многочлен от двух переменных

$$d(x,\ y) = \sum_{i'=0}^{N'-1} \sum_{i''=0}^{N''-1} d_{i',\,i''} y^{i''} x^{i'}.$$

Аналогичные обозначения можно ввести для $\mathbf{g}$ и $\mathbf{s}$:

$$g_{i'}(y) = \sum_{i''=0}^{L''-1} g_{i',\,i''} y^{i''}, \qquad s_{i'}(y) = \sum_{i''=0}^{L''+N''-2} s_{i',\,i''} y^{i''},$$

$$g(x,\ y) = \sum_{i'=0}^{L'-1} \sum_{i''=0}^{L''-1} g_{i',\,i''} y^{i''} x^{i'},$$

$$s(x,\ y) = \sum_{i'=0}^{L'+N'-2} \sum_{i''=0}^{L''+N''-2} s_{i',\,i''} y^{i''} x^{i'}.$$

[1]) Так определенная двумерная свертка в дальнейшем часто называется $(N' \times N'')$-сверткой; ее не следует путать с определенной в гл. 3 линейной $(L \times N)$-сверткой. — *Прим. перев.*

Двумерная свертка в этих обозначениях может быть записана или как одномерная свертка многочленов,

$$s_{i'}(y) = \sum_{k'=0}^{N'-1} g_{i'-k'}(y)\, d_{k'}(y),$$

или как произведение многочленов от двух переменных,

$$s(x, y) = g(x, y)\, d(x, y).$$

Двумерная циклическая свертка также может быть выписана в терминах многочленов, а именно,

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^{n'} - 1} \pmod{y^{n''} - 1},$$

где n' не обязательно равно n''. Для двумерной циклической свертки выполняется теорема о свертке. Если $\mathbf{D}$ и $\mathbf{G}$ обозначают двумерные преобразования Фурье таблиц $\mathbf{d}$ и $\mathbf{g}$ соответственно и $S_{k', k''} = G_{k', k''} D_{k', k''}$, то $\mathbf{S}$ является двумерным преобразованием Фурье таблицы $\mathbf{s}$. Следовательно, для вычисления двумерной циклической свертки можно пользоваться двумерным БПФ-алгоритмом. Можно, конечно, пользоваться и прямыми методами вычисления двумерных сверток.

Имеется много способов прямого вычисления двумерной линейной или циклической свертки. Простейший путь состоит в том, чтобы попытаться вычислить ее как последовательность одномерных сверток сначала по строкам, а затем по столбцам. Заметим, однако, что в формуле

$$s_{i', i''} = \sum_{k'=0}^{N'-1} \left[\sum_{k''=0}^{N''-1} g_{i'-k', i''-k''} d_{k', k''} \right]$$

выделенная в квадратные скобки линейная свертка по k'' должна быть вычислена для всех i' и k'.

Понять двумерную свертку легче, если выписать ее в виде свертки многочленов,

$$s_{i'}(y) = \sum_{k'=0}^{N'-1} g_{i'-k'}(y)\, d_{k'}(y) \qquad i' = 0, \ldots, L' + N' - 2.$$

Из этой формулы ясно видна реализация двумерной свертки как $L'N'$ произведений многочленов, каждое из которых, в свою очередь, является сверткой, требующей $L''N''$ умножений. Всего, таким образом, получаем $L'L''N'N''$ умножений, что, конечно, недопустимо для больших задач, так что нужны быстрые алгоритмы.

Хотя рассмотренные в гл. 3 алгоритмы свертки строились над полями, они на самом деле выполнимы и в некоторых кольцах, в частности, в кольце многочленов. Следовательно, свертка многочленов может быть вычислена любым из этих быстрых алгоритмов. Сложение в этих алгоритмах становится сложением многочленов, а умножение — умножением многочленов. Эти последние умножения, в свою очередь, являются свертками и могут быть вычислены с помощью быстрых алгоритмов свертки. Таким образом, мы получаем способ вычисления двумерной свертки, состоящий во включении одного быстрого алгоритма одномерной свертки в другой быстрый алгоритм одномерной свертки.

Все сказанное здесь относительно линейной свертки, конечно, относится и к циклической свертке и к вычислению произведения многочленов по модулю многочлена $m(x)$.

Например, для вычисления двумерной циклической (4×4)-свертки воспользуемся алгоритмом 4-точечной циклической свертки в виде, представленном на рис. 3.14: $\mathbf{s} = \mathbf{C}\,[(\mathbf{Bg})\cdot(\mathbf{Ad})]$. Этот алгоритм содержит пять умножений и 15 сложений. Следовательно, применение его для циклической свертки многочленов потребует пяти умножений многочленов и 15 сложений многочленов. Каждое умножение многочленов само является сверткой, и, следовательно, для его вычисления по этому алгоритму потребуется пять вещественных умножений и 15 вещественных сложений. Таким образом, для вычисления двумерной циклической (4×4)-свертки потребуется в данном алгоритме 25 вещественных умножений и 135 вещественных сложений. Эта процедура представлена на рис. 7.1 для вычисления циклической $(n'\times n'')$-свертки. Как показано на рисунке, подпрограмма должна знать индекс k'' и иметь доступ к заранее вычисленной таблице $G_{k'k''}$.

Используя последовательные операции над двумерными таблицами, этот составной алгоритм можно представить в виде единого алгоритма. Сначала умножим строку таблицы $\mathbf{d}$ на матрицу $\mathbf{A}''$, а каждую строку таблицы $\mathbf{g}$ на матрицу $\mathbf{B}''$. Затем умножим каждый столбец первой новой таблицы на $\mathbf{A}'$ и каждый столбец второй новой таблицы на $\mathbf{B}'$. Это даст нам две таблицы размера $M(n')\times M(n'')$. Перемножим покомпонентно эти таблицы. Полученную таким образом $(M(n')\times M(n''))$-таблицу преобразуем к нужной выходной таблице $\mathbf{s}$ размера $n'\times n''$, умножив сначала каждый ее столбец на $\mathbf{C}'$, а затем каждую строку полученной таблицы на $\mathbf{C}''$.

На рис. 7.2 для сравнения выписаны эта процедура и процедура, основанная на двумерном преобразовании Фурье и теореме о свертке. Ясно, что рассматриваемая процедура является обобщением процедуры, основанной на двумерном преобразовании Фурье. Теперь «преобразование» состоит только из сложений, но размер промежуточной таблицы равен не $n'\times n''$, а $M(n')\times M(n'')$.

Гнездовой алгоритм является более общим и в другом отношении: он позволяет вычислять двумерное произведение много-

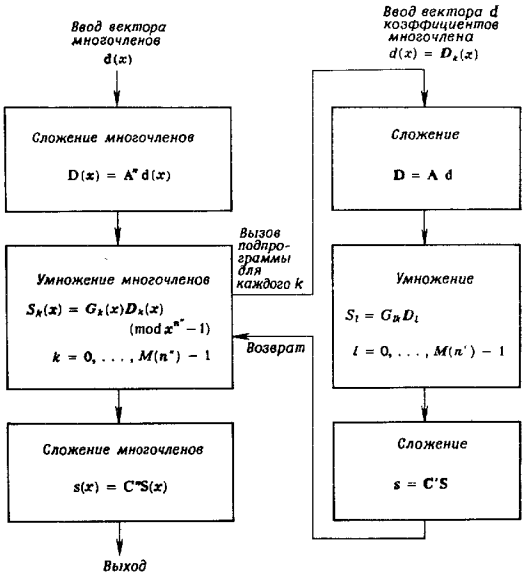


Рис. 7.1. Алгоритм двумерной циклической свертки.

членов по модулю любых многочленов $m'(x)$ и $m''(y)$, а метод, основанный на преобразовании Фурье, позволяет вычислять произведение только по модулю $x^{n'}-1$ и $y^{n''}-1$.

Характеристики гнездового алгоритма задаются числом $M(n'\times n'')$ необходимых умножений и числом $A(n'\times n'')$ необходимых сложений. Легко видеть, что полное число умножений равно

$$M(n'\times n'') = M(n')\,M(n'').$$

Аналогично, из рис. 7.1 легко увидеть, что

$$A(n'\times n'') = n'A(n'') + M(n'')\,A(n').$$

Полное число умножений не зависит от того, какой из делителей назван n', а какой n''. Но число сложений зависит от такого выбора, и поэтому необходимо просматривать оба варианта. На-

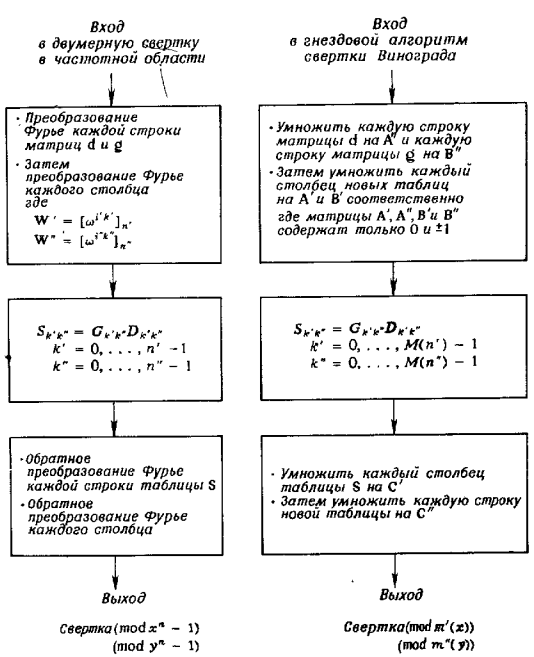


Рис. 7.2. Сравнение способов вычисления двумерной свертки.

пример, двумерную циклическую (7×9)-свертку можно вычислять с помощью одномерных алгоритмов с характеристиками

$$M(7) = 16,\ A(7) = 70,$$

$$M(9) = 19,\ A(9) = 74.$$

Полное число умножений при этом равно 304, а полное число сложений равно 1848 или 1814 в зависимости от того, чему выбрано равным n'.

7.2. Алгоритм Агарвала—Кули вычисления свертки

Алгоритм Агарвала—Кули представляет собой метод преобразования $n'n''$-точечной одномерной циклической свертки в двумерную циклическую $(n' \times n'')$-свертку при условии, что числа n' и n'' взаимно просты. Его можно использовать для того, чтобы скомбинировать алгоритм Винограда вычисления n'-точечной циклической свертки, содержащий $M(n')$ умножений, с алгоритмом Винограда вычисления n''-точечной циклической свертки, содержащим $M(n'')$ умножений, и построить алгоритм вычисления $n'n''$-точечной циклической свертки, содержащий $M(n')\,M(n'')$ умножений.

В основе алгоритма Агарвала—Кули преобразования одномерной циклической свертки в многомерную циклическую свертку лежит китайская теорема об остатках для целых чисел. Этим алгоритм отличается от алгоритма Винограда, в котором используется китайская теорема об остатках для многочленов. Он не обеспечивает столь сильного уменьшения числа умножений, как алгоритм Винограда вычисления свертки, но, с другой стороны, он не имеет и тенденции к чрезмерному росту с ростом n числа сложений, как в алгоритме Винограда. Кроме того, он лучше структурирован. Если n велико, то алгоритм Агарвала—Кули более управляем, так как может быть разбит на подпрограммы. Для построения хороших алгоритмов свертки надо использовать алгоритм Агарвала—Кули разбиения длинной свертки на короткие циклические свертки, для вычисления которых использовать затем эффективные алгоритмы Винограда вычисления коротких циклических сверток.

Алгоритм Агарвала—Кули строится на трех идеях. Сначала одномерная циклическая свертка с помощью китайской теоремы об остатках (К. Т. О.) для целых чисел преобразуется в многомерную циклическую свертку; затем вдоль каждой координаты используется алгоритм Винограда вычисления короткой свертки; и, наконец, для сборки вместе полученных преобразований используется теорема о кронекеровском произведении для матриц.

Пусть заданы d_i и g_i для $i = 0, \ldots, n-1$, и нам надо построить компоненты циклической свертки

$$s_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k, \qquad i = 0, \ldots, n-1,$$

где двойные скобки в индексах обозначают вычисление по модулю n.

Мы хотим преобразовать одномерную свертку в двумерную. Для этого воспользуемся китайской теоремой об остатках, позво-

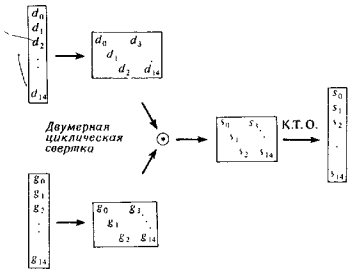


Рис. 7.3. Иллюстрация алгоритма Агарвала—Кули.

ляющей отобразить одномерный вектор входных данных в двумерную таблицу и выходную двумерную таблицу в одномерный вектор. Заменим индексы i и k парами индексов (i', i'') и (k', k'') по правилу

$$i' = i \pmod{n'}, \qquad i'' = i \pmod{n''},$$
$$k' = k \pmod{n'}, \qquad k'' = k \pmod{n''}.$$

Занимаясь китайской теоремой об остатках, мы уже видели, что старые индексы восстанавливаются по новым согласно формулам

$$i = N''n''i' + N'n'i'' \pmod{n}, \qquad i' = 0, \ldots, n'-1, \quad i'' = 0, \ldots, n''-1,$$
$$k = N''n''k' + N'n'k'' \pmod{n}, \qquad k' = 0, \ldots, n'-1, \quad k'' = 0, \ldots, n''-1,$$

где целые числа N' и N'' определяются равенством

$$N'n' + N''n'' = 1.$$

Это то же самое соответствие между компонентами вектора и компонентами двумерной таблицы, которое было использовано в алгоритме Гуда—Томаса. Описанное отображение иллюстрируется рис. 7.3.

Свертку

$$s_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k$$

можно записать в виде

$$s_{N''n''i'+N'n'i''} = \sum_{k'=0}^{n'-1} \sum_{k''=0}^{n''-1} g_{N''n''(i'-k')+N'n'(i''-k'')} d_{N''n''k'+N'n'k''}.$$

Длина n	Число вещественных умножений $M(n)$	Число вещественных сложений $A(n)$	Число вещественных умножений на точку $M(n)/n$	Число вещественных сложений на точку $A(n)/n$
18	38	184	2 11	10 22
20	50	230	2 50	11 50
24	56	272	2 33	11.33
30	80	418	2.67	13.93
36	95	505	2.64	14.03
48	132	900	2.75	18.75
60	200	1120	3.33	18 67
72	266	1450	3.69	20 14
84	320	2100	3.81	25 00
120	560	3096	4 67	25 80
180	950	5470	5.28	30 39
210	1280	7958	6.10	37 90
240	1056	10176	4.40	42.40
360	2280	14748	6.33	40.97
420	3200	20420	7.62	48.62
504	3648	26304	7.24	52.19
840	7680	52788	9.14	62.84
1008	10032	71265	9.95	70.70
1260	12160	95744	9.65	75.99
2520	29184	241680	11.58	95 90

Рис. 7.4. Характеристики алгоритма Агарвала—Кули вычисления свертки.

Двойное суммирование по k' и k'' эквивалентно одинарному суммированию по k, так как затрагивает одни и те же члены. Определим двумерные переменные, которые назовем также d, g и s, задавая их равенствами

$$d_{k',\,k''} = d_{N''n''k'+N'n'k''}, \qquad k' = 0, \ldots, n'-1, \quad k'' = 0, \ldots, n''-1,$$

$$g_{k',\,k''} = g_{N''n''k'+N'n'k'}, \qquad k' = 0, \ldots, n'-1, \quad k'' = 0, \ldots, n''-1,$$

$$s_{k',\,k''} = s_{N''n''k'+N'n'k''}, \qquad k' = 0, \ldots, n'-1, \quad k'' = 0, \ldots, n''-1.$$

В новых переменных свертка записывается в виде

$$s_{i',\,i''} = \sum_{k'=0}^{n'-1} \sum_{k''=0}^{n''-1} g_{((i'-k'))_{n'},\,((i''-k''))_{n''}} d_{k'k''},$$

где первые и вторые индексы вычисляются соответственно по модулю n' и n''. Теперь это двумерная циклическая свертка. Тем не менее, мы пока не получили никакого уменьшения сложности вычислений. Для каждой пары индексов (i', i'') каждое число $d_{k',\,k''}$ на что-то умножается, так что пока полное число умножений равно $(n'n'')^2$.

Для того, чтобы сделать алгоритм эффективным, надо воспользоваться развитыми в предыдущем разделе методами быстрого

вычисления двумерной свертки. Алгоритм Агарвала—Кули вычисления свертки состоит в переходе от одномерных векторов к двумерным таблицам, в применении быстрых алгоритмов двумерной свертки и в обратном переходе от двумерной таблицы к одномерному выходному вектору.

Характеристики n-точечного алгоритма Агарвала—Кули подытожены в таблице на рис. 7.4. Полное число сложений и умножений определяется гнездовым методом построения алгоритмов двумерной циклической свертки из эффективных одномерных алгоритмов составляющих длин n' и n'' по формулам

$$A(n) = n'A(n'') + M(n'')\,A(n'),$$

$$M(n) = M(n')\,M(n'').$$

Из этих формул видно, что число умножений в малых алгоритмах играет существенно более важную роль, чем число сложений. Например, предположим, что алгоритм 504-точечной циклической свертки строится из 7-, 9- и 8-точечных циклических сверток с

$$M(7) = 16,\ A(7) = 70,$$

$$M(9) = 19,\ A(9) = 74,$$

$$M(8) = 14,\ A(8) = 46 \text{ или } M(8) = 12,\ A(8) = 72.$$

Может показаться, что алгоритм 8-точечной циклической свертки с 14 умножениями лучше, так как он содержит всего на два умножения больше, но зато на 26 сложений меньше. Однако включение его в алгоритм 504-точечной свертки приводит к удивительным результатам:

$$M(504) = 4256, \qquad A(504) = 28\,240$$

или

$$M(504) = 3648, \qquad A(504) = 26\,304.$$

Таким образом, использование в 504-точечном алгоритме 8-точечного алгоритма с 12 умножениями приводит к уменьшению и числа умножений, и числа сложений.

Выполнять вычисления надо следующим образом. Сначала каждый столбец двумерной таблицы умножается на матрицу $\mathbf{A}''$. Затем начинается вычисление произведений многочленов. Этот процесс начинается умножением каждой строки двумерной таблицы на матрицу $\mathbf{A}'$. Затем каждая строка покомпонентно умножается на вектор констант, так что и вся двумерная таблица умножается покомпонентно на таблицу констант. Умножение многочленов завершается умножением каждого столбца на матрицу $\mathbf{C}'$, а затем каждой строки на матрицу $\mathbf{C}''$.

Эта последовательность вычислений может быть представлена в более компактном виде, если собрать воедино два алгоритма циклических сверток. Запишем входную двумерную таблицу в виде стека по столбцам, представив ее одномерным входным массивом. Аналогично запишем и выходную двумерную таблицу, вытянув ее в выходной одномерный массив. Тогда алгоритм примет вид

$$\mathbf{s} = ((\mathbf{C}'' \times \mathbf{C}')(\mathbf{G}' \times \mathbf{G}'')(\mathbf{A}' \times \mathbf{A}''))\,\mathbf{d}.$$

Аналогично, можно выходную двумерную таблицу записывать в виде одномерного массива, считывая ее по строкам. Тогда алгоритм запишется даже в более симметричной форме

$$\mathbf{s} = ((\mathbf{C}' \times \mathbf{C}'')(\mathbf{G}' \times \mathbf{G}'')(\mathbf{A}' \times \mathbf{A}''))\,\mathbf{d},$$

где компоненты вектора $\mathbf{s}$ выписаны в другом порядке, хотя и остаются неизменными. В любом из случаев, вводя очевидные обозначения для матриц, можно записать

$$\mathbf{s} = \mathbf{CGA}\mathbf{d}.$$

Есть еще одна последняя деталь, о которой надо позаботиться: компоненты векторов $\mathbf{s}$ и $\mathbf{d}$ выписаны не в своем естественном порядке. Сначала они выписывались в двумерную матрицу вдоль главной диагонали, а затем в виде стека столбцов (или строк). Для того, чтобы вернуть компоненты на их естественное место, надо переставить столбцы матрицы $\mathbf{A}$ и строки матрицы $\mathbf{C}$.

В качестве примера построим алгоритм 12-точечной циклической свертки. Воспользуемся следующим 3-точечным алгоритмом циклической свертки:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 2 \\ 1 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} G_0'' & & \\ & G_1'' & \\ & & G_2'' \\ & & & G_3'' \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix},$$

где

$$\begin{bmatrix} G_0'' \\ G_1'' \\ G_2'' \\ G_3'' \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ \frac{1}{3} & \frac{1}{3} & -\frac{2}{3} \end{bmatrix} \begin{bmatrix} g_0'' \\ g_1'' \\ g_2'' \end{bmatrix}$$

и следующим алгоритмом 4-точечной циклической свертки

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & -1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} G_0' & & & & \\ & G_1' & & & \\ & & G_2' & & \\ & & & G_3' & \\ & & & & G_4' \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix},$$

где

$$\begin{bmatrix} G_0' \\ G_1' \\ G_2' \\ G_3' \\ G_4' \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{2} \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} g_0' \\ g_1' \\ g_2' \\ g_3' \end{bmatrix}.$$

Сначала выпишем 12 коэффициентов фильтра в виде двумерной (3×4)-таблицы, располагая их вдоль главной диагонали, и вытягивая затем их в стек по столбцам:

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \cdot \\ \cdot \\ \cdot \\ g_{11} \end{bmatrix} \rightarrow \begin{bmatrix} g_0 & g_9 & g_6 & g_3 \\ g_4 & g_1 & g_{10} & g_7 \\ g_8 & g_5 & g_2 & g_{11} \end{bmatrix} \rightarrow \begin{bmatrix} g_0 \\ g_4 \\ g_8 \\ g_9 \\ g_1 \\ g_5 \\ g_6 \\ g_{10} \\ g_2 \\ g_3 \\ g_7 \\ g_{11} \end{bmatrix}$$

Тогда диагональная (20×20)-матрица $\mathbf{G}$ в терминах кронекеровского произведения записывается равенством

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ \cdot \\ \cdot \\ \cdot \\ G_{19} \end{bmatrix} = \left\{ \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \times \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ \frac{1}{3} & \frac{1}{3} & -\frac{2}{3} \end{bmatrix} \right\} \begin{bmatrix} g_0 \\ g_4 \\ g_8 \\ g_9 \\ \cdot \\ \cdot \\ \cdot \\ g_{11} \end{bmatrix}.$$

Ту же самую конструкцию надо применить к 12 компонентам входного вектора данных; выпишем кронекеровскую матрицу $\mathbf{A}' \times \mathbf{A}''$ полностью:

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ \\ \\ \\ \\ \\ \\ \\ D_{11} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\ 1 & 1 & -2 & 1 & 1 & -2 & 1 & 1 & -2 & 1 & 1 & -2 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 & -1 & 1 & 0 & 1 & -1 & 0 & -1 & 1 \\ 1 & 1 & -2 & -1 & -1 & 2 & 1 & 1 & -2 & -1 & -1 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 0 & -1 & 1 & 0 & -1 & -1 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 & 1 & -1 & 0 & -1 & 1 & 0 & -1 & 1 \\ 1 & 1 & -2 & 1 & 1 & -2 & -1 & -1 & 2 & -1 & -1 & 2 \\ 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 1 & 1 & -2 & 0 & 0 & 0 & -1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & 1 & -2 & 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \\ d_{10} \\ d_{11} \end{bmatrix}.$$

Наконец, перестановка столбцов дает

$$\begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ \\ \\ \\ \\ \\ \\ \\ D_{11} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\ 1 & 1 & -2 & 1 & 1 & -2 & 1 & 1 & -2 & 1 & 1 & -2 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & 1 \\ 0 & -1 & -1 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & 1 & 1 \\ 1 & -1 & -2 & -1 & 1 & 2 & 1 & -1 & -2 & -1 & 1 & 2 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 0 & 1 & -1 & 0 & -1 & -1 & 0 & -1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & -1 & 0 & -1 & -1 & 0 & -1 & 1 \\ 1 & 1 & 2 & -1 & 1 & -2 & -1 & -1 & -2 & 1 & -1 & 2 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 1 & 0 & 2 & 0 & 1 & 0 & -1 & 0 & -2 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 & -2 & 0 & -1 & 0 & 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ \\ \\ \\ \\ \\ \\ d_{11} \end{bmatrix}.$$

Это полностью определяет вычисление $\mathbf{D} = \mathbf{Ad}$. Аналогичные рассуждения приводят к построению матрицы $\mathbf{C}$, и, в итоге, к стандартной матричной форме $\mathbf{s} = \mathbf{CGAd}$ алгоритма вычисления 12-точечной циклической свертки. В такой форме алгоритма все операции перестановки компонент скрыты в матрицах предсложений и постсложений.

7.3. Алгоритмы разложения

Мы уже видели, как гнездовой метод позволяет преобразовать алгоритм одномерной свертки в многомерные алгоритмы. Для вычисления

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^{n'} - 1} \pmod{y^{n''} - 1}$$

мы объединяли алгоритмы для вычислений

$s(x) = g(x)\, d(x) \pmod{x^{n'} - 1}$ и $s(y) = g(y)\, d(y) \pmod{y^{n''} - 1}$.

В более общей формулировке, для построения алгоритма вычисления

$$s(x, y) = g(x, y)\, d(x, y) \pmod{p(x)} \pmod{q(y)},$$

где $p(x)$ и $q(y)$ представляют собой многочлены степеней n' и n'' соответственно, мы комбинировали алгоритмы для решения двух простых одномерных задач

$s(x) = g(x)\, d(x) \pmod{p(x)}$ и $s(y) = g(y)\, d(y) \pmod{q(y)}$.

Если каждый из подалгоритмов содержит соответственно $M(n')$ и $M(n'')$ умножений, то полученный таким методом алгоритм решения двумерной задачи содержит $M(n')\, M(n'')$ умножений.

Используемый вдоль каждого из направлений индивидуальный одномерный алгоритм был построен на основании китайской теоремы об остатках. Это открывает еще одно направление возможных исследований. Можно попытаться использовать китайскую теорему об остатках для разбиения двумерной задачи на подзадачи вычисления двумерных сверток меньших размеров. Предположим, что

$$p(x) = p^{(0)}(x)\, p^{(1)}(x),$$

где многочлены $p^{(0)}(x)$ и $p^{(1)}(x)$ взаимно просты. Вычисление произведения многочленов

$$s(x, y) = g(x, y)\, d(x, y) \pmod{p(x)} \pmod{q(y)}$$

можно разбить на вычисления произведений

$$s^{(0)}(x, y) = g^{(0)}(x, y)\, d^{(0)}(x, y) \pmod{p^{(0)}(x)} \pmod{q(y)}$$

и

$$s^{(1)}(x, y) = g^{(1)}(x, y)\, d^{(1)}(x, y) \pmod{p^{(1)}(x)} \pmod{q(y)}.$$

Китайская теорема об остатках позволяет восстановить произведение из этих фрагментов. Если $q(y) = q^{(0)}(y)\, q^{(1)}(y)$, то можно произвести факторизацию и по переменной y. Обе эти возможности позволяют разбить задачу на следующие фрагменты:

$$s^{(0,0)}(x, y) = g^{(0,0)}(x, y)\, d^{(0,0)}(x, y) \pmod{p^{(0)}(x)} \pmod{q^{(0)}(y)},$$

$$s^{(0,1)}(x, y) = g^{(0,1)}(x, y)\, d^{(0,1)}(x, y) \pmod{p^{(0)}(x)} \pmod{q^{(1)}(y)},$$

$$s^{(1,0)}(x,\ y) = g^{(1,0)}(x,\ y)\, d^{(1,0)}(x,\ y) \pmod{p^{(1)}(x)} \pmod{q^{(0)}(y)},$$
$$s^{(1,1)}(x,\ y) = g^{(1,1)}(x,\ y)\, d^{(1,1)}(x,\ y) \pmod{p^{(1)}(x)} \pmod{q^{(1)}(y)},$$

где

$$g^{(i,j)}(x,\ y) = g(x,\ y) \pmod{p^{(i)}(x)} \pmod{q^{(j)}(y)},$$

и остальные многочлены определяются аналогично. Выходной многочлен $s(x, y)$ можно восстановить по этим фрагментам, дважды применяя китайскую теорему об остатках для многочленов.

Пусть M_0 и M_1 обозначают число умножений, необходимых для вычисления произведения многочленов по модулю $p^{(0)}(x)$ и $p^{(1)}(x)$ соответственно, а M_0' и M_1' — число умножений, необходимых для вычисления произведения многочленов по модулю $q^{(0)}(y)$ и $q^{(1)}(y)$ соответственно. Тогда прямой метод вычисления произведения

$$s(x, y) = g(x, y)\, d(x, y) \pmod{p(x)} \pmod{q(y)}$$

требует $M = (M_0 + M_1)(M_0' + M_1')$ умножений, а описанное использование китайской теоремы об остатках на двумерных фрагментах $M = M_0M_0' + M_0M_1' + M_1M_0' + M_1M_1'$ умножений, так что число умножений не уменьшается.

Хотя описанный метод и не приводит к улучшениям по числу необходимых умножений, он полезен в том отношении, что расширяет возможности разбиения задачи на подзадачи и уменьшает число сложений.

Имеется много способов разбиения задачи вычисления двумерной свертки с помощью китайской теоремы об остатках. На рис. 7.5 приведены примеры характеристик, которые могут быть получены путем модификации алгоритма Агарвала—Кули с помощью описанных разбиений. Данные на рис. 7.5 следует сравнить с приведенными на рис. 7.4 характеристиками.

В качестве примера опишем построение алгоритма 20-точечной циклической свертки. Сначала преобразуем свертку в двумерную (4×5)-свертку

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^4 - 1} \pmod{y^5 - 1}.$$

Разложим теперь этот алгоритм на двумерные фрагменты в виде

$$s^{(0)}(x,\ y) = g^{(0)}(x,\ y)\, d^{(0)}(x,\ y) \pmod{x^4 - 1} \pmod{y - 1},$$
$$s^{(1)}(x,\ y) = g^{(1)}(x,\ y)\, d^{(1)}(x,\ y) \pmod{x^4 - 1} \pmod{y^4 + y^3 + y^2 + y + 1}.$$

Дальнейшего разложения на двумерные фрагменты делать не будем, а применим к вычислению $s^{(0)}(x, y)$ гнездовой алгоритм вычисления произведения многочленов по модулю $x^4 - 1$, используя для свертки многочленов произведения по модулю $y - 1$, а для вычисления $s^{(1)}(x, y)$ — гнездовой алгоритм, основанный на свертке многочленов по модулю $x^4 - 1$, вложив в него умножение многочленов по модулю $y^4 + y^3 + y^2 + y + 1$. Наконец, произведение $s(x, y)$ вычислим по правилу

$$s(x,\ y) = a^{(0)}(y)\, s^{(0)}(x,\ y) + a^{(1)}(y)\, s^{(1)}(x,\ y) \pmod{y^5 - 1},$$

где $a^{(0)}(y)$ и $a^{(1)}(y)$ вычисляются в соответствии с китайской теоремой об остатках для многочленов. Они равны тем же многочленам, которые должны использоваться для $s^0(y)$ и $s^1(y)$ в одномерной задаче вычисления произведения многочленов по модулю $y^5 - 1$.

Как мы видим, вычисление разбито на те же подзадачи, что рассматривались и ранее, но сочетаются они несколько иначе. Теперь мы пользуемся китайской теоремой об остатках после выполнения гнездового алгоритма, перед которым была проведена одна редукция многочленов. Чтобы подсчитать число необходимых сложений, остановимся подробнее на числе сложений, необходимом для выполнения 5-точечной циклической свертки. Оно равно:

умножение многочленов по модулю $y - 1$: 0 сложений,
умножение многочленов по модулю $y^4 + y^3 + y^2 + y + 1$: 16 сложений,
китайская теорема об остатках: 15 сложений.

Длина n	Число вещественных умножений $M(n)$	Число вещественных сложений $A(n)$	Число вещественных умножений на точку $M(n)/n$	Число вещественных сложений на точку $A(n)/n$
18	38	184	2.11	10.22
20	50	215	2.50	10.75
24	56	244	2.33	10.17
30	80	392	2.67	13.07
36	95	461	2.64	12.81
48	132	840	2.75	17.50
60	200	964	3.33	16.07
72	266	1186	3.69	16.47
84	320	1784	3.81	21.24
120	560	2468	4.67	20.57
180	950	4382	5.28	24.34
210	1280	6458	6.10	30.75
240	1056	9696	4.40	40.40
360	2280	11840	6.33	32.89
420	3200	15256	7.62	36.32
504	3648	21844	7.24	43.34
840	7680	39884	9.14	47.48
1008	10032	56360	9.95	55.91
1260	12160	72268	9.65	57.36
2520	29184	190148	11.58	75.46

Рис. 7.5. Характеристики усиленного алгоритма Агарвала—Кули вычисления свертки.

Вычисления по модулю $x^4 - 1$ содержат 15 сложений. Таким образом, перебирая все три составляющие, для полного числа сложений в рассматриваемом двумерном алгоритме получаем

$$A(20) = (4 \cdot 0 + 1 \cdot 15) + (4 \cdot 15 + 5 \cdot 16) + (15 \cdot 4) = 215,$$

что меньше, чем 230 сложений, необходимых в чистом алгоритме Агарвала—Кули.

Заметим, что метод разложения и гнездовой алгоритм не вызывают заметного усложнения организации вычислений. Все изменения сводятся к повторным переходам от вычислений, связанных со сложениями по строкам, к вычислениям, связанным со сложениями по столбцам, и обратно. Это можно считать простым изменением последовательности обращений к вызываемым подпрограммам.

Чтобы на основании китайской теоремы об остатках добиться дальнейшего уменьшения числа умножений, надо ввести еще одну идею: надо подняться в такое расширение поля, в котором задача начнет распадаться. Можно допустить разложение многочлена $q(y)$ на взаимно простые множители, коэффициенты которых зависят от неопределенной переменной x. (Говоря формально, можно рассмотреть разложение многочлена $q(y)$ в расширении поля, получаемом присоединением формальной переменной x.) Это приведет к тому, что многочлен $q(y)$ может распасться на множители, степени которых меньше возникавших ранее в одномерных задачах степеней. Следовательно, могут быть построены алгоритмы с лучшими характеристиками. При этом символ x появится в свертке по y, так что сама по себе свертка по y становится более сложной. Но при переходе в алгоритм свертки по x возникающий при разложении $q(y)$ символ x может алгебраически взаимодействовать с символом x в свертке по x, что упрощает двумерный алгоритм.

Поясним идею на конкретном примере. Пусть надо вычислить двумерную циклическую свертку

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^4 - 1} \pmod{y^4 - 1}.$$

Разобьем задачу на две подзадачи:

$$s^{(0)}(x, y) = g^{(0)}(x, y)\, d^{(0)}(x, y) \pmod{y^4 - 1} \pmod{x^2 - 1}$$

и

$$s^{(1)}(x, y) = g^{(1)}(x, y)\, d^{(1)}(x, y) \pmod{y^4 - 1} \pmod{x^2 + 1}.$$

Решение первой подзадачи уже рассматривалось; алгоритм вычисления этой свертки содержит десять умножений, если его построить на основании 4-точечного и 2-точечного алгоритмов циклических сверток, содержащих 5 и 2 умножения соответственно.

Если для решения второй подзадачи воспользоваться лучшими известными алгоритмами вычисления произведений по модулю $y^4 - 1$ и по модулю $x^2 + 1$ соответственно, требующими 15 умножений, то получим в результате алгоритм, содержащий 25 умножений, — тот же результат, который получится, если использовать по обоим измерениям алгоритм 4-точечной циклической свертки.

Вместо этого рассмотрим разложение

$$y^4 - 1 = (y - 1)(y + 1)(y - x)(y + x) \pmod{x^2 + 1}.$$

Такая запись осмысленна, поскольку по модулю $x^2 + 1$ выполняется равенство $x^2 = -1$. Теперь можно построить алгоритм вычисления по модулю $y^4 - 1$ с четырьмя умножениями. Каждое из умножений представляет собой умножение многочленов от x первой степени по модулю $x^2 + 1$. Но умножения многочленов первой степени в любом случае входят в алгоритм. Используя разложение $y^2 + 1 = (y - x)(y + x)$, мы как бы частично уменьшаем зависимость по y, перенося ее в зависимость по x и перекладывая часть работы в вычисления по x, которые приходится выполнять в любом случае.

Этот путь позволяет вычислить $s^{(1)}(x, y)$ за 12 умножений, так что исходный алгоритм циклической (4×4)-свертки будет содержать 22 умножения.

Идею можно продолжить дальше, вводя еще больше формальных переменных в качестве корней многочлена $y^n + 1$. В пределе этот метод приведет к вычислению преобразования Фурье в поле разложения этого многочлена, записанном в полиномиальном представлении. Мы сейчас оставим эту тему, но встретимся с ней в измененном виде позже в разд. 7.5 при рассмотрении полиномиальных представлений полей расширения.

7.4. Итеративные алгоритмы

Итеративные методы построения алгоритмов свертки базируются на том, что малые алгоритмы свертки, независимо от того, в каком именно поле F они были построены, на самом деле представляют собой некоторые тождества, справедливые в любом содержащем поле F кольце. Окончательный алгоритм не предполагает коммутативности умножения и не содержит делений, за исключением деления на некоторые малые константы. В качестве операций над входными переменными допускаются только сложения, вычитания и умножения.

Сначала рассмотрим линейную одномерную 4-точечную свертку $s(x) = g(x)\, d(x)$, где степень многочленов $g(x)$ и $d(x)$ равна трем. Расставим скобки по правилу:

$$g(x) = (g_3 x + g_2) x^2 + (g_1 x + g_0),$$
$$d(x) = (d_3 x + d_2) x^2 + (d_1 x + d_0).$$

Определим следующие многочлены от двух переменных:

$$g(y, z) = (g_3 y + g_2) z + (g_1 y + g_0),$$
$$d(y, z) = (d_3 y + d_2) z + (d_1 y + d_0),$$
$$s(y, z) = g(y, z)\, d(y, z).$$

Тогда искомая свертка получается из многочлена $s(y, z)$ по правилу

$$s(x) = s(x, x^2).$$

В сокращенной форме эти вычисления могут быть записаны равенством

$$s_2(y) z^2 + s_1(y) z + s_0(y) = (g_1(y) z + g_0(y))\,(d_1(y) z + d_0(y)),$$

в котором все коэффициенты представляют собой многочлены от y первой степени.

Алгоритм 2-точечной линейной свертки записывается в виде

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_0 + g_1 \\ g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}.$$

Эти тождества справедливы даже тогда, когда входные переменные являются элементами кольца многочленов. Соответственно

$$\begin{bmatrix} s_0(y) \\ s(y) \\ s_2(y) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0(y) \\ g_0(y) + g_1(y) \\ g_1(y) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0(y) \\ d_1(y) \end{bmatrix}.$$

Здесь имеются три умножения многочленов и три сложения многочленов, исключая сложения, относящиеся к коэффициентам многочлена $g(x)$. Каждое произведение многочленов само является 2-точечной линейной сверткой и, следовательно, требует три умножения и три сложения. Таким образом, полное число умножений в итеративном алгоритме 4-точечной линейной свертки равно 9, тогда как оптимальный алгоритм содержит только семь умножений. Преимущество итеративного алгоритма состоит в уменьшении числа сложений: исключая сложения, касающиеся коэффициентов многочлена $g(x)$, он содержит всего 15 сложений.

Полученный алгоритм 4-точечной свертки можно итерировать опять и построить алгоритм 16-точечной свертки, содержащий 81 умножение и 195 сложений (5.06 умножений и 12.19 сложений на одну компоненту на выходе). Этот алгоритм можно в свою очередь итерировать опять для построения 256-точечной свертки алгоритма с 6561 умножениями и 18 915 сложениями (25.63 умножений и 73.89 сложений на одну компоненту на выходе).

В общем случае для вычисления $s(x) = g(x)\, d(x)$ можно использовать итерацию, если число коэффициентов многочлена $g(x)$ и число коэффициентов многочлена $d(x)$ имеют общий делитель. Пусть $\deg d(x) = MN - 1$ и $\deg g(x) = ML - 1$. Преобразуем многочлен $d(x)$ в многочлен от двух переменных $d(y, z)$, полагая

$$d(y, z) = \sum_{l=0}^{N-1} \left(\sum_{k=0}^{M-1} d_{Ml+k} y^k \right) z^l,$$

где старый индекс i связан с парой (k, l) новых индексов равенством $i = Ml + k$, $l = 0, \ldots, N - 1$, $k = 0, \ldots, M - 1$. Аналогично, многочлен $g(x)$ также преобразуем в многочлен от двух переменных $g(y, z)$, полагая

$$g(y, z) = \sum_{l=0}^{L-1} \left(\sum_{k=0}^{M-1} g_{Ml+k} y^k \right) z^l,$$

где $i = Ml + k$, $l = 0, \ldots, L - 1$ и $k = 0, \ldots, M - 1$. Вычислим произведение $s(y, z) = g(y, z)\, d(y, z)$. Тогда $s(x)$ можно вычислить согласно равенству

$$s(x) = s(x, x^M),$$

для чего требуются только сложения. Этот двумерный алгоритм свертки основан на алгоритме линейной свертки последовательностей длины L и N и алгоритме M-точечной линейной свертки. Число необходимых умножений равно произведению чисел необходимых умножений в двух составляющих малых алгоритмах линейных сверток.

Один из двух составляющих алгоритмов может быть построен с помощью итерации. В общем случае этот процесс можно использовать любое число раз. Более того, даже если числа коэффициентов многочленов $g(x)$ и $d(x)$ не имеют общего делителя, то это легко подправить, введя в один из них члены с нулевыми коэффициентами, и воспользоваться итерацией.

Итерацию можно использовать и для вычисления произведения

$$s(x) = g(x)\, d(x) \pmod{m(x)}.$$

Простейший способ состоит в вычислении линейной свертки с последующим приведением по модулю многочлена $m(x)$. Но обычно удается добиться лучшего, если усложнить процедуру.

Мы рассмотрим только случай, когда $m(x)$ равно $x^n + 1$, причем n представляет собой некоторую степень двойки. Это важный случай, так как

$$x^{2n} - 1 = (x^n - 1)(x^n + 1),$$

так что произведение многочленов по модулю $x^n + 1$ возникает и тогда, когда надо вычислить $2n$-точечную циклическую свертку, опираясь на китайскую теорему об остатках.

Для вычисления

$$s(x) = g(x)\, d(x) \pmod{x^n + 1}$$

заменим многочлен $d(x) = \sum_{i=0}^{n-1} d_i x^i$ от одной переменной многочленом от двух переменных, полагая $n = n'n''$:

$$d(y, z) = \sum_{i''=0}^{n''-1} \sum_{i'=0}^{n'-1} d_{i'+n'i''} y^{i'} z^{i''}.$$

Исходный многочлен получается из этого многочлена подстановкой $y = x$ и $z = x^{n'}$. Аналогично определим $g(y, z)$ и положим

$$s(y, z) = g(y, z)\, d(y, z) \pmod{z^{n''} + 1}.$$

Искомый многочлен $s(x)$ получается из этого многочлена по правилу

$$s(x) = s(x, x^{n'}) \pmod{x^n + 1},$$

не требующему никаких умножений. Следовательно, наша задача свелась к вычислению многочлена $s(y, z)$.

Рассмотрим линейную свертку $s(y) = g(y)\, d(y)$, в которой коэффициенты многочленов $s(y)$, $g(y)$ и $d(y)$ в свою очередь представляют собой многочлены от z и умножение этих многочленов от z понимается как умножение по модулю $z^{n''} + 1$.

Возможно, эту процедуру легче понять, если выбрать $n'' = 2$ и заменить z на j. Тогда проделанный шаг означает замену произведения вещественных многочленов по модулю $x^n + 1$ произведением комплексных многочленов по модулю $x^{n/2} + 1$. Преимущество, даваемое таким шагом, состоит в возможности применения алгоритма Кука—Тоома с корнями в точках $\pm j$. В общем случае можно z понимать как корень степени n'' из -1.

Воспользуемся алгоритмом Кука—Тоома, допуская в качестве «корней» степени формальной переменной z. (Формальное утверждение состоит в том, что мы выбираем корни в расширении поля, получаемом присоединением z.) Тогда при приведении по модулю $y - z^l$ коэффициенты многочленов $g(y)$ и $d(y)$ становятся многочленами от z. Так как они уже являются многочленами от z, то проделанный шаг не приводит к какому бы то ни было усложнению, если позаботиться о том, чтобы степень этих многочленов от z не начинала превышать свою первоначальную границу. Это приводит к некоторому неравенству, связывающему делители n' и n'' числа n.

Алгоритм Кука—Тоома можно использовать для вычисления линейной свертки, если корни выбранного многочлена расположены в точках $\pm z^l$ при l меньших n''. Степень многочлена

$$m(y) = y(y - \infty) \prod_{l=0}^{n''-1} (y - z^l)(y + z^l)$$

равна $2n'' + 2$, так что его можно использовать для вычисления свертки $s(y)$ степени не более $2n'' + 1$. Если степень многочлена $s(y)$ меньше, то часть множителей можно из $m(y)$ отбросить. Мы будем пользоваться многочленом $m(y)$, удовлетворяющим условию $\deg m(y) = \deg s(y) + 1 = 2n' - 1$.

Так как все входящие в $m(y)$ множители являются многочленами первой степени, то для вычисления каждого из коэффициентов многочлена $s(y)$ требуется только одно умножение, которое, конечно, представляет собой умножение многочленов от z по модулю $z^{n''} + 1$.

Итеративный алгоритм применим, если $2n' - 1 < 2n'' + 1$. Если это условие выполнено, то вычисление произведения многочленов по модулю $x^n + 1$ разбивается на вычисления $2n' - 1$ произведений многочленов по модулю $x^{n''} + 1$. В свою очередь, это произведение повторением описанной процедуры может быть разбито на еще более мелкие подзадачи так, как это показано на рис. 7.6. (На диаграмме выпущены детали, связанные с критерием выбора разложения $n'n'' = n$ и с общими правилами предсложений и постсложений.) Число необходимых в алгоритме умножений и сложений описывается рекуррентными равенствами

$$M(n) = (2n' - 1)\, M(n''),$$

$$A(n) = (2n' - 1)\, A(n'') + A_1(n) + A_2(n),$$

где $A_1(n)$ обозначает число предсложений, связанных с приведением $d(y)$ по модулю множителей многочлена $m(y)$, а $A_2(n)$ обозначает число сложений, необходимых для восстановления многочлена $s(y)$ по его вычетам. Истинные значения величин $A_1(n)$ и $A_2(n)$ зависят от выбора $2n' - 1$ множителей разложения многочлена $m(y)$.

Например, вычисление произведения многочленов по модулю $x^4 + 1$ можно свести к вычислению трех произведений многочленов по модулю $x^2 + 1$, каждое из которых требует трех умножений; это дает алгоритм, содержащий девять умножений. Аналогично, вычисление произведения по модулю $x^{16} + 1$ можно свести к вычислению семи произведений многочленов по модулю $x^4 + 1$, что приведет к алгоритму с полным числом умножений, равным 63 или 49 в зависимости от выбора алгоритма вычисления произведения многочленов по модулю $x^4 + 1$. Характеристики описанного итеративного алгоритма в зависимости от выбора составляющих подалгоритмов приведены на рис. 7.7. Варианты определяются способом разложения длины n на множители n' и n''; n'' можно разложить далее. Все они выписаны в последнем столбце таблицы.

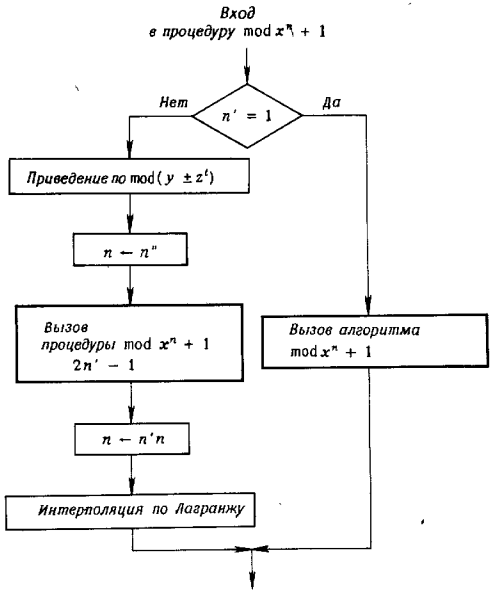


Рис. 7.6. Принципиальная схема итеративного алгоритма.

Несколько видоизмененная версия алгоритма получается, если выбрать

$$m(x) = \prod_{l=0}^{n'-1} (x - z^l)(x + z^l) = \prod_{l=0}^{2n'-1} (x - z^l) \pmod{z^{n''} + 1}.$$

Теперь степень основного модуля $m(x)$ равна $2n''$, хотя достаточной является степень $2n'' - 1$. Следовательно, такой выбор многочлена $m(x)$ приводит к одному дополнительному (по сравнению с необходимым их числом) вычислению произведения многочленов. Однако такая форма алгоритма позволяет при больших n' получить преимущества в числе предсложений и постсложений, если воспользоваться БПФ-алгоритмом Кули—Тьюки по основанию 2. Здесь мы уже продвинулись в исследованиях столь глубоко, что подошли к новому методу, известному под названием полиномиального преобразования. Этому методу посвящена оставшаяся часть данной главы. Мы прервем нить изложения, которой следовали до сих пор, так как полиномиальное преобразование должно быть введено более фундаментальным образом.

Модули	Число умножений	Число сложений	Итеративная ($n' \times n''$) схема
$x^2 + 1$	3	3	–
$x^4 + 1$	9	15	2 × 2
	7	41	–
$x^8 + 1$	27	57	2 × (2 × 2)
	21	77	2 × 4
$x^{16} + 1$	63	205	4 × (2 × 2)
	49		4 × 4
$x^{32} + 1$	189	599	4 × (2 × (2 × 2))
	147	739	4 × (2 × 4)
$x^{64} + 1$	405	1599	8 × (2 × (2 × 2))
	315	1899	8 × (2 × 4)
$x^{128} + 1$	945	4563	8 × (4 × (2 × 2))
$x^{256} + 1$	1953	10531	16 × (4 × (2 × 2))
$x^{512} + 1$	5859	26921	16 × (4 × (2 × (2 × 2)))
$x^{1024} + 1$	11907	58889	32 × (4 × (2 × (2 × 2)))
$x^{2048} + 1$	25515	143041	32 × (8 × (2 × (2 × 2)))
$x^{4096} + 1$	51435	304769	64 × (8 × (2 × (2 × 2)))

Рис. 7.7. Характеристики некоторых алгоритмов вычисления произведения многочленов по модулю $x^n + 1$.

7.5. Полиномиальное представление расширений полей

Наиболее часто применяемое дискретное преобразование Фурье длины n отображает вектор с вещественными компонентами в вектор с комплексными компонентами. В практических задачах обработки дискретных сигналов векторы с произвольными вещественными значениями компонент никогда не нужны; всегда оказывается достаточным ограничить рассмотрение рациональными числами, или, даже еще проще, целыми. В данном разделе область определения преобразования Фурье ограничена множеством рациональных чисел. Это ограничение не снижает практической ценности алгоритмов, но при этом оно приводит к совершенно другому взгляду на характер вычислений.

Преобразование Фурье длины n отображает вектор длины n с рациональными компонентами в вектор длины n, компоненты

которого принадлежат некоторому подмножеству комплексных чисел, на самом деле счетному подмножеству. Обычно в исследованиях преобразования Фурье в качестве значений компонент выходного вектора допускаются произвольные комплексные числа, которые в практических вычислениях ограничиваются конечным множеством путем фиксации разрядности допустимых чисел. Это можно сделать до некоторой степени произвольно; условность представления проявляется в выборе необходимой длины слова с учетом допустимой погрешности округления. Даже тогда, когда областью определения преобразования Фурье является множество целых чисел, для точного вычисления преобразования необходимо, чтобы разрядность двоичного представления числа была бесконечной.

В настоящем разделе рассматривается другой подход к заданию областей определения и значения преобразования Фурье. Он состоит в полиномиальном описании расширений полей рациональных или комплексных чисел.

Пусть $\mathbf{v}$ представляет собой вектор длины n с рациональными компонентами и пусть $\omega = e^{-j2\pi/n}$. Тогда задаваемый преобразованием Фурье вектор

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \quad k = 0, \ldots, n-1,$$

имеет комплексные компоненты. Тем не менее произвольное комплексное число не может быть результатом преобразования. Все возможные в результате вычисления преобразования Фурье компоненты принадлежат подполю $\mathbb{Q}(\omega)$, или, в более простых обозначениях, подполю $\mathbb{Q}^n$, которое представляет собой наименьшее содержащее ω подполе поля комплексных чисел. Это подполе содержит поле рациональных чисел, поскольку любое подполе поля комплексных чисел содержит все рациональные числа.

Пусть над полем рациональных чисел разложение многочлена $x^n - 1$ на простые множители дается равенством

$$x^n - 1 = p_0(x)\, p_1(x) \ldots p_s(x).$$

Простые делители являются круговыми многочленами, и для малых n их коэффициенты равны -1, 0, 1. Так как элемент ω является корнем многочлена $x^n - 1$, то он должен быть корнем одного из круговых многочленов, скажем, корнем многочлена $p(x)$ степени m со старшим коэффициентом, равным единице:

$$p(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1 x + p_0.$$

Так как $p(\omega) = 0$, то

$$\omega^m = -p_{m-1}\omega^{m-1} - \ldots - p_1\omega - p_0.$$

Следовательно, ω^m может быть выражен в виде линейной комбинации меньших степеней элемента ω. Для меньших степеней

элемента ω такое представление невозможно, так как в противном случае ω был бы корнем многочлена степени меньше, чем степень многочлена $p(x)$.

Поле $\mathbb{Q}^m$ может быть задано как множество всех многочленов от ω с рациональными коэффициентами степеней, не превосходящих $m-1$. Операцией сложения в таком представлении поля является сложение многочленов, а операцией умножения — умножение многочленов по модулю многочлена $p(\omega)$. Многочлены задаются списком своих m коэффициентов. Такой способ задания элементов поля $\mathbb{Q}^m$ требует m слов памяти вместо двух, необходимых для запоминания комплексного числа.

Чтобы подчеркнуть, что сами числа задаются многочленами, а не их комплексными значениями в точке ω, будем в записи чисел пользоваться символом x вместо ω. Тогда числа поля равны

$$a = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1 x + a_0$$

и даются списком коэффициентов a_i. Конечно, если мы пожелаем узнать «истинное» комплексное значение числа a, то надо вместо x в многочленное выражение для a подставить ω и произвести соответствующие этому многочлену вычисления. Однако наша цель состоит в разработке алгоритмов, внутренние переменные которых имеют полиномиальное представление, и в некоторых отношениях такая форма действительно приводит к более простым алгоритмам.

Например, если n равно степени двух, то

$$(x^n - 1) = (x^{n/2} + 1)(x^{n/4} + 1) \ldots (x + 1)(x - 1).$$

Круговой многочлен $x^{n/2} + 1$ приводит к расширению поля $\mathbb{Q}$, все элементы которого представляют собой многочлены с рациональными коэффициентами степени меньше, чем $n/2$. Сложением в поле является сложение многочленов, а умножением — умножение многочленов по модулю $x^{n/2} + 1$.

Например, поле $\mathbb{Q}^8$ состоит из всех многочленов с рациональными коэффициентами степени не более семи с арифметикой, выполняемой по модулю многочлена $x^8 + 1$. Пример умножения в поле дается равенством $\left(x^7 - \frac{1}{2}x^3 + \frac{1}{4}\right)(x^2 - 1) = x^9 - x^7 - \frac{1}{2}x^4 + \frac{3}{4}x^2 - \frac{1}{4} = -x^7 - \frac{1}{2}x^4 + \frac{3}{4}x^2 - x - \frac{1}{4}$.

Полиномиальное представление расширения полей относится не только к полю рациональных чисел. Его можно использовать и для расширений поля комплексных чисел. Например, поле *комплексных рациональных чисел* — это множество комплексных чисел вида $v = a + jb$, где a и b — рациональные числа. Это подмножество является подполем поля комплексных чисел, которое иногда обозначается через $\mathbb{Q}(j)$. Во многих приложениях цифровой обработки сигналов компоненты обрабатываемых век-

торов являются комплексными. Так как разрядность записи комплексных чисел практически всегда ограничена, то по существу всегда рассматриваются векторы с комплексными рациональными компонентами (или, даже более частный случай, с комплексными целыми компонентами).

Поле комплексных рациональных чисел можно расширить, присоединив корень ω степени n из единицы. Это дает наименьшее расширение $\mathbb{Q}(\omega)$, в котором имеется преобразование Фурье длины n. Если $\mathbb{Q}(\omega)$ содержит j (т. е. содержит корень многочлена x^2+1), то такое расширение $\mathbb{Q}(\omega)$ содержит все комплексные рациональные числа. Это именно то расширение, которое нам нужно. Если круговой многочлен равен x^r+1 для некоторого четного числа r, то это всегда так. В противном случае элемент j надо присоединять.

Пусть n не равно степени двойки, ω обозначает корень степени n из единицы, а $p(x)$ обозначает круговой многочлен степени m, корнем которого ω является. Тогда расширение $\mathbb{Q}(j, \omega)$, или, в более простых обозначениях, $\mathbb{Q}(j)^m$, представляет собой множество многочленов с коэффициентами из поля $\mathbb{Q}(j)$, степени которых не превосходят $m-1$. Сложение в поле совпадает со сложением многочленов, а умножение — с умножением многочленов по модулю $p(x)$. Коэффициенты этих многочленов складываются и вычитаются как комплексные числа.

Для такого задания одного элемента поля $\mathbb{Q}(j, \omega)$ требуется $2m$ рациональных чисел. С точностью до этого различия и более общих правил сложения и умножения коэффициентов многочленов, все, что будет сказано в следующих разделах главы относительно обработки последовательностей рациональных чисел, справедливо и для последовательностей комплексных рациональных чисел. Таким образом, к полю комплексных рациональных чисел можно больше не возвращаться.

7.6. Свертка в полиномиальных расширениях полей

Если $\mathbf{v}$ — вектор с рациональными компонентами, то компоненты

$$V_k = \sum_{i=0}^{n-1} \omega^{ik} v_i, \qquad k = 0, \ldots, n-1,$$

его преобразования Фурье принадлежат полю $\mathbb{Q}(\omega)$; в более общем случае, если компоненты $\mathbf{v}$ принадлежат $\mathbb{Q}(\omega)$, то компоненты его преобразования Фурье также принадлежат $\mathbb{Q}(\omega)$.

Пусть степень кругового многочлена, корнем которого является ω, равна m. Тогда каждый элемент в $\mathbb{Q}(\omega)$ записывается

в виде многочлена над $\mathbb{Q}$ степени, меньшей m. В полиномиальной записи преобразование Фурье имеет вид

$$V_k = \sum_{i=0}^{n-1} x^{ik} v_i \pmod{p(x)}, \quad k = 0, \ldots, n-1.$$

Эта формула вычисляется проще, так как умножение на x сводится к операциям над индексами, а приведение по модулю $p(x)$ содержит не более m сложений. Можно воспользоваться любым БПФ-алгоритмом, например БПФ-алгоритмом Кули—Тьюки, но все умножения представляют собой умножения на x, и, следовательно, требуют для своего выполнения не более m вещественных сложений и не содержат вещественных умножений. Аналогично, каждое сложение является сложением многочленов и выполняется с m вещественными сложениями.

Применительно ко входу с рациональными компонентами преобразование Фурье можно рассматривать как отображение векторов из многочленов первой степени в векторы из многочленов степени $m-1$. В более общем случае, преобразование Фурье в такой трактовке отображает векторы, компонентами которых служат многочлены степени $m-1$, в векторы из многочленов степени $m-1$.

Справедливость теоремы о свертке не зависит от того, как именно представлены числа в данной арифметической системе. Она применима в равной степени как к векторам, компоненты которых записаны в полиномиальной форме, так и к векторам, компоненты которых записаны в обычном виде. Следовательно, для вычисления циклической свертки последовательностей с рациональными компонентами, $s_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k$, можно взять преобразование Фурье векторов $\mathbf{g}$ и $\mathbf{d}$ в поле $\mathbb{Q}^m$, вычислить в частотной области покомпонентное произведение $S_k = G_k D_k$ и взять обратное преобразование Фурье. В полиномиальной записи поля $\mathbb{Q}^m$ оба преобразования Фурье при этом не содержат умножений. Покомпонентное спектральное произведение, однако, теперь является произведением многочленов по модулю $p(x)$, и, следовательно, требует большого числа вещественных умножений. Сложность вычислений переместилась с одного этапа на другой. Усложнение вычисления спектрального произведения сводит на нет выигрыш, полученный в преобразованиях Фурье. Позже мы увидим, что при вычислении двумерной циклической свертки все же можно получить выигрыш от полиномиального представления.

Например, пусть $n = 64$; круговой многочлен равен $p(x) = x^{32}+1$. Элементы поля $\mathbb{Q}(\omega)$ представляют собой многочлены степени 31 и задаются списком из 32 рациональных чисел. Каждое

произведение $S_k = G_k D_k$ является произведением многочленов по модулю $x^{32} + 1$. Этот многочлен неприводим, так что для каждого такого произведения необходимо по меньшей мере $2 \cdot 32 - 1$ вещественное умножение. На самом деле число умножений существенно больше 63; практически используемый 32-точечный алгоритм, приведенный на рис. 7.7, содержит 147 умножений.

Таким образом, каждая из 64 компонент преобразования Фурье требует 147 вещественных умножений. Это существенно больше, чем число вещественных умножений, необходимых при использовании общепринятого для этого случая БПФ-алгоритма.

На самом деле величины S_k не являются произвольными, а должны удовлетворять некоторым ограничениям, связанным с тем, что обратное преобразование Фурье должно иметь рациональные компоненты. 64 компоненты s_i обратного преобразования Фурье представляют собой 64 многочлена, каждый из которых задается 32 рациональными числами. На самом деле, 31 из этих чисел равны нулю, так как каждый из многочленов является многочленом нулевой степени. Это позволяет выписать ограничения, которым должны удовлетворять величины S_k, и окажется, что 64 из 147 этих величин нет надобности вычислять. Детальное выписывание этой процедуры, однако, приводит к алгоритму, очень похожему на алгоритм, основанный на китайской теореме об остатках, который проще выписать непосредственно. Поэтому мы не будет продолжать рассмотрение этой процедуры.

В полиномиальном расширении поля, $\mathbb{Q}^m$, можно определить и более короткие преобразования Фурье. Предположим, что число n является составным: $n = n'n''$. Тогда n'-точечное преобразование Фурье определяется равенством

$$V_k = \sum_{i=0}^{n'-1} (x^{n''})^{ik} v_i, \quad k = 0, \ldots, n' - 1,$$

где арифметика задается как полиномиальная арифметика поля $\mathbb{Q}^m$. Ядро преобразования теперь вместо x равно $x^{n''}$. При таком определении преобразования Фурье, так же как и ранее, существует и обратное к нему преобразование и справедлива теорема о свертке.

7.7. Полиномиальное преобразование Нуссбаумера

Преобразования, построенные как преобразования Фурье в полиномиальных расширениях полей, заслуживают самостоятельного исследования. Мы определили их для поля рациональных чисел, так что коэффициенты всех многочленов рациональны. Так как построенное преобразование обратимо, то оно задает

некоторое множество тождеств для многочленов с рациональными коэффициентами. Тождества остаются справедливыми, даже если в качестве коэффициентов многочленов допустить вещественные (или комплексные) числа.

В данном разделе мы вновь обращаемся к изучению полиномиального преобразования, но на сей раз мы будем его рассматривать не как преобразование Фурье, а как самостоятельное преобразование.

В кольце многочленов по модулю $p(x)$ полиномиальное преобразование определяется равенством

$$V_k(x) = \sum_{i=0}^{n-1} \omega(x)^{ik} v_i(x), \quad k = 0, \ldots, n - 1,$$

где $\omega(x)$ представляет собой элемент порядка n и умножение, конечно, является кольцевым умножением многочленов по модулю $p(x)$. Наше рассмотрение будет ограничено случаем, когда $\omega(x) = x$ и многочлен $p(x)$ делит $x^n - 1$. Как представляется, никакие другие случаи не имеют какого-либо реального практического интереса.

Определение 7.7.1. Пусть $p(x)$ — некоторый многочлен степени m и пусть многочлены $v_i(x)$, $i = 0, \ldots, n - 1$, над полем F имеют степени не более $m - 1$ и представляют собой компоненты полиномиального вектора. *Полиномиальным преобразованием* этого вектора называется вектор с компонентами

$$V_k(x) = \sum_{i=0}^{n-1} x^{ik} v_i(x) \pmod{p(x)}, \quad k = 0, \ldots, n - 1.$$

Полиномиальное преобразование легко вычислить, так как оно не содержит умножений элементов кольца общего вида. Умножения на x^{ik} тривиальны, и, если многочлен $p(x)$ выбран так, что все его коэффициенты равны 0, 1 или -1, то приведение по модулю $p(x)$ также выполняется одними сложениями.

Ценность полиномиального преобразования обуславливается двумя теоремами: теоремой о существовании обратного преобразования и теоремой о свертке. Оба эти утверждения, конечно, сразу вытекают из возможности интерпретации полиномиального преобразования как преобразования Фурье, подобно тому, как это делалось в предыдущем разделе. Тем не менее поучительно дать более прямое доказательство.

Пусть n обозначает наименьшее целое число, для которого $p(x)$ делит $x^n - 1$. В доказательстве следующей теоремы используется тот факт, что если многочлен $a(x)$ делит $x^n - 1$, то

для произвольного многочлена $f(x)$ выполняется равенство

$$R_{a(x)}[f(x)] = R_{a(x)}[R_{x^n-1}[f(x)]].$$

Начнем с простейшего случая, когда число n просто.

Теорема 7.7.2. *Предположим, что многочлен $p(x)$ над полем F степени m делит многочлен $x^n - 1$, причем число n является простым. Тогда полиномиальный вектор $\mathbf{v}(x)$ длины n, компонентами которого являются многочлены степени $m - 1$, и его полиномиальное преобразование $\mathbf{V}(x)$ связаны соотношениями*

$$V_k(x) = \sum_{l=0}^{n-1} x^{lk} v_l(x) \pmod{p(x)}, \quad k = 0, \ldots n-1,$$

$$v_i(x) = \frac{1}{n} \sum_{k=0}^{n-1} x^{(n-1)\,ik} V_k(x) \pmod{(p(x))}, \quad i = 0, \ldots, n-1.$$

Доказательство. Исключая тривиальный случай, можно полагать, что степень многочлена $p(x)$ равна по меньшей мере двум. Вычислим правую часть второго равенства:

$$\frac{1}{n}\sum_{k=0}^{n-1} x^{(n-1)\,ik} V_k(x) = \frac{1}{n}\sum_{k=0}^{n-1} x^{(n-1)\,ik}\left[\sum_{l=0}^{n-1} x^{lk} v_l(x)\right] \pmod{p(x)} =$$

$$= \frac{1}{n}\sum_{l=0}^{n-1}\left[v_l(x)\sum_{k=0}^{n-1} x^{(l+ni-i)\,k}\right] \pmod{p(x)}.$$

Вычислим внутреннюю сумму. Если i равно l, то так как $p(x)$ делит $x^n - 1$, имеем:

$$\frac{1}{n}\sum_{k=0}^{n-1} x^{(l+ni-i)\,k} = \frac{1}{n}\sum_{k=0}^{n-1} x^{nik} = \frac{1}{n}\sum_{k=0}^{n-1} 1 \pmod{x^n - 1} =$$

$$= 1 \pmod{p(x)}.$$

При i, не равном l, воспользуемся тождеством, справедливым в кольце:

$$(1 - x^r)\sum_{k=0}^{n-1} x^{rk} = 1 - x^{rn}.$$

Так как $r = l + ni - i$ не кратно n, то

$$1 - x^r \neq 0 \pmod{x^n - 1},$$

в то время, как

$$1 - x^{rn} = 0 \pmod{x^n - 1}.$$

Так как $p(x)$ делит $x^n - 1$, то отсюда вытекает, что

$$\sum_{k=0}^{n-1} x^{rk} = 0 \pmod{p(x)}.$$

Итак, мы доказали, что

$$\frac{1}{n}\sum_{k=0}^{n-1} x^{(l+ni-i)\,k} = \begin{cases} 1, & \text{если } l = i \pmod{p(x)}, \\ 0, & \text{если } l \neq i \pmod{p(x)}. \end{cases}$$

Следовательно,

$$\frac{1}{n}\sum_{k=0}^{n-1} x^{(n-1)\,ik} V_k(x) = v_i(x) \pmod{p(x)},$$

что и завершает доказательство. □

Так как $x^n = 1$, то обратное полиномиальное преобразование может быть записано в более простом виде

$$v_i(x) = \frac{1}{n}\sum_{k=0}^{n-1} x^{-ik} V_k(x) \pmod{p(x)},$$

хотя формально x^{-ik} не является многочленом. Точно так же, как умножение на x выполняется с помощью переиндексации коэффициентов и приведения x^m по модулю $p(x)$, умножение на x^{-1} также можно выполнить с помощью переиндексации коэффициентов и приведения по модулю $p(x)$, причем для исключения x^{-1} надо воспользоваться тем, что $p_m x^m + p_{m-1}x^{m-1} + \cdots + p(x) + p_0 = 0$, так что

$$x^{-1} = p_0^{-1}[p_m x^{m-1} + p_{m-1}x^{m-2} + \ldots + p_1].$$

Следовательно, обратное полиномиальное преобразование вычисляется столь же просто, сколь и прямое полиномиальное преобразование.

Теорема 7.7.3 (теорема о свертке). *Если в кольце многочленов по модулю $p(x)$ вектор многочленов с компонентами $s_i(x)$, $i = 0, \ldots, n-1$, связи с векторами многочленов, имеющими компоненты $g_i(x)$ и $d_i(x)$, $i = 0, \ldots, n-1$, соотношением циклической свертки многочленов*

$$s_i(x) = \sum_{l=0}^{n-1} g_{i-l}(x) d_l(x) \pmod{p(x)},$$

то полиномиальный спектр $S_k(x)$ связи со спектрами $G_k(x)$ и $D_k(x)$ покомпонентным произведением многочленов

$$S_k(x) = G_k(x) D_k(x) \pmod{p(x)}, \quad k = 0, \ldots, n-1.$$

Доказательство. Используя обратное полиномиальное преобразование, имеем

$$s_i(x) = \sum_{l=0}^{n-1} g_{i-l}(x)\, d_l(x) = \frac{1}{n} \sum_{l=0}^{n-1} \sum_{k=0}^{n-1} x^{(n-1)(i-l)k} G_k(x)\, d_l(x) =$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} x^{(n-1)ik} G_k(x) \sum_{l=0}^{n-1} x^{lk} d_l(x),$$

где x^{nk} следует положить равным единице, потому что, как и ранее, все вычисления производятся по модулю $x^n - 1$. Следовательно,

$$s_i(x) = \frac{1}{n} \sum_{k=0}^{n-1} x^{(n-1)ik} G_k(x)\, D_k(x),$$

и $G_k(x)\, D_k(x)$ должны быть равным $S_k(x)$. □

7.8. Быстрая свертка многочленов

Полиномиальное представление расширения поля оказывается полезным при вычислении многомерных сверток. Так как при этом мы исходно имеем дело с произведениями многочленов, то переход от произведений спектров к произведениям многочленов не требует дополнительных вычислений, поскольку их можно включить в ту часть работы, которая все равно должна быть выполнена в любом случае.

Достаточно рассмотреть двумерную циклическую свертку. Запишем эту свертку в виде одномерной циклической свертки многочленов:

$$s_{i'}(y) = \sum_{k'=0}^{n'-1} g_{((i'-k'))}(y)\, d_{k'}(y) \pmod{y^{n'} - 1}.$$

Так как многочлен $y^{n'} - 1$ распадается в произведение круговых многочленов, то эту задачу можно разбить на множество подзадач вида

$$s_{i'}(y) = \sum_{k'=0}^{n'-1} g_{((i'-k'))}(y)\, d_{k'}(y) \pmod{p(y)},$$

где $p(y)$ представляет собой один из круговых многочленов, скажем, степени m', делящих $y^{n'} - 1$. Алгоритм решения каждой из этих подзадач может быть погружен в алгоритм решения исходной задачи.

Рассмотрим элементы $g_{i'}(y)$ и $d_{i'}(y)$ для всех $i = 0, \ldots, n' - 1$ как элементы поля $Q(\omega)$. Тогда для спектров соответственно имеем для всех $k' = 0, \ldots, n' - 1$:

$$G_{k'}(y) = \sum_{i'=0}^{n'-1} y^{i'k'} g_{i'}(y) \pmod{p(y)},$$

$$D_{k'}(y) = \sum_{i'=0}^{n'-1} y^{i'k'} d_{i'}(y) \pmod{p(y)}$$

и

$$S_{k'}(y) = G_{k'}(y)\, D_{k'}(y) \pmod{p(y)}, \quad k' = 0, \ldots, n' - 1.$$

Используя обратное преобразование Фурье, получаем

$$s'_i(y) = \frac{1}{n'} \sum_{k'=0}^{n'-1} y^{-i'k'} S_{k'}(y) \pmod{p(y)}.$$

Преобразования Фурье не содержат вещественных умножений.

Все умножения в описанной процедуре сосредоточены в спектральных произведениях $G_{k'}(y)\, D_{k'}(y)$. Всего имеется n' таких произведений многочленов по модулю $p(y)$, каждое из которых содержит по меньшей мере $2m' - 1$ умножений. Таким образом, в общей сложности для вычисления двумерной свертки, содержащей $n'm'$ чисел, требуется $n'(2m' - 1)$ умножений. На самом деле число умножений, необходимых для вычисления произведения многочленов по модулю $p(y)$, существенно больше, чем $2m' - 1$, но все же достаточно мало для того, чтобы приводить к эффективным алгоритмам. Некоторые подходящие для этой цели алгоритмы приведены на рис. 7.7.

Рассмотрим теперь двумерную циклическую свертку, записанную в виде произведения многочленов:

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^{n'} - 1} \pmod{y^{n''} - 1}.$$

Используя китайскую теорему об остатках, разобьем эту задачу на подзадачи. Детально остановимся только на частном случае, когда n' и n'' равны степеням двойки (и могут совпадать). Метод вычисления является рекуррентным, основанным на использовании двумерных циклических сверток меньшего размера. Таким образом, с помощью рекурсии двумерная циклическая свертка по основанию 2 сводится к некоторому числу меньших подзадач.

Пусть $n' = 2^{m'}$ и $n'' = 2^{m''}$ при $m'' \geqslant m'$. Тогда имеет место разложение

$$y^{n''} - 1 = (y^{n''/2} + 1)(y^{n''/2} - 1) =$$

$$= (y^{n''/2} + 1) \ldots (y^{n'/2} + 1)(y^{n'/2} - 1).$$

Вообще говоря, последний член можно разлагать и дальше, но мы здесь остановимся. Обозначим члены разложения в правой

части соответственно через $f_{R-1}(y)$, ..., $f_0(y)$ и для $r = 0, \ldots, R-1$ определим

$$d^{(r)}(x, y) = d(x, y) \pmod{f_r(y)},$$
$$g^{(r)}(x, y) = g(x, y) \pmod{f_r(y)}$$

и

$$s^{(r)}(x, y) = s(x, y) \pmod{f_r(y)}.$$

Тогда

$$s^{(r)}(x, y) = g^r(x, y)\, d^r(x, y) \pmod{x^{n'} - 1} \pmod{f_r(y)},$$

и, согласно китайской теореме об остатках для многочленов,

$$s(x, y) = \sum_{r=0}^{R-1} a^{(r)}(y)\, s^{(r)}(x, y) \pmod{x^{n''} - 1},$$

где $a^{(r)}(y)$ при $r = 0, \ldots, R-1$ определяются этой же теоремой. Вычисления на этом последнем шаге не содержат вещественных умножений.

Основная часть вычислений падает на вычисление остатков $s^{(r)}(x, y)$. Эти вычисления разбиваются на два типа, а именно,

$$s^{(r)}(x, y) = g^{(r)}(x, y)\, d^{(r)}(x, y) \pmod{x^{n'} - 1} \pmod{y^{n'} + 1}$$

и

$$s^{(r)}(x, y) = g^{(r)}(x, y)\, d^{(r)}(x, y) \pmod{x^{n'} - 1} \pmod{y^{n'/2} - 1}.$$

Второй тип вычислений, с точностью до замены x и y друг на друга, представляет собой копию решаемой задачи меньшего размера. Опираясь на приведенную на рис. 7.8 рекуррентную формулу, можно полагать, что эта меньшая задача решена, поскольку решена задача большего размера.

Опуская индекс r, первый тип вычислений запишем в виде задачи

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^{n'} - 1} \pmod{y^{m'} + 1},$$

где $m' \geqslant n'/2$. Эту задачу можно рассматривать как задачу вычисления одномерной свертки в поле $\mathbb{Q}^{m'}$, а именно,

$$s(x) = g(x)\, d(x) \pmod{x^{n'} - 1},$$

где все коэффициенты выписанных многочленов представляют собой элементы поля $\mathbb{Q}^{m'}$ и, следовательно, записываются в виде многочленов от y степени, меньшей m'. Так как $n' \leqslant 2m'$, то для вычисления свертки можно воспользоваться имеющимся преобразованием Фурье длины n'. Преобразование Фурье выполняется с помощью одних сложений и не содержит умножений.

В частотной области свертка преобразуется к произведениям

$$S_{k'} = G_{k'} D_{k'}.$$

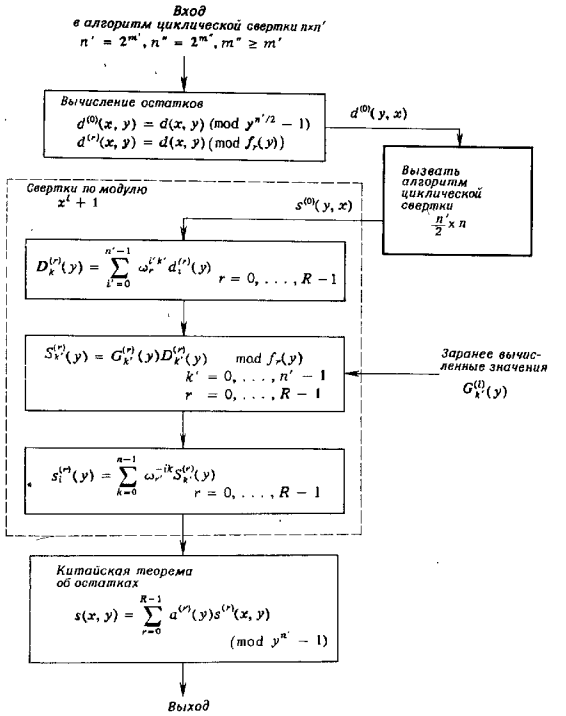


Рис. 7.8. Алгоритм многомерной циклической свертки.

Для их вычисления нужно n' умножений в $\mathbb{Q}$, каждое из которых представляет собой произведение многочленов по модулю $y^{m'} + 1$ и поэтому требует $2m' - 1$ вещественных умножений. Следовательно, для вычисления

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^{n'} - 1} \pmod{y^{m'} + 1}$$

Размер таблицы	Число вещественных умножений	Число вещественных сложений	Число вещественных умножений на точку	Число вещественных сложений на точку
3 × 3	13	70	1.44	7.78
4 × 4	22	122	1.37	7.62
5 × 5	55	369	2.20	14.76
6 × 6	52	424	1.44	11.78
7 × 7	121	1163	2.47	23.73
8 × 8	130	750	2.03	11.72
9 × 9	193	1382	2.38	17.06
10 × 10	220	1876	2.20	18.76
14 × 14	484	5436	2.47	27.73
16 × 16	634	4774	2.48	18.65
18 × 18	772	6576	2.38	20.30
24 × 24	1402	12954	2.43	22.49
27 × 27	2893	21266	3.97	29.17
32 × 32	3658	24854	3.57	24.27
64 × 64	17770	142902	4.34	34.89
128 × 128	78250	720502	4.78	43.98

Рис. 7.9. Характеристики некоторых алгоритмов вычисления двумерной циклической свертки.

необходимо всего $n'(2m'-1)$ вещественных умножений, а для вычисления исходной двумерной циклической свертки требуется сделать несколько таких вычислений.

Например, для вычисления двумерной циклической (64×64)-свертки требуется $64\cdot(63)$ вещественных умножений и вычисления двумерной циклической (64×32)-свертки. В свою очередь, для вычисления двумерной циклической (64×32)-свертки требуется в идеале $32\cdot(63)+32\cdot(31)$ вещественных умножений и вычисления одной двумерной циклической (16×32)-свертки. В свою очередь, вычисление двумерной циклической (32×16)-свертки требует $16\cdot(31)+16\cdot(15)$ умножений и свертки еще меньшего размера. Продолжая этот процесс до тех пор, пока не дойдем до тривиальной свертки, получаем около 800 умножений. Это существенно меньше, чем число умножений, необходимое при использовании двумерного алгоритма Кули—Тьюки и теоремы о свертке, равное 73 728.

Практические алгоритмы вычисления произведений многочленов, как видно из затабулированных на рис. 7.7 данных, содержат большее число умножений, чем равный $n'(2m'-1)$ теоретический минимум. В частности, рассмотренный алгоритм вычисления двумерной циклической (64×64)-свертки содержит на самом деле 17 770 вещественных умножений. Характеристики некоторых других практических алгоритмов вычисления двумерной циклической свертки, построенных по описанному методу, приведены на рис. 7.9.

Задачи

7.1. **а.** Вычислить характеристики алгоритма 7560-точечной циклической свертки, построенного на основе алгоритма Агарвала — Кули.
б. Вычислить характеристики гнездового алгоритма двумерной (504×504)-свертки.

7.2. Алгоритм вычисления 12-точечной циклической свертки можно построить прямо по методу Винограда или по алгоритму Агарвала — Кули, сочетая алгоритм 3-точечной циклической свертки с алгоритмом 4-точечной циклической свертки. Сравнить числа умножений, необходимых в каждом из этих методов.

7.3. **а.** Описать метод построения алгоритма вычисления 6-точечной циклической свертки $s(x)=g(x)\,d(x) \pmod{x^6-1}$. Сколько необходимо умножений?
б. Воспользуйтесь алгоритмом Агарвала — Кули построения 6-точечной циклической свертки из алгоритма 2-точечной циклической свертки и алгоритма 3-точечной циклической свертки. Сколько необходимо умножений?

7.4. Воспользовавшись алгоритмом Агарвала — Кули, привести матрицы алгоритма 15-точечной циклической свертки к стандартному виду

$$\mathbf{s}=\mathbf{CGAd}.$$

7.5. Пусть длина циклической свертки равна $n=n_1n_2n_3n_4$, где все делители взаимно просты. Различные алгоритмы вычисления циклической свертки такой длины получаются различными способами сочетания алгоритмов циклических сверток длин n_1, n_2, n_3 и n_4. Две возможные схемы построения алгоритмов даются следующими правилами расстановки скобок: $n=((n_1n_2)(n_3n_4))$ и $n=(n_1(n_2(n_3(n_4))))$. Доказать, что если в каждом парасочетании используемые алгоритмы оптимальны, то обе схемы содержат одинаковое число умножений и сложений.

7.6. Алгоритм вычисления комплексной свертки по модулю x^n-1 описан в разд. 3.7.
а. Построить этот алгоритм как алгоритм двумерного произведения многочленов по модулям $x^{2r}+1$ и y^2+1.
б. Для этой же задачи можно построить два алгоритма, отталкиваясь соответственно от

$$s(x,y)=g(x,y)\,d(x,y) \pmod{x^{2r}+1} \pmod{y^2+1}$$

или от

$$s(x,y)=g(x,y)\,d(x,y) \pmod{y^2+1} \pmod{x^{2r}+1}.$$

Какой из алгоритмов лучше?

7.7. Кольцо кватернионов было определено в задаче 2.15.
а. Записать двумерную циклическую (2×2)-свертку

$$s(x,y)=g(x,y)\,d(x,y) \pmod{x^2-1} \pmod{y^2-1}$$

в виде произведения (4×4)-матрицы на вектор, содержащий 4 компоненты. Дать алгоритм, содержащий четыре умножения.
б. Записать полиномиальное произведение

$$s,(x,y)=g(x,y)\,d(x,y) \pmod{x^2+1} \pmod{y^2-1}$$

в виде произведения (4×4)-матрицы на вектор, содержащий 4 компоненты. Дать алгоритм, содержащий шесть умножений.
в. Записать полиномиальное произведение

$$s(x,y)=g(x,y)\,d(x,y) \pmod{x^2+1} \pmod{y^2+1}$$

в виде произведения (4×4)-матрицы на вектор, содержащий 4 компоненты. Дать алгоритм, содержащий шесть умножений.

г. Записать произведение кватернионов $s = gd$ в виде произведения (4×4)-матрицы на вектор, содержащий 4 компоненты. Дать алгоритм, содержащий девять умножений.

7.8. Определить число умножений и число сложений в алгоритме Агарвала — Кули для вычисления 45-точечной циклической свертки. Какое улучшение дает переход к алгоритму разложения?

7.9. а. Найти характеристики гнездового алгоритма вычисления циклической (15×15)-свертки, сочетающего алгоритм циклической (3×3)-свертки с алгоритмом циклической (5×5)-свертки. Сравнить эти характеристики с характеристиками гнездового алгоритма, сочетающего алгоритм 15-точечной одномерной циклической свертки с ним самим.

б. Повторить вычисления для двумерной циклической (21×21)-свертки.

7.10. Для вычисления циклической $(n \times n)$-свертки комплексных векторов можно попросту воспользоваться алгоритмом вычисления циклической $(n \times n)$-свертки вещественных векторов, заменив все вещественные умножения и сложения на комплексные умножения и сложения. На сколько можно улучшить алгоритм вычисления циклической (5×5)-свертки комплексных векторов? Насколько лучше циклическая (4×4)-свертка комплексных векторов?

Замечания

На простейшем уровне конструкций двумерное преобразование Фурье состоит из двух независимых преобразований Фурье, применяемых отдельно вдоль каждой из осей; аналогичное утверждение относится к двумерной свертке. Следовательно, быстрые алгоритмы вычисления одномерной свертки в равной мере применимы и для вычисления двумерной свертки. Являющиеся по существу двумерными алгоритмы возникли позже. Пионером в этой области явился Нуссбаумер [1] (1977), введя свои полиномиальные преобразования. Они были сначала определены эвристически по аналогии с преобразованиями Фурье, поскольку эти преобразования и представляли интерес. Только позже стало понятно, что полиномиальные преобразования представляют нечто большее, чем просто аналогию преобразованиям Фурье. Их интерпретация в виде преобразования Фурье в полиномиальном представлении расширений полей была дана Блейхутом [2] (1983). Аналогичные идеи были высказаны Бетом, Фуми и Махвельдом [3] (1982). Другое полиномиальное преобразование, с более ясной структурой, но с большей сложностью вычислений, введено Арамбеполой и Райнером [4] (1979).

Использование индексации типа Гуда — Томаса для вычисления одномерных сверток принадлежит Агарвалу и Кули [5] (1977). Они преобразовали одномерную свертку в многомерную и для вычисления последней применили гнездовой алгоритм, сочетающий одномерные алгоритмы, — процедура, описанная нами в терминах кронекеровского произведения. Их гнездовой метод совпадает с предложенным Виноградом [8] (1978) для вычисления многомерных преобразований Фурье.

Идея использования для уменьшения числа сложений при вычислении многомерной свертки китайской теоремы об остатках для многочленов принадлежит Нуссбаумеру [6] (1978). Использовать в качестве модулей многочлены с коэффициентами в полиномиальных расширениях полей было предложено Питасом и Стринзисом [7] (1982). Многие из приведенных в главе таблиц основаны на работе Нуссбаумера.

Глава 8

БЫСТРЫЕ АЛГОРИТМЫ МНОГОМЕРНЫХ ПРЕОБРАЗОВАНИЙ

В предыдущих главах были рассмотрены способы разбиения задачи вычисления преобразования Фурье на подзадачи и способы преобразования алгоритмов свертки в алгоритмы вычисления преобразования Фурье. Задачу вычисления многомерного преобразования Фурье также можно разбить на подзадачи и использовать алгоритмы вычисления многомерных сверток для вычисления многомерных преобразований Фурье. В данном случае эти возможности даже богаче, чем в одномерном случае. Мы рассмотрим многие из них.

Алгоритмы многомерных преобразований Фурье рассматриваются на простейшем примере двумерных преобразований Фурье. Методы и формулы, полученные для двумерного преобразования Фурье, переносятся на произвольный многомерный случай непосредственно.

Многомерные преобразования Фурье возникают естественно в тех задачах цифровой обработки сигналов, которые по существу многомерны. Но они возникают и искусственным путем как способ вычисления одномерного преобразования Фурье. В данной главе изучаются эти методы и тем самым даются практические способы построения алгоритмов одномерного преобразования Фурье больших длин из рассмотренных в гл. 4 малых алгоритмов преобразования Фурье.

8.1. Алгоритмы Кули—Тьюки по малому основанию

Для вычисления двумерного преобразования Фурье можно использовать БПФ-алгоритм Кули—Тьюки, применяя его сначала к каждой строке, а потом к каждому столбцу. Это обосновывается простой расстановкой скобок в уравнениях, определяющих двумерное преобразование Фурье. Обозначим через $\mathbf{v}$ двумерную таблицу с элементами $v_{i',i''}$, $i' = 0, \ldots, n' - 1$, $i'' = 0, \ldots, n'' - 1$, из поля F. Двумерное преобразование Фурье таблицы $\mathbf{v}$

определяется как двумерная таблица $\mathbf{V}$ с элементами из поля F, вычисляемыми по формулам

$$V_{k',\,k''} = \sum_{i'=0}^{n'-1} \sum_{i''=0}^{n''-1} \omega^{i'k'} \mu^{i''k''} v_{i',\,i''}, \qquad \begin{matrix} k' = 0, \ldots, n' - 1, \\ k'' = 0, \ldots, n'' - 1, \end{matrix}$$

где ω — первообразный корень степени n' из единицы в поле F, а μ — первообразный корень степени n'' из единицы в поле F; при $n' = n''$ обычно полагают $\omega = \mu$. Ясно, что

$$V_{k',\,k''} = \sum_{i'=0}^{n'-1} \omega^{i'k'} \left[\sum_{i''=0}^{n''-1} \mu^{i''k''} v_{i',\,i''} \right] = \sum_{i''=0}^{n''-1} \mu^{i''k''} \left[\sum_{i'=0}^{n'-1} \omega^{i'k'} v_{i',\,i''} \right].$$

Отсюда видно, что двумерное преобразование Фурье можно вычислять, либо вычисляя сначала одномерные преобразования по столбцам и вслед за этим одномерные преобразования по строкам, либо сначала по строкам, а вслед за этим по столбцам. Любые из БПФ-алгоритмов пригодны и для строк и для столбцов, и даже не нужно, чтобы они совпадали. В нашем распоряжении имеется много хороших БПФ-алгоритмов, и любой из них можно выбрать, чтобы уменьшить число необходимых умножений и число необходимых сложений.

При больших объемах таблиц помимо числа сложений и умножений возникает задача управления данными. (1024×1024)-таблица над полем вещественных чисел содержит более одного миллиона чисел, и вдвое больше над полем комплексных чисел. Процессор может запоминать большую часть таблицы в глобальной памяти, выбирая только часть данных в локальную память. Обмен данными между глобальной и локальной памятью является не менее важным моментом, чем число умножений и число сложений.

Мы рассмотрим простейшую модель механизма обмена данными, в которой данные в глобальную память записываются по строкам и считываются в локальную память по строкам. Тогда процесс вычисления двумерного преобразования Фурье сводится к одномерному преобразованию Фурье каждой строки, транспонированию таблицы и повторному применению одномерного преобразования Фурье к каждой новой строке. Если окончательный результат должен быть записан в глобальной памяти в виде истинных строк преобразования, то нужно еще раз выполнить транспонирование. Быстрые алгоритмы транспонирования будут рассмотрены в гл. 10.

Для того, чтобы избежать операции транспонирования, посмотрим более внимательно на используемые в алгоритме Кули—Тьюки правила разбиения и применим их к построению многомерных алгоритмов БПФ. В многомерных алгоритмах будем делать разбиения не по строкам и столбцам, а прямо разбивая двумерную таблицу. Различие этих способов разбиения показано на рис. 8.1. В частности, двумерное разбиение по основанию 2 разбивает $(n \times n)$-таблицу на четыре $((n/2) \times (n/2))$-таблицы, а двумерное разбиение по основанию 4 разбивает $(n \times n)$-таблицу на шестнадцать $((n/4) \times (n/4))$-таблиц. Последнее разбиение, в частности, привлекательно не только тем, что преобразует данные к малым объемам, но также и тем, что позволяет уменьшить число необходимых умножений и сложений.

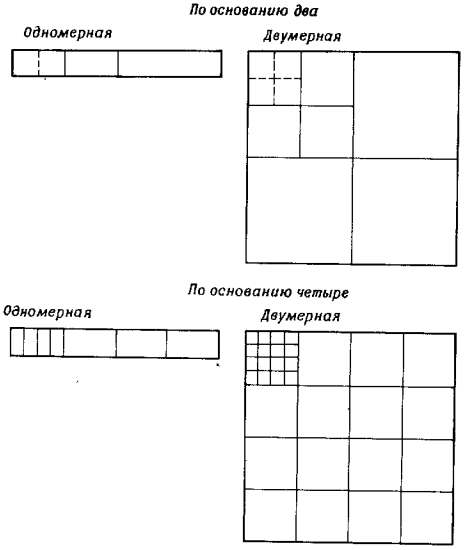


Рис. 8.1. Некоторые схемы прореживания.

Рассмотрим задачу вычисления двумерного $(n \times n)$-точечного преобразования Фурье

$$V_{k,\,l} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \omega^{ik} \omega^{jl} v_{i,\,j},$$

где $n = n'n''$. Отметим, что мы специально убрали штрихованные обозначения индексов, чтобы воспользоваться разбиением Кули—Тьюки. Теперь n' и n'' — размеры прямоугольной таблицы.

Напомним приведенную на рис. 4.1 формулу разбиения Кули—Тьюки:

$$V_{k',\,k''} = \sum_{i'=0}^{n'-1} \beta^{i'k'} \left[\omega^{i'k''} \sum_{i''=0}^{n''-1} \gamma^{i''k''} v_{i',\,i''} \right].$$

Применим эту формулу к двумерному преобразованию дважды — один раз к строчным индексам, а другой раз к столбцовым индексам. Тогда, меняя порядок суммирования, можно записать

$$V_{k',\,k'',\,l',\,l''} =$$

$$= \sum_{i'=0}^{n'-1} \sum_{j'=0}^{n'-1} \beta^{i'k'} \beta^{j'l'} \left[\omega^{l'k''} \omega^{j'l''} \sum_{i''=0}^{n''-1} \sum_{j''=0}^{n''-1} \gamma^{i''k''} \gamma^{j''l''} v_{i',\,i'',\,j',\,j''} \right].$$

Теперь мы получили запись преобразования в виде двумерного $(n''\times n'')$-точечного преобразования для каждых i' и j', за которым следует покомпонентное умножение, а за ним $(n'\times n')$-точечное двумерное преобразование Фурье для каждых k'' и l''.

Для построения двумерного алгоритма БПФ с разбиением по времени по основанию 2 выберем $n' = 2$ и $n'' = n/2$. Тогда уравнения в матричном виде для $k = 0, \ldots, n/2 - 1$ и $l = 0, \ldots, n/2 - 1$ даются равенством

$$\begin{bmatrix} V_{k,\,l} \\ V_{k+n/2,\,l} \\ V_{k,\,l+n/2} \\ V_{k+n/2,\,l+n/2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \sum_{i=0}^{n/2-1} \sum_{j=0}^{n/2-1} \omega^{2ik} \omega^{2jl} v_{2i,\,2j} \\ \omega^k \sum_{i=0}^{n/2-1} \sum_{j=0}^{n/2-1} \omega^{2ik} \omega^{2jl} v_{2i+1,\,2j} \\ \omega^l \sum_{i=0}^{n/2-1} \sum_{j=0}^{n/2-1} \omega^{2ik} \omega^{2jl} v_{2i,\,2j+1} \\ \omega^k \omega^l \sum_{i=0}^{n/2-1} \sum_{j=0}^{n/2-1} \omega^{2ik} \omega^{2jl} v_{2i+1,\,2j+1} \end{bmatrix}$$

Этот БПФ-алгоритм разбивает таблицу входных данных на четыре таблицы в соответствии с четностью или нечетностью обоих индексов. Выходная таблица также разбивается на четыре таблицы, хотя и по другому правилу, определяемому принадлежностью строк и столбцов к первой или второй половине. Вычисления теперь свелись к четырем двумерным $((n/2)\times(n/2))$-точечным преобразованиям Фурье и $(3/4)\,n^2$ умножениям на степени элемента ω. Мы здесь не учитываем $(1/4)\,n^2$ тривиальных умножений, которые

формируются в один блок и легко из алгоритма выбрасываются. Оставшаяся часть тривиальных умножений мала и входит в полное число $(3/4)\,n^2$ умножений. Пусть n равно степени двойки и $M(n\times n)$ обозначает число умножений в поле F, необходимых для вычисления двумерного $(n\times n)$-точечного преобразования Фурье описанным алгоритмом. Тогда мы получаем следующую рекурсию:

$$M(n \times n) = 4M\left(\frac{n}{2} \times \frac{n}{2}\right) + \frac{3}{4} n^2.$$

Решение этого рекуррентного уравнения дается равенством

$$M(n \times n) = \frac{3}{4} n^2 (\log_2 n - C),$$

где константа C определяется числом умножений в самом внутреннем шаге. В частности, можно отталкиваться от (2×2)-преобразования Фурье, не содержащего умножений, так что $M(2\times 2) = 0$, или от (4×4)-преобразования Фурье, также не содержащего умножений, так что $M(4\times 4) = 0$. Тогда

$$M(n \times n) = \frac{3}{4} n^2 (\log_2 n - 1)$$

или

$$M(n \times n) = \frac{3}{4} n^2 (\log_2 n - 2).$$

Возможно и дальнейшее уменьшение числа умножений, но структура алгоритма при этом усложняется.

Полученные формулы интересно сравнить с числом

$$M(n\times n) = n^2 \log_2 n$$

умножений в поле F, необходимым в двумерном алгоритме, основанном на использовании БПФ-алгоритма Кули—Тьюки по основанию 2 по столбцам и по строкам.

Некоторое уменьшение числа необходимых умножений является приятным свойством рассмотренного двумерного алгоритма. Но намного более важной является используемая в нем форма записи входных данных, так как она уменьшает необходимое число обменов данными между глобальной и локальной памятью процессора. Входная таблица считывается в локальную память по две строки одновременно. Объем локальной памяти должен быть достаточен для запоминания двух строк. Вдоль каждой пары строк вычисляются все двумерные (2×2)-преобразования. Этот процесс для данной таблицы повторяется $\log_2 n$ раз. В каждой итерации процесс спаривания двух строк управляется алгоритмом адресного тасования Кули—Тьюки; формирование (2×2)-таблиц в пределах двух спаренных строк также управляется алгоритмом адресного тасования Кули—Тьюки. Так как входная таблица содержит n строк, и так как она трансформируется всего $\log_2 n$ раз, то в общей сложности мы получаем $n \log_2 n$ переносов строк из глобальной памяти в локальную и такое же число обратных переносов.

Чтобы построить двумерный БПФ-алгоритм по основанию 4 положим $n' = 4$ и $n'' = n/4$. Это разобьет исходную двумерную таблицу на 16 подтаблиц. Вычисления можно записать в виде следующего матричного уравнения:

$$
\begin{bmatrix}
V_{k,l} \\
V_{k+n/4,l} \\
V_{k+n/2,l} \\
V_{k+3n/4,l} \\
V_{k,l+n/4} \\
V_{k+n/4,l+n/4} \\
V_{k+n/2,l+n/4} \\
V_{k+3n/4,l+n/4} \\
V_{k,l+n/2} \\
V_{k+n/4,l+n/2} \\
V_{k+n/2,l+n/2} \\
V_{k+3n/4,l+n/2} \\
V_{k,l+3n/4} \\
V_{k+n/4,l+3n/4} \\
V_{k+n/2,l+3n/4} \\
V_{k+3n/4,l+3n/4}
\end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -j & -1 & j \\
1 & -1 & 1 & -1 \\
1 & j & -1 & -j
\end{bmatrix}
\times
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -j & -1 & j \\
1 & -1 & 1 & -1 \\
1 & j & -1 & -j
\end{bmatrix}
\begin{bmatrix}
\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i,2j} \\
\omega^{k}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+1,2j} \\
\omega^{2k}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+2,2j} \\
\omega^{3k}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+3,2j} \\
\omega^{l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i,2j+1} \\
\omega^{k+l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+1,2j+1} \\
\omega^{2k+l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+2,2j+1} \\
\omega^{3k+l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+3,2j+1} \\
\omega^{2l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i,2j+2} \\
\omega^{k+2l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+1,2j+2} \\
\omega^{2k+2l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+2,2j+2} \\
\omega^{3k+2l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+3,2j+2} \\
\omega^{3l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i,2j+3} \\
\omega^{k+3l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+1,2j+3} \\
\omega^{2k+3l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+2,2j+3} \\
\omega^{3k+3l}\sum_{i=0}^{n/4-1}\sum_{j=0}^{n/4-1} \omega^{4ik}\omega^{4jl}v_{2i+3,2j+3}
\end{bmatrix}
$$

Здесь (16×16)-матрица, на которую выполняется умножение, представлена в виде кронекеровского произведения двух (4×4)-матриц с элементами ±1 и $\pm j$. 15/16 из стоящих в правой части равенства членов умножаются на степени элемента ω. Небольшая часть из них представляет собой тривиальные умножения, но, чтобы структура оставалась простой и удобной для расчетов, мы их выбрасывать не будем. Тогда число умножений описывается рекуррентным соотношением

$$M(n\times n) = 16M\left(\frac{n}{4}\times\frac{n}{4}\right) + \frac{15}{16}n^2,$$

решением которого является

$$M(n\times n) = \frac{15}{32}n^2(\log n - C),$$

где константа C имеет такой же смысл, как и для алгоритма по основанию 2. Если выбрать внутреннюю свертку так, что $M(4\times4)=0$, то

$$M(n\times n) = \frac{15}{32}n^2(\log n - 2),$$

и мы видим, что в добавление к хорошей структурированности алгоритм по основанию 4 эффективен в смысле числа необходимых умножений.

Для того, чтобы воспользоваться двумерным разбиением по основанию 4, локальная память должна быть столь большой, чтобы можно было одновременно записать четыре строки матрицы входных данных. В общей сложности таблица входных данных будет переноситься $\log_4 n$ раз, так что всего потребуется $\frac{1}{2}n\log_2 n$ переносов строк из глобальной памяти в локальную и наоборот.

8.2. Гнездовые алгоритмы преобразования

Теперь мы рассмотрим другой метод, известный под названием гнездового, в котором многомерные БПФ-алгоритмы строятся сочетанием одномерных БПФ-алгоритмов, как в БПФ-алгоритме Винограда. Напомним, что следствием возможности изменения порядка суммирования

$$V_{k',k''} = \sum_{i'=0}^{n'-1}\omega^{i'k'}\left[\sum_{i''=0}^{n''-1}\mu^{i''k''}v_{i',i''}\right] =$$
$$= \sum_{i''=0}^{n''-1}\mu^{i''k''}\left[\sum_{i'=0}^{n'-1}\omega^{i'k'}v_{i',i''}\right]$$

является то, что двумерное преобразование Фурье можно вычислять как последовательность одномерных преобразований Фурье

сначала по строкам, а затем по столбцам, или сначала по столбцам, а затем по строкам. Для вычисления одномерных преобразований Фурье можно пользоваться любой удобной комбинацией алгоритмов вычисления, применяя их к строкам и столбцам, входящим в двумерное преобразование. Мы будем работать с малыми БПФ-алгоритмами Винограда, отыскивая эффективные способы их сочетания.

Пусть $M(n')$ и $A(n')$ обозначают соответственно числа умножений и сложений некоторого имеющегося в нашем распоряжении одномерного алгоритма n'-точечного преобразования Фурье. Для выполнения n'' таких одномерных преобразований нужно $n''M(n')$ умножений и $n''A(n')$ сложений. Аналогично, чтобы выполнить n' преобразований длины n'', необходимо $n'M(n'')$ умножений и $n'A(n'')$ сложений. Таким образом, используя последовательно одномерные преобразования, получаем, что вычислительная сложность двумерного преобразования Фурье равна

$$M(n' \times n'') = n''M(n') + n'M(n''),$$
$$A(n' \times n'') = n''A(n') + n'A(n'').$$

Какое из измерений таблицы выбрано первым, не играет роли. Поскольку в такой процедуре обработка осуществляется последовательно, то сложность будет той же самой.

К лучшей процедуре вычислений приводит сочетание алгоритмов по гнездовому методу Винограда. Так как для двумерного преобразования Фурье несущественно, что именно, строки или столбцы, обрабатываются в первую очередь, то, возможно, существует и путь их совместного вычисления. Именно это и делает гнездовой метод Винограда. Он так связывает вычисления по строкам и по столбцам, что полное число необходимых умножений уменьшается. Таким образом, мы будем пользоваться одномерными алгоритмами преобразования Фурье, но сочетая их более эффективным образом. Для описания метода воспользуемся кронекеровским произведением матриц.

Пусть $\mathbf{W}'$ и $\mathbf{W}''$ обозначают соответственно матрицы преобразования Фурье длин n' и n'', так что $\mathbf{V}' = \mathbf{W}'\mathbf{v}'$, $\mathbf{V}'' = \mathbf{W}''\mathbf{v}''$ и

$$V'_k = \sum_{i=0}^{n'-1} \beta^{ik} v'_i, \qquad V''_{k''} = \sum_{i=0}^{n''-1} \gamma^{ik} v''_i.$$

Двумерное $(n' \times n'')$-преобразование Фурье двумерного сигнала $v_{i',\, i''}$ получается применением матрицы $\mathbf{W}'$ к каждому столбцу (столбец содержит n' компонент) и последующим применением матрицы $\mathbf{W}''$ к каждой строке. Это двумерное вычисление можно преобразовать в одномерное так, как показано на рис. 8.2.

Выпишем двумерные таблицы $\mathbf{v}$ и $\mathbf{V}$ в виде одномерных таблиц, считывая их в стеки по столбцам, и сохраним за ними те же обозначения, так что входной и выходной одномерные векторы содержат $n'n''$ компонент и имеют вид

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_{n''-1} \end{bmatrix}, \qquad \mathbf{V} = \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_{n''-1} \end{bmatrix},$$

где $\mathbf{v}_{i''}$ и $\mathbf{V}_{i''}$ обозначают соответственно столбцы исходных двумерных таблиц:

$$\mathbf{v}_{i''} = \begin{bmatrix} v_{0i''} \\ v_{1i''} \\ v_{2i''} \\ \vdots \\ v_{n'-1,i''} \end{bmatrix}, \qquad \mathbf{V}_{i''} = \begin{bmatrix} V_{0i''} \\ V_{1i''} \\ V_{2i''} \\ \vdots \\ V_{n'-1,i''} \end{bmatrix}.$$

Если под $\mathbf{v}$ и $\mathbf{V}$ понимать так построенные одномерные $n'n''$-точечные векторы, то двумерное преобразование Фурье записывается

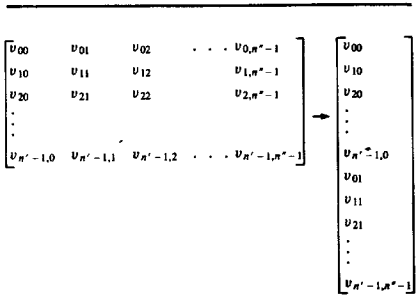


Рис. 8.2. Отображение двумерной таблицы в одномерную.

в виде кронекеровского произведения. Сначала запишем вычисления в виде

$$\begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_{n''-1} \end{bmatrix} = \begin{bmatrix} w''_{00}\mathbf{I} & w''_{01}\mathbf{I} & w''_{02}\mathbf{I} & \dots & w''_{0,n''-1}\mathbf{I} \\ w''_{10}\mathbf{I} & w''_{11}\mathbf{I} & w''_{12}\mathbf{I} & \dots & w''_{1,n''-1}\mathbf{I} \\ w''_{20}\mathbf{I} & w''_{21}\mathbf{I} & w''_{22}\mathbf{I} & \dots & w''_{2,n''-1}\mathbf{I} \\ \vdots & & & & \\ w''_{n''-1,0}\mathbf{I} & & & & w''_{n''-1,n''-1}\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{W}' & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{W}' & \mathbf{0} & & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{W}' & & \mathbf{0} \\ \vdots & & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & & \mathbf{W}' \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_{n''-1} \end{bmatrix},$$

где $\mathbf{W}'$ — определенная выше $(n' \times n')$-матрица, $\mathbf{0}$ — нулевая $(n' \times n')$-матрица и $\mathbf{I}$ — единичная $(n' \times n')$-матрица. В этом произведении матриц легко узнать кронекеровское произведение матриц, так что в сокращенной записи имеем

$$\mathbf{V} = \mathbf{W}\mathbf{v},$$

где $\mathbf{W} = \mathbf{W}'' \times \mathbf{W}'$ представляет собой $(n'n'' \times n'n'')$-матрицу.

В БПФ-алгоритмах Винограда длин n' и n'' соответственно матрицы $\mathbf{W}'$ и $\mathbf{W}''$ разлагаются в виде $\mathbf{W}' = \mathbf{C}'\mathbf{B}'\mathbf{A}'$ и $\mathbf{W}'' = \mathbf{C}''\mathbf{B}''\mathbf{A}''$, где $\mathbf{A}'$, $\mathbf{A}''$, $\mathbf{C}'$ и $\mathbf{C}''$ представляют собой матрицы, состоящие только из нулей и единиц, а матрицы $\mathbf{B}'$ и $\mathbf{B}''$ являются диагональными. Все умножения в алгоритмах Винограда исчерпываются умножениями на матрицы $\mathbf{B}'$ и $\mathbf{B}''$ соответственно. Пусть $\mathbf{W} = \mathbf{W}'' \times \mathbf{W}'$; применяя дважды теорему 2.5.5., получаем

$$\mathbf{W} = (\mathbf{C}''\mathbf{B}''\mathbf{A}'') \times (\mathbf{C}'\mathbf{B}'\mathbf{A}') = (\mathbf{C}'' \times \mathbf{C}')\,(\mathbf{B}'' \times \mathbf{B}')\,(\mathbf{A}'' \times \mathbf{A}') = \\ = \mathbf{C}\mathbf{B}\mathbf{A},$$

где кронекеровское произведение $\mathbf{C} = \mathbf{C}'' \times \mathbf{C}'$ и $\mathbf{A} = \mathbf{A}'' \times \mathbf{A}'$ приводит опять к матрицам, состоящим только из нулей и единиц, а кронекеровское произведение $\mathbf{B} = \mathbf{B}'' \times \mathbf{B}'$ представляет собой диагональную матрицу. Таким образом, мы построили алгоритм двумерного $n'n''$-точечного преобразования Фурье в той же форме, что и БПФ-алгоритм Винограда. Это и дает способ построения двумерных БПФ-алгоритмов из одномерных алгоритмов.

Хороший способ организации вычисления двумерного преобразования Фурье диктуется формулой

$$\mathbf{V} = (\mathbf{C}'' \times \mathbf{C}')\,(\mathbf{B}'' \times \mathbf{B}')\,(\mathbf{A}'' \times \mathbf{A}')\,\mathbf{v},$$

и иллюстрируется рис. 8.3. Здесь предполагается, что данные записаны в виде двумерной таблицы. Для умножения на $(\mathbf{A}'' \times \mathbf{A}')$ сначала каждый столбец входной двумерной таблицы умножается на матрицу $\mathbf{A}'$, а затем каждая строка полученной матрицы умножается на $\mathbf{A}''$. Первая из этих операций не содержит умножений и преобразует входную $(n' \times n'')$-таблицу в таблицу размера $(M(n') \times M(n''))$; вторая операция также содержит только сложения и преобразует данные в таблицу размера $(M(n') \times M(n''))$. Далее происходит умножение каждого столбца на матрицу $\mathbf{B}'$ и затем умножение каждой строки на матрицу $\mathbf{B}''$. Эта процедура содержит $2M(n')\,M(n'')$ умножений. Другой способ реализации этого шага вычислений состоит в предварительном вычислении и запоминании $(M(n') \times M(n''))$-матрицы $\mathbf{B} = \mathbf{B}'' \times \mathbf{B}'$. Тогда на этом шаге потребуется только $M(n')\,M(n'')$ умножений, но память для констант вычисления растет. Наконец, вычисленная на втором шаге $(M(n') \times M(n''))$-матрица преобразуется в $(n' \times n'')$-матрицу умножением сначала каждого столбца на $\mathbf{C}'$, а затем каждой строки на $\mathbf{C}''$. Последний шаг содержит только сложения.

Полное число умножений в алгоритме равно

$$M(n' \times n'') = M(n')\,M(n'').$$

Формула для полного числа сложений выводится несколько сложнее, так как зависит от распределения числа сложений в составляющих алгоритмах между предсложениями и постсложениями. Чтобы упростить эту формулу и ее вывод (и даже уменьшить число необходимых сложений), предположим, что можно менять порядок

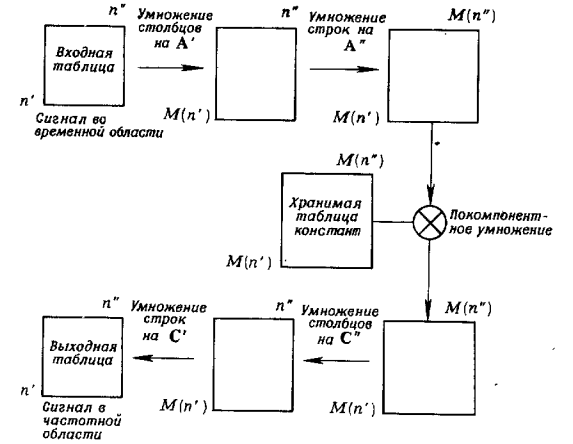


Рис. 8.3. Гнездовой метод вычисления двумерного преобразования Фурье.

применения матриц $\mathbf{C}'$ и $\mathbf{C}''$. (Изменение порядка суммирования оправдано общим тождеством $\sum_{i'} C_{i'k'} \sum_{i''} C_{i''k''} V_{i'i''} = \sum_{i''} C_{i''k''} \times \times \sum_{i'} C_{i'k'} V_{i'i''}$.) Тогда

$$A(n' \times n'') = n'A(n'') + M(n'')\,A(n').$$

Теперь мы уже имеем два способа вычисления двумерного преобразования Фурье. Первый из них содержит

$$M(n' \times n'') = n''M(n') + n'M(n'')$$

умножений, но строится на основе любого одномерного БПФ-алгоритма, а второй содержит

$$M(n' \times n'') = M(n')\,M(n'')$$

умножений, но в конструкции используются только БПФ-алгоритмы Винограда вдоль каждого из измерений

Например, реализация (1008×1008)-преобразования Фурье для комплексного входа в первом случае требует $4 \times 1008 \times 1782$ вещественных умножений (см. разд. 8.3), а во втором случае $2 \times \times 1782^2$ вещественных умножений. (В случае вещественного входа во втором случае требуется вдвое меньше вещественных умножений, а в первом случае только 3/4 от указанного числа, так как после применения БПФ-алгоритма по строкам данные становятся комплексными.)

Второй способ, очевидно, лучше, но для его реализации требуется создание временного массива памяти для запоминания комплексных данных объема $2 \times 1782 \times 1782$ (который в начале может содержать $2 \times 1008 \times 1008$ входных данных); для первого способа необходима только временная одномерная память для запоминания 3564 слов.

8.3. Алгоритмы Винограда быстрого вычисления преобразования Фурье большой длины

Гнездовые методы построения двумерных БПФ-алгоритмов можно использовать, наоборот, для построения алгоритмов вычисления одномерного преобразования Фурье. В настоящем разделе мы возвращаемся к задаче вычисления одномерного преобразования Фурье и строим БПФ-алгоритм Винограда для больших длин преобразования (большой БПФ-алгоритм Винограда).

Большой БПФ-алгоритм Винограда представляет собой метод эффективного вычисления дискретного преобразования Фурье, если длина n преобразования распадается в произведение взаимно

Характеристики БПФ-алгоритма Винограда (комплексные входные данные)			БПФ-алгоритм Кули–Тьюки (комплексные входные данные)		
Длина n	Число вещественных умножений	Число вещественных сложений	Длина n	Число вещественных умножений	Число вещественных сложений
30	72	384	32	320	480
48	92	636			
60	144	888	64	768	1152
91	318	2648			
120	288	2076	128	1792	2688
168	432	3492			
240	648	5016	256	4096	6144
420	1296	11352			
504	1584	14642	512	9216	13824
840	2592	24804			
1008	3564	34920	1024	20480	30720
2520	9504	100188	2048	45056	67584
10920	38760	320196	8192	212992	319488

Рис. 8.4. Сравнение некоторых БПФ-алгоритмов.

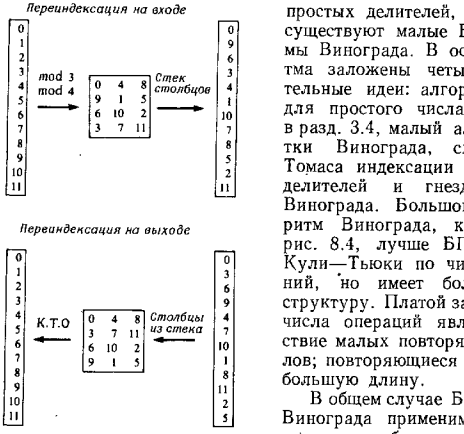


Рис. 8.5. Переиндексация в 12-точечном алгоритме Винограда.

простых делителей, для которых существуют малые БПФ-алгоритмы Винограда. В основу алгоритма заложены четыре самостоятельные идеи: алгоритм Рейдера для простого числа, описанный в разд. 3.4, малый алгоритм свертки Винограда, схема Гуда—Томаса индексации для простых делителей и гнездовой метод Винограда. Большой БПФ-алгоритм Винограда, как видно из рис. 8.4, лучше БПФ-алгоритма Кули—Тьюки по числу умножений, но имеет более сложную структуру. Платой за уменьшение числа операций является отсутствие малых повторяющихся циклов; повторяющиеся циклы имеют большую длину.

В общем случае БПФ-алгоритм Винограда применим для вычисления преобразования, длина n которого равна произведению малых простых чисел или произведению степеней малых простых чисел. Мы рассмотрим только случай двух делителей: $n = n'n''$. Используя алгоритм Гуда—Томаса для простых делителей, преобразуем n-точечное преобразование Фурье в двумерное ($n' \times n''$)-точечное преобразование Фурье. Для вычисления этого двумерного преобразования можно воспользоваться малым n'-точечным БПФ-алгоритмом Винограда и малым n''-точечным БПФ-алгоритмом Винограда, применяя их вдоль соответствующих длине измерений. Вместо этого мы воспользуемся описанным в предыдущем разделе гнездовым методом Винограда, позволяющим так сочетать эти два алгоритма, что число умножений падает.

Рассматриваемая нами процедура иллюстрируется приведенным на рис. 8.5 примером 12-точечного БПФ-алгоритма. На рисунке показано преобразование входного 12-компонентного вектора данных в двумерную таблицу по алгоритму Гуда—Томаса, сводящее задачу к вычислению двумерного преобразования Фурье. За этим следует преобразование полученной двумерной таблицы в новый одномерный массив, формируемый как стек столбцов. (Все эти манипуляции можно проделывать с индексами; совсем не обязательно физически переупорядочивать данные.)

Формирование 12-точечной матрицы преобразования Фурье в виде кронекеровского произведения матриц-составляющих 3-точечного и 4-точечного преобразований Фурье показано на рис. 8.6.

Пусть $n = n'n''$ и $\mathbf{W}'$ и $\mathbf{W}''$ обозначают соответственно матрицы n'-точечного и n''-точечного преобразований Фурье, так что $\mathbf{V}' = \mathbf{W}'\mathbf{v}'$ и $\mathbf{V}'' = \mathbf{W}''\mathbf{v}''$. Двумерное ($n' \times n''$)-преобразование Фурье двумерного сигнала $v_{i'i''}$ получается применением $\mathbf{W}'$ к каждому столбцу сигнала, а затем применением $\mathbf{W}''$ к каждой строке. Если двумерный ($n' \times n''$) сигнал $v_{i'i''}$ был получен описанным переупорядочиванием компонент из одномерного, то надо выполнить обратную перестановку, считывая двумерный результат по столбцам.

Если под $\mathbf{v}$ и $\mathbf{V}$ понимать соответственно входной и выходной векторы, компоненты которых переставлены описанным образом, то в терминах кронекеровского произведения преобразование записывается равенством $\mathbf{V} = (\mathbf{W}'' \times \mathbf{W}')\,\mathbf{v}$. Но $\mathbf{W}' = \mathbf{C}'\mathbf{B}'\mathbf{A}'$ и $\mathbf{W}'' = \mathbf{C}''\mathbf{B}''\mathbf{A}''$, где элементами матриц $\mathbf{A}'$, $\mathbf{A}''$, $\mathbf{C}'$ и $\mathbf{C}''$ являются только нули и единицы, а матрицы $\mathbf{B}'$ и $\mathbf{B}''$ являются диагональными. Так же, как в предыдущем разделе, получаем

$$\mathbf{W} = (\mathbf{C}''\mathbf{B}''\mathbf{A}'') \times (\mathbf{C}'\mathbf{B}'\mathbf{A}') = (\mathbf{C}'' \times \mathbf{C}')\,(\mathbf{B}'' \times \mathbf{B}')\,(\mathbf{A}'' \times \mathbf{A}') = \mathbf{CBA},$$

где кронекеровские произведения $\mathbf{C} = \mathbf{C}'' \times \mathbf{C}'$ и $\mathbf{A} = \mathbf{A}'' \times \mathbf{A}'$ представляют собой матрицы, состоящие только из нулей и единиц, а кронекеровское произведение $\mathbf{B} = \mathbf{B}'' \times \mathbf{B}'$ является диагональной матрицей. Следовательно, мы построили алгоритм $n'n''$-точечного преобразования Фурье в форме БПФ-алгоритма Винограда: $\mathbf{V} = \mathbf{CBA}\mathbf{v}$.

При построении алгоритма мы предполагали, что компоненты векторов $\mathbf{v}$ и $\mathbf{V}$ переупорядочены необходимым образом. Но коль скоро алгоритм в данной форме уже построен, то выполняя тривиальную перестановку столбцов матрицы $\mathbf{A}$, можно работать с естественным порядком компонент вектора $\mathbf{v}$, а переставляя строки матрицы $\mathbf{C}$, — с естественным порядком компонент вектора $\mathbf{V}$.

Обозначим через $M(n')$ и $M(n'')$ размеры матриц $\mathbf{B}'$ и $\mathbf{B}''$ соответственно. Эти числа равны числу умножений соответственно в n'-точечном БПФ-алгоритме и в n''-точечном БПФ-алгоритме с учетом тривиальных умножений на единицу. Тогда число умножений, необходимых для выполнения $n'n''$-точечного БПФ-алгоритма Винограда, включая и умножения на единицу, равно $M(n) = M(n')\,M(n'')$. Это происходит потому, что матрица $\mathbf{B}'' \times \mathbf{B}'$ снова является диагональной, а размер ее равен $M(n')\,M(n'')$.

На рис. 8.7 приведен перечень малых БПФ-алгоритмов Винограда, из которых можно строить большие БПФ-алгоритмы Винограда, а на рис. 8.8 выписаны характеристики больших БПФ-алгоритмов Винограда. На рис. 8.9 дается пример 1008-точечного

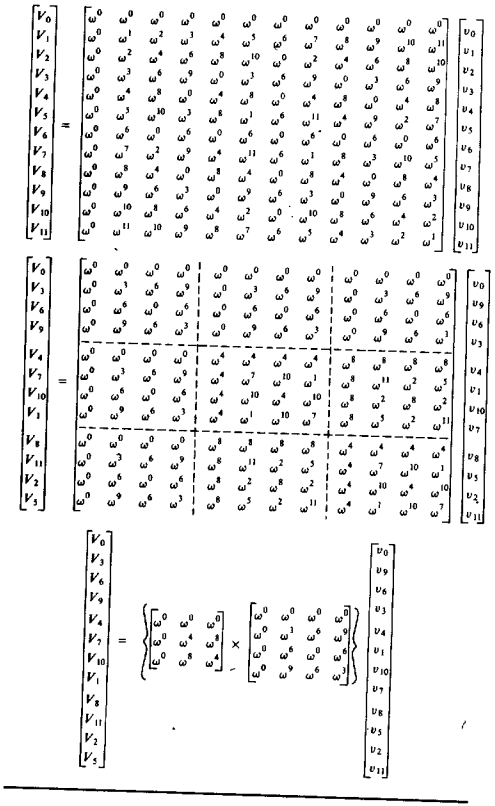
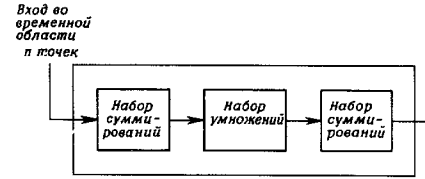


Рис. 8.6. Сведение 12-точечного преобразования Фурье к кронекеровскому произведению.



Малый БПФ-алгоритм Винограда

Меню

A	B	C	D	E	F
n = 2					
	n = 3				
n = 4					
		n = 5			
			n = 7		
n = 8					
	n = 9				
				n = 11	
					n = 13
n = 16					

Сложность	
Умножения	*Сложения*
0 (2)	2
2 (3)	6
0 (4)	8
5 (6)	17
8 (9)	36
2 (8)	26
10 (11)	44
20 (21)	84
20 (21)	94
10 (18)	74

Рис. 8.7. Набор малых преобразований Винограда.

Длина n	*Число вещественных умножений*		*Число вещественных сложений**	*Число нетривиальных умножений на точку**	*Число сложений на точку*
	Полное	*Нетривиальных*			
15	18	17	81	1.13	5.4
21	27	26	150	1.24	7.14
30	36	34	192	1.13	6.40
35	54	53	333	1.51	9.51
48	54	46	318	0.96	6.62
63	99	98	704	1.56	11.17
80	94	86	676	1.07	8.45
120	144	138	1038	1.15	8.65
168	216	210	1746	1.25	10.39
240	324	316	2508	1.32	10.45
420	648	644	5676	1.53	13.51
504	792	786	7270	1.56	14.42
840	1296	1290	12402	1.54	14.76
1008	1782	1774	17334	1.76	17.20
2520	4752	4746	49814	1.88	19.77

* *При комплексном входе удвоить*

Рис. 8.8. Характеристики больших БПФ-алгоритмов Винограда.

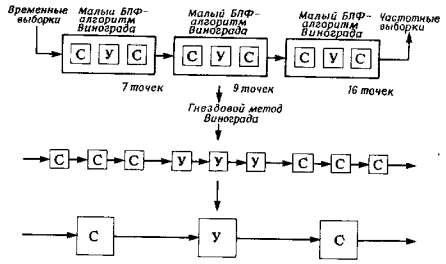


Рис. 8.9. Структура большого 1008-точечного БПФ-алгоритма Винограда.

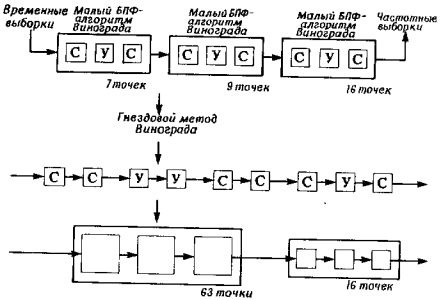


Рис. 8.10. Структура другого 1008-точечного преобразования.

преобразования. Этот БПФ-алгоритм содержит 3564 вещественных умножений, которое получено как произведение чисел необходимых умножений в 7-, 8- и 16-точечных БПФ-алгоритмах Винограда, удвоенное для комплексного входа.

Более регулярный БПФ-алгоритм с разбиением данных на меньшие блоки приведен на рис. 8.10. В этом алгоритме сочетаются только 7-точечное и 9-точечное преобразования. 1008-точечное преобразование вычисляется затем как двумерное (63×16)-преобразование Фурье. Каждая составляющая вычисляется БПФ-алгоритмом Винограда. Полное число вещественных умножений равно $4396 = 2\,(16M\,(63) + 63M\,(16))$.

8.4. Алгоритм Джонсона—Барраса быстрого преобразования Фурье

Мы рассмотрели два способа сочетания малых БПФ-алгоритмов Винограда, а именно, гнездовую схему Гуда—Томаса для взаимно простых делителей и гнездовой метод Винограда. БПФ-алгоритмы Джонсона—Барраса представляют собой целое семейство гнездовых алгоритмов, включающее в себя методы Гуда—Томаса и Винограда в качестве двух экстремальных случаев. Идея БПФ-алгоритма Джонсона—Барраса состоит в модернизации использования кронекеровского произведения для переупорядочивания последовательности вычислений. Это позволяет уменьшить число необходимых умножений, сохраняя малым число необходимых сложений. Кроме того, архитектура алгоритма позволяет улучшить структурирование и управление потоком данных.

Мы рассмотрим только случай, когда длина n преобразования распадается в произведение двух взаимно простых множителей n' и n''. Можно, конечно, применить тот же метод и к большему числу делителей, но тогда число конструктивных возможностей становится необозримым. Например, если n разлагается в произведение четырех множителей, то из одних и тех же малых БПФ-алгоритмов Винограда можно построить больше чем 10^{12} различных БПФ-алгоритмов Джонсона—Барраса. Чтобы все их проанализировать, необходима специальная процедура поиска типа динамического программирования.

Предположим, что n-точечный входной вектор преобразуется в двумерную $(n' \times n'')$-таблицу $\mathbf{V}$ с помощью отображения на расширенную диагональ. Согласно алгоритму Гуда—Томаса для вычисления одномерного преобразования Фурье исходного вектора данных можно сначала вычислить n'-точечное преобразование Фурье каждого столбца таблицы, а затем n''-точечное преобразование Фурье каждой строки. Это можно записать в виде $\mathbf{V} = \mathbf{W}''\mathbf{W}'\mathbf{v}$. Такая запись не является общепринятой, так как $\mathbf{v}$ представляет собой двумерную таблицу. Под $\mathbf{W}'\mathbf{v}$ понимается умножение каждого столбца таблицы $\mathbf{v}$ по одному разу на матрицу $\mathbf{W}'$, а под $\mathbf{W}''\mathbf{v}$ понимается умножение каждой строки таблицы $\mathbf{v}$ по одному разу на матрицу $\mathbf{W}''$. Для большей строгости надо было бы ввести дополнительные обозначения для отделения матриц, действующих на столбцы, от матриц, действующих на строки; но мы предпочитаем для указания этой информации пользоваться штрихом и двумя штрихами.

Так как безразлично, вычисляются ли первыми преобразования по строкам или преобразования по столбцам, то можно также записать $\mathbf{V} = \mathbf{W}'\mathbf{W}''\mathbf{v}$. Это объясняется тем, что операции по строкам коммутируют с операциями по столбцам, так как такая запись означает просто изменение порядка суммирования. Пред-

положим теперь, что имеются малые БПФ-алгоритмы Винограда с $\mathbf{W}' = \mathbf{C}'\mathbf{B}'\mathbf{A}'$ и $\mathbf{W}'' = \mathbf{C}''\mathbf{B}''\mathbf{A}''$. Тогда

$$\mathbf{V} = (\mathbf{C}''\mathbf{B}''\mathbf{A}'')(\mathbf{C}'\mathbf{B}'\mathbf{A}')\,\mathbf{v} = (\mathbf{C}'\mathbf{B}'\mathbf{A}')(\mathbf{C}''\mathbf{B}''\mathbf{A}'')\,\mathbf{v}.$$

Две матрицы с одним и тем же количеством штрихов не коммутируют. Однако, как мы видели при исследовании гнездового метода Винограда, матрицы с различным количеством штрихов коммутируют. Для формального доказательства можно обратиться к теореме о кронекеровском произведении матриц. На самом деле, это опять не более чем изменение порядка суммирования. Таким образом, можно записать равенство

$$\mathbf{V} = \mathbf{C}'\mathbf{C}''\mathbf{B}'\mathbf{B}''\mathbf{A}'\mathbf{A}''\mathbf{v},$$

которое представляет собой большой БПФ-алгоритм Винограда. Его можно записать также в виде

$$\mathbf{V} = \mathbf{C}'\mathbf{B}'\mathbf{C}''\mathbf{B}''\mathbf{A}'\mathbf{A}''\mathbf{v}.$$

Для построения БПФ-алгоритма Джонсона—Барраса необходима еще одна хитрость. Прежде чем менять порядок следования матриц суммирования, разложим их в виде произведений:

$$\mathbf{W}' = (\mathbf{G}'\mathbf{H}')\,\mathbf{B}'\,(\mathbf{E}'\mathbf{F}'), \qquad \mathbf{W}'' = (\mathbf{G}''\mathbf{H}'')\,\mathbf{B}''\,(\mathbf{E}''\mathbf{F}''),$$

где $\mathbf{C}' = \mathbf{G}'\mathbf{H}'$ и так далее. Тогда мы получаем произведение

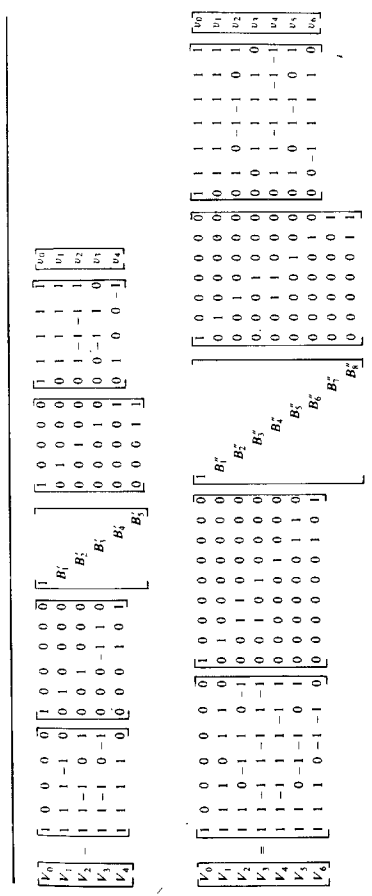
$$\mathbf{V} = \mathbf{G}'\mathbf{H}'\mathbf{B}'\mathbf{E}'\mathbf{F}'\mathbf{G}'\mathbf{H}''\mathbf{B}''\mathbf{E}''\mathbf{F}''\mathbf{v},$$

которое можно переупорядочить следующим образом:

$$\mathbf{V} = (\mathbf{G}'\mathbf{G}''\mathbf{H}''\mathbf{H}')\,(\mathbf{B}'\mathbf{B}'')\,(\mathbf{E}'\mathbf{E}''\mathbf{F}'\mathbf{F}'')\,\mathbf{v}.$$

В этой форме записи обе диагональные матрицы $\mathbf{B}'$ и $\mathbf{B}''$ собраны вместе и даже заключены в скобки, чтобы подчеркнуть этот факт.

Эти идеи хорошо иллюстрируются 35-точечным преобразованием Фурье. На рис. 8.11 выписаны 5-точечный и 7-точечный БПФ-алгоритмы Винограда, каждый из которых состоит из пяти этапов вычислений. Имеется множество способов сочетания их в 35-точечный алгоритм БПФ, наиболее интересные три из которых указаны на рисунке. Два из них соответствуют гнездовому методу Гуда—Томаса и гнездовому методу Винограда. Гнездовой метод Гуда—Томаса приводит к меньшему числу сложений, а гнездовой метод Винограда — к меньшему числу умножений. Надобности выбирать между этими альтернативами, однако, не возникает, так как имеется гнездовой алгоритм Джонсона—Барраса, который содержит столько же умножений, сколько алгоритм Винограда, и число сложений, близкое к числу сложений в алгоритме Гуда—Томаса, и, следовательно, ему надо отдать предпочтение.



Гнездовые методы	Упорядочение	Вещественных умножений	Вещественных сложений
Гуда—Томаса	$(G'H'B'E'F')(G''H''B''E''F'')$	75	299
Винограда	$(G'H'G''H''B'B'')(E'E''F'F'')$	53	333
Джонсона—Барраса	$(G'G''H''H')(B'B'')(E'E''F'F'')$	53	305

Рис. 8.11. Пример БПФ-алгоритма Джонсона—Барраса.

Использование БПФ-алгоритма Джонсона—Барраса не приводит к каким-либо осложнениям. Все новое, что он привносит, сводится к разбиению сложений на отдельные пакеты, которые можно реализовать в виде отдельных подпрограмм и затем обращаться к ним, вызывая их в нужной последовательности. В процессе вычислений мы постоянно переходим от сложений по строкам к сложениям по столбцам и обратно.

8.5. Алгоритмы разложения

Малый БПФ-алгоритм Винограда сводит задачу вычисления одномерного преобразования Фурье к задаче вычисления одномерной циклической свертки, используя для этого алгоритм Рейдера. Теперь мы воспользуемся этой же процедурой для многомерных преобразований. В общем случае вычисление преобразования разлагается в вычисление нескольких циклических сверток. Число точек преобразования по каждой из осей многомерного преобразования должно быть равно простому числу или степени простого числа, хотя и не обязательно одной и той же для различных измерений. Сначала, используя вдоль каждой из осей алгоритм Рейдера или его обобщение, сведем многомерное преобразование Фурье к многомерной свертке. Теперь применим алгоритм быстрого вычисления многомерной свертки. Для того случая, когда число точек преобразования вдоль всех осей одно и то же, в разд. 8.7 будет описан алгоритм с лучшими характеристиками, которому и следует отдать предпочтение. Поэтому алгоритмы разложения представляют интерес в первую очередь для тех случаев, когда соответствующие различным осям числа точек преобразования различны.

Идея сводится к простому обобщению процедуры для одномерного случая. Начнем с двумерного $(n' \times n'')$-преобразования Фурье для взаимно простых n' и n'', возможно, различных:

$$V_{k',\,k''} = \sum_{i'=0}^{n'-1} \sum_{i''=0}^{n''-1} \omega^{i'k'} \mu^{i''k''} v_{i',\,i''}, \quad \begin{matrix} k' = 0, \ldots, n' - 1, \\ k'' = 0, \ldots, n'' - 1. \end{matrix}$$

Для того чтобы воспользоваться алгоритмом Рейдера, надо исключить нуль из показателя степени. Для этого по аналогии с алгоритмом Рейдера разобьем уравнения на четыре группы:

$$V_{0,0} = \sum_{i'=0}^{n'-1} \sum_{i''=0}^{n''-1} v_{i',\,i''},$$

$$V_{0,k''} - V_{0,0} = \sum_{i''=1}^{n''-1} (\mu^{i''k''} - 1) \sum_{i'=0}^{n'-1} v_{i',\,i''}, \quad k'' = 1, \ldots, n'' - 1,$$

$$V_{k',\,0} - V_{0,0} = \sum_{i'=1}^{n'-1} (\omega^{i'k'} - 1) \sum_{i''=0}^{n''-1} v_{i',\,i''}, \quad k' = 1, \ldots, n' - 1,$$

$$V_{k',\,k''} - V_{k',0} - V_{0,\,k''} + V_{0,0} = \sum_{i'=1}^{n'-1} \sum_{i''=1}^{n''-1} (\omega^{i'k'} - 1)(\mu^{i''k''} - 1) v_{i',\,i''},$$
$$k' = 1, \ldots, n' - 1, \quad k'' = 1, \ldots, n'' - 1.$$

Сделав такое разбиение и применив перестановку, даваемую алгоритмом Рейдера, мы сводим задачу к вычислению $(n' - 1)$-точечной свертки, $(n'' - 1)$-точечной свертки и $(n' - 1) \times (n'' - 1)$-точечной двумерной свертки. Для вычисления обеих одномерных сверток можно воспользоваться построенными в гл. 3 алгоритмами Винограда. Согласно теоремам 4.6.1 и 4.6.2, все умножения в этих алгоритмах являются умножениями только на вещественные или только на мнимые константы, и, следовательно, требуют в случае вещественного входа только по одному вещественному умножению.

Для вычисления двумерной свертки можно воспользоваться любым хорошим методом. Одним из них является описанное в гл. 7 полиномиальное преобразование. Алгоритм двумерной $((n' - 1) \times (n'' - 1))$-точечной свертки можно также строить гнездовым методом, используя подходящие двумерные циклические свертки меньших объемов или даже малые алгоритмы одномерных циклических сверток. Например, задачу вычисления двумерной (4×12)-свертки можно преобразовать, используя по второй оси алгоритм Агарвала—Кули, в задачу вычисления трехмерной циклической $(4 \times 4 \times 3)$-свертки. Для решения последней задачи можно воспользоваться гнездовым методом сочетания алгоритма вычисления двумерной циклической (4×4)-свертки с алгоритмом 3-точечной циклической свертки.

Если для вычисления двумерного преобразования Фурье используется алгоритм двумерной циклической свертки, то во всех умножениях один из множителей всегда оказывается либо чисто вещественным числом, либо чисто мнимым числом. Частный случай этого утверждения дается следующей теоремой, представляющей собой аналог теоремы 4.6.1.

Теорема 8.5.1. *Пусть p — нечетное простое число и $g(x, y)$ — многочлен Рейдера от двух переменных степени $p - 1$ по каждой из них. Пусть $Q(x)$ и $Q'(x)$ — круговые многочлены, делящие $x^p - 1$. Тогда по модулю $Q(x)$ и по модулю $Q'(y)$ коэффициенты многочлена $g(x, y)$ представляют собой либо чисто вещественные числа, либо чисто мнимые числа.*

Доказательство аналогично доказательству теоремы 4.6.1. □

Можно доказать более общую теорему для случая, когда числа точек преобразования по обеим осям равны (не обязательно одинаковым) простым числам или степеням простых чисел.

Размер массива $n \times n$	Число вещественных умножений		Число вещественных сложений	Нетривиальных умножений на точку на выходе	Сложений на точку на выходе
	общее	нетривиальных			
5×5	33	32	230	1.28	9.20
7×7	69	68	650	1.39	13.26
9×9	109	108	908	1.33	11.21
7×9	87	86	712	1.36	11.30
5×13	114	113	917	1.74	14.10

Для комплексных входных данных все цифры удваиваются

Рис. 8.12. Характеристики некоторых алгоритмов разложения Нуссбаумера—Квенделла.

Рис. 8.13. Разбиение (7×7)-преобразования Фурье.

Характеристики некоторых построенных таким способом двумерных алгоритмов выписаны на рис. 8.12. Если сравнить эти характеристики с приведенными на рис. 8.18 характеристиками, то можно увидеть, что для $(p \times p)$-точечных БПФ-алгоритмов данный способ проигрывает приведенному на рис. 8.18.

В качестве примера рассмотрим двумерное (7×7)-точечное преобразование Фурье. Как показано на рис. 8.13, оно распадается в вычисление двух 6-точечных циклических сверток и одной двумерной (6×6)-точечной циклической свертки. Для вычисления каждой 6-точечной циклической свертки необходимо восемь вещественных умножений. Алгоритм вычисления двумерной (6×6)-свертки можно строить гнездовым методом, сочетая алгоритм циклической (2×2)-свертки с алгоритмом циклической (3×3)-свертки, данным на рис. 7.9. Для такого вычисления необходимо 52 умножения, так что в общей сложности получаем 68 умножений. Еще одно, тривиальное, умножение (на 1) нужно для того, чтобы представить в том же виде вычисление компоненты $V_{0,0}$; это существенно при использовании данного (7×7)-БПФ-алгоритма в качестве составляющей в гнездовом методе построения больших БПФ-алгоритмов.

Подсчет числа сложений несколько более громоздок, и результат зависит от того, насколько удачно организованы уравнения. Сначала рассмотрим уравнения в том виде, как они были выписаны выше. Тогда вычисления состоят из некоторых предсложений, предшествующих обращению к двум 6-точечным циклическим сверткам и одной (6×6)-точечной двумерной циклической свертке, и

следующих за этим постсложений. Некоторые предсложения и постсложения входят также и в алгоритмы сверток. Полное число сложений тогда вычисляется так:

Предсложения	84
Две 6-точечные циклические свертки	68
Циклическая (6×6)-свертка	424
Постсложения	84
	660

Однако мы знаем, что если собрать вместе все предсложения, включая и те, которые входят в алгоритмы сверток, то возможно уменьшение числа сложений. Аналогичного уменьшения числа сложений можно добиться, собирая вместе все постсложения. Мы не можем сделать это в достаточно общем виде, поскольку нет никакой общей теории минимизации числа сложений и, что еще важнее, поскольку это не позволяет построить алгоритм, хорошо распадающийся на мелкие подпрограммы так, как рассматриваемый алгоритм. Тем не менее кое-что можно сделать. 6-точечные циклические свертки можно скомбинировать с некоторыми предсложениями и постсложениями так, чтобы сформировать 7-точечные преобразования Фурье. Это не меняет числа необходимых умножений. Вычисления в так организованном алгоритме будут состоять из некоторых предсложений, одного 7-точечного преобразования Фурье, одной 6-точечной циклической свертки, одной двумерной (6×6)-свертки и некоторых постсложений. Полное число сложений уменьшается следующим образом:

Предсложения	78
7-точечное БПФ	36
6-точечная циклическая свертка	34
Циклическая (6×6)-свертка	424
Постсложения	78
	650

Это, конечно, небольшое улучшение.

В качестве второго примера рассмотрим двумерное (5×13)-преобразование Фурье. Как показано на рис. 8.14, разобъем его на 4-точечную циклическую свертку, 12-точечную циклическую свертку и двумерную циклическую (4×12)-свертку. Для вычисления двумерной циклической (4×12)-свертки преобразуем ее в трехмерную циклическую (4×4×3)-свертку и воспользуемся гнездовым алгоритмом, сочетающим алгоритм циклической (4×4)-свертки с алгоритмом 3-точечной циклической свертки; этот алгоритм содержит 88 вещественных умножений и 608 вещественных сложений. Присоединяя 4-точечную циклическую свертку и 12-точечную циклическую свертку (20 вещественных умножений и 100 вещественных сложений) и учитывая одно связанное с компонен-

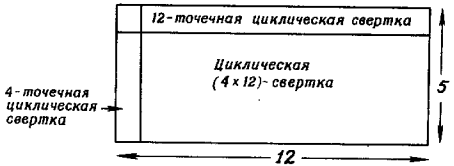


Рис. 8.14. Разбиение (5×13)-преобразования Фурье.

той $V_{0,0}$ тривиальное умножение, получаем (5×13)-БПФ-алгоритм, содержащий 114 вещественных умножений и 939 вещественных сложений.

(5×13)-БПФ-алгоритм с несколько лучшими характеристиками получается, если подпрограмму 13-точечного БПФ с 20 нетривиальными вещественными умножениями и 94 вещественными сложениями использовать вместо 12-точечной циклической свертки; тогда число вещественных умножений равно 114, а число необходимых вещественных сложений равно 917.

Описанный метод пригоден и в том случае, когда числа точек по осям равны степеням простых чисел. Разбиения множеств индексов на подмножества даются теперь более сложными правилами. Например, рассмотрим задачу вычисления двумерного (7×9)-преобразования Фурье. Так как число 9 не является простым, то для построения алгоритма нужен более общий, чем рассмотренный выше, метод. Напомним, что обобщая в гл. 4 алгоритм Рейдера на случай вычисления 9-точечной свертки, мы выбросили все не взаимно простые с 9 индексы и свели задачу к вычислению 6-точечной циклической свертки и двух 2-точечных циклических сверток. Поступая так же и теперь, мы получаем двумерную циклическую (6×6)-свертку и две двумерные циклические (2×6)-свертки. Таким образом, как показано на рис. 8.15, двумерный (7×9)-БПФ-алгоритм строится из двумерной циклической (6×6)-свертки, двух двумерных циклических (6×2)-сверток, 9-точечного БПФ-алгоритма и 6-точечной циклической свертки. В общей сложности такой алгоритм требует $52 + 2 \times 16 + 10 + 8 = 102$ умноже-

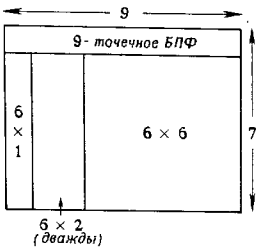


Рис. 8.15. Разбиение (7×9)-преобразования Фурье.

x^2+x+1	$x-1$	$x+1$	x^2-x+1	
6	0 (3)	0 (3)	6	y^2+y+1
3	0 (1)	0 (1)	3	$y-1$
3	0 (1)	0 (1)	3	$y+1$
6	0 (3)	0 (3)	6	y^2-y+1

Рис. 8.16. Разбиение двумерной (6×6)-свертки.

ний, но это число можно уменьшить, если более тщательно структурировать алгоритм. Мы не учли того факта, что 6-точечная свертка, получающаяся при преобразовании 9-точечного преобразования Фурье по алгоритму Рейдера, приводит к нескольким умножениям на нуль. Эти умножения, описываемые теоремой 4.6.2, можно не учитывать.

(6×6)-свертка возникает из двумерного аналога обобщенного алгоритма Рейдера. Для двумерного (7×9)-преобразования Фурье многочлен Рейдера от двух переменных равен

$$g(x, y) = \left[\sum_{l=0}^{5} (\omega^{3l} - 1)\, x^l\right] \left[\sum_{k=0}^{5} (\mu^{2k} - 1)\, y^k\right],$$

где ω — первообразный корень степени шесть из единицы, а μ — первообразный корень степени девять из единицы.

Для построения алгоритма циклической (6×6)-свертки запишем

$$x^6 - 1 = (x^2 + x + 1)(x - 1)(x + 1)(x^2 - x + 1),$$
$$y^6 - 1 = (y^2 + y + 1)(y - 1)(y + 1)(y^2 - y + 1),$$

и вычислим вычеты многочлена $g(x, y)$ по модулям, равным этим делителям. Как показано на рис. 8.16, эти вычисления можно разбить на 16 фрагментов. Согласно теореме 4.6.2, фрагменты, связанные с модулями $(y - 1)$ и $(y + 1)$, приводят к равным нулю вычетам, и, следовательно, могут быть выброшены. В квадратах на рис. 8.16 указаны числа умножений, необходимых при вычислении каждого фрагмента, причем в скобках приведено число умножений, необходимое в случае, когда вычет не равен нулю. Отсюда видно, что для вычисления циклической (6×6)-свертки требуется только 36 умножений, так что полное число умножений в алгоритме (7×9)-точечного БПФ равно 86.

8.6. Улучшенный алгоритм Винограда быстрого преобразования Фурье

Коль скоро задача вычисления одномерного преобразования Фурье с помощью алгоритма Гуда—Томаса сведена к задаче вычисления многомерного преобразования Фурье, то дальше можно ее решать сведением к многомерной свертке. Такой путь вычисления одномерного преобразования Фурье очень похож на большой БПФ-алгоритм Винограда, но с одним отличием. Напомним, что большой БПФ-алгоритм Винограда комбинирует два или больше малых БПФ-алгоритмов, но не меняет их структуру. Однако каждый из малых алгоритмов строится на основании быстрого алгоритма свертки, так что можно ожидать, что в большой задаче содержится многомерная свертка. Большой БПФ-алгоритм Винограда не пытается извлечь из многомерной свертки никакой пользы и он в конечном счете вычисляет ее, сочетая гнездовым способом два одномерных алгоритма. Если многомерная свертка достаточно велика, то характеристики можно улучшить, применяя для вычисления этой свертки более сильные алгоритмы. Характеристики $n'n''$-точечного улучшенного БПФ-алгоритма Винограда лучше, чем характеристики большого $n'n''$-точечного БПФ-алгоритма Винограда, если $\varphi(n')$ и $\varphi(n'')$ имеют общий делитель, равный по меньшей мере четырем. Характеристики для некоторых таких алгоритмов приведены на рис. 8.17.

В основе улучшенного алгоритма лежит сочетание трех идей: сведение одномерного преобразования Фурье к многомерному с помощью алгоритма Гуда—Томаса, многомерный аналог алгоритма Рейдера и использование полиномиального преобразования для вычисления многомерной свертки Рейдера. Первая из этих идей относится только к индексации данных. Сочетание второй идеи с третьей уже рассматривалось в предыдущем разделе при построении алгоритмов разложения. Для построения улучшенного алгоритма надо модифицировать один из двумерных алгоритмов, используя перестановку столбцов матрицы предсложений и перестановку строк матрицы постсложений.

Рассмотрим сначала 15-точечное преобразование Фурье. Его можно трансформировать в двумерное (3×5)-точечное преобразование Фурье. Используя далее алгоритм Рейдера вдоль каждой оси, выделим двумерную циклическую (2×4)-свертку, записываемую в виде

$$s(x, y) = g(x, y)\, d(x, y) \pmod{x^4 - 1} \pmod{y^2 - 1}.$$

Эта свертка так коротка, что наилучшим способом ее вычисления является гнездовое сочетание одномерных алгоритмов свертки. Для $n = 15$ улучшенный БПФ-алгоритм не лучше большого БПФ-

Длина n	Число вещественных умножений		Число вещественных сложений	Нетривиальных умножений на точку на выходе	Сложений на точку на выходе
	общее	нетривиальных			
15*	18	17	103	1.13	6.87
21*	27	26	184	1.24	8.76
35*	54	53	411	1.51	11.74
63	86	85	712	1.37	11.30
65	114	113	917	1.74	14.10
91	159	158	1560	1.75	17.14

Для комплексных входных данных все цифры удваиваются. Для длин, помеченных звездочкой, число умножений совпадает с таковым для большого БПФ-алгоритма Винограда.

Рис. 8.17. Характеристики улучшенного быстрого алгоритма преобразования Фурье.

алгоритма Винограда, а на самом деле даже хуже, так как содержит больше сложений.

Теперь рассмотрим 63-точечное преобразование Фурье. Большой БПФ-алгоритм Винограда содержит в этом случае 98 нетривиальных вещественных умножений и 704 вещественных сложений. Рассмотрим улучшенный БПФ-алгоритм, начинающийся сведением 63-точечного преобразования Фурье к двумерному (7×9)-преобразованию Фурье, для вычисления которого используется показанная на рис. 8.15 структура. Такой алгоритм содержит 85 нетривиальных вещественных умножений и 712 вещественных сложений.

8.7. Перестановочный алгоритм Нуссбаумера—Квенделла

В предыдущей главе, записав многомерную свертку в виде свертки многочленов, мы смогли воспользоваться полиномиальным представлением расширений полей для построения алгоритмов многомерных сверток. Теперь мы воспользуемся полиномиальным представлением расширений полей для того, чтобы построить алгоритмы вычисления многомерных преобразований Фурье. Одним из применений многомерных преобразований Фурье является, конечно, вычисление многомерных сверток, так что фактически мы построим другой способ вычисления многомерных сверток.

Еще раз нам удастся построить хорошие алгоритмы для одного класса задач, сведя этот класс к другому классу задач, для которого решение уже имеется. Такой подход может показаться чрезвычайно искусственным, так как сначала мы ввели полиномиальное преобразование как одну из форм записи преобразования Фурье, а затем отказались от этой точки зрения. Теперь мы

опять переворачиваем всю постановку задачи и используем полиномиальное преобразование для вычисления преобразования Фурье, хотя и не того самого, которое первоначально привело нас к определению полиномиального преобразования.

Перестановочный алгоритм Нуссбаумера — Квенделла представляет собой алгоритм многомерного быстрого преобразования Фурье. Он строится сведением многомерного преобразования к вычислению некоторого количества одномерных преобразований Фурье, что и отличает его от рассмотренных в разд. 8.5 алгоритмов разложения, в которых многомерное преобразование Фурье сводится к вычислению нескольких многомерных сверток. БПФ-алгоритм Нуссбаумера — Квенделла имеет лучшие характеристики, но применим только тогда, когда многомерное преобразование имеет одинаковую длину по всем измерениям или когда все длины имеют общий множитель, так что можно применить китайскую теорему об остатках. В настоящем разделе рассматривается БПФ-алгоритм Нуссбаумера — Квенделла; характеристики этого алгоритма для двумерного случая приведены на рис. 8.18, а для трехмерного случая — на рис. 8.19.

Алгоритм Нуссбаумера — Квенделла является алгоритмом вычисления многомерного преобразования Фурье в случае, когда число точек во всех измерениях равно одному и тому же числу n, которое либо просто, либо равно степени простого числа. Конструкции различны для трех случаев: n — простое число; n равно степени нечетного простого числа; n равно степени двойки, $n = 2^m$. Гнездовой метод позволяет из этих алгоритмов строить большие многомерные преобразования точно таким же образом, как это делается в случае одномерного преобразования Фурье. Например, для построения (63×63)-преобразования Фурье сначала оно отображается в четырехмерное ((7×9) × (7×9))-преобразование, которое затем рассматривается как ((7×7) × (9×9))-преобразование, для вычисления которого используется гнездовой метод сочетания (7×7)-алгоритма с (9×9)-алгоритмом. Это приводит к 49-кратному применению двумерного (9×9)-преобразования Фурье к 49 подтаблицам данных, за которым следует 81-кратное применение двумерного (7×7)-преобразования Фурье к 81 подтаблицам данных. Все алгоритмы могут быть приведены в стандартную форму в виде матрицы предсложений при условии, что каждая из (7×7)-подтаблиц сформирована как 49-компонентный вектор и каждая из (9×9)-подтаблиц сформирована как 81-компонентный вектор. (Все описанные манипуляции с входной (63×63)-таблицей относятся только к индексам. Они не содержат арифметических вычислений и не требуют никакой физической перестановки данных.) Далее, так же как и для одномерного случая, можно использовать теорему о кронекеровском произведении матриц для приведения вычислений к стандартному виду, выделяя все предсложения перед всеми умножениями и все умножения перед всеми постсложениями. Такой гнездовой метод построения алгоритма вычисления $(n'n'' \times n'n'')$-преобразования Фурье содержит

$$M(n'n'' \times n'n'') = M(n' \times n')\, M(n'' \times n'')$$

умножений. Число тривиальных умножений описывается равенством такого же типа; число сложений в алгоритме равно

$$A(n'n'' \times n'n'') = (n')^2 A(n'' \times n'') + M(n'' \times n'')\, A(n' \times n').$$

Вывод этих формул полностью совпадает с рассуждениями, проведенными в случае одномерного преобразования Фурье. Характеристики гнездовых БПФ-алгоритмов Нуссбаумера—Квенделла приведены на рис. 8.20.

Размер массива	Число вещественных умножений		Число вещественных сложений	Нетривиальных умножений на точку на выходе	Сложений на точку на выходе
	общее	нетривиальных			
2×2	4	0	8	0	2
3×3	9	8	36	0.89	4
4×4	16	0	64	0	4
5×5	31	30	221	1.20	8.84
7×7	65	64	635	1.31	12.96
8×8	64	24	408	0.375	6.37
9×9	105	104	785	1.28	9.69
16×16	304	216	2264	0.84	8.85

Для комплексных входных данных все цифры удваиваются

Рис. 8.18. Характеристики БПФ-алгоритмов Нуссбаумера—Квенделла.

Размер массива $n \times n \times n$	Число вещественных умножений		Число вещественных сложений	Нетривиальных умножений на точку на выходе	Сложений на точку на выходе
	общее	нетривиальных			
2×2×2	8	0	24	0.00	3.0
3×3×3	27	26	162	0.96	6.0
4×4×4	64	0	384	0.00	6.0
5×5×5	156	155	1 686	1.24	13.5
7×7×7	457	456	6 767	1.33	19.7
8×8×8	512	224	4 832	0.44	9.4
9×9×9	963	962	10 383	1.32	14.2
16×16×16	4992	3808	52 960	0.93	12.9

Для комплексного входа все цифры удваиваются

Рис. 8.19. Характеристики трехмерных БПФ-алгоритмов Нуссбаумера—Квенделла.

Размер массива		Число вещественных умножений		Число вещественных сложений	Нетривиальных умножений на точку на выходе	Сложений на точку на выходе
		общее	нетривиальных			
12	12	144	128	1 152	0.89	8.00
15	15	279	278	2 889	1.24	12.84
21	21	585	584	7 499	1.32	16.96
30	30	1 116	1 112	13 356	1.24	14.84
35	35	2 015	2 014	30 240	1.64	24.90
48	48	2 736	2 648	29 592	1.15	12.84
63	63	6 825	6 824	102 460	1.72	25.81
80	80	9 424	9 336	123 784	1.46	19.34
120	120	17 856	17 816	276 696	1.24	19.21
168	168	37 440	37 400	658 584	1.32	23.33
240	240	84 816	84 728	1 344 456	1.47	23.34
420	420	290 160	290 144	5 765 760	1.65	32.69
504	504	436 800	436 760	8 176 792	1.72	32.19
840	840	1 160 640	1 160 600	24 738 840	1.64	35.06
1008	1008	2 074 800	2 074 712	40 133 656	2.04	39.50
2520	2520	13 540 800	13 540 760	298 481 560	2.13	47.00

Для комплексных данных все цифры удваиваются

Рис. 8.20. Характеристики некоторых гнездовых БПФ-алгоритмов.

Изучаемые в настоящем разделе алгоритмы строятся сведением двумерного $(p \times p)$-преобразования Фурье к вычислению $(p+1)$ одномерных p-точечных преобразований Фурье для случая, когда число p простое. Для иллюстрации структуры, с которой мы будем работать, рассмотрим пример двумерного (3×3)-преобразования Фурье. Если записать входную и выходную (3×3)-таблицы по столбцам в виде векторов с 9 компонентами, то рассматриваемое (3×3)-преобразование запишется в виде

$$\begin{bmatrix} V_{0,0} \\ V_{1,0} \\ V_{2,0} \\ V_{0,1} \\ V_{1,1} \\ V_{2,1} \\ V_{0,2} \\ V_{1,2} \\ V_{2,2} \end{bmatrix} = \left[\begin{array}{ccc|ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & 1 & \omega & \omega^2 & 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega & 1 & \omega^2 & \omega & 1 & \omega^2 & \omega \\ \hline 1 & 1 & 1 & \omega & \omega & \omega & \omega^2 & \omega^2 & \omega^2 \\ 1 & \omega & \omega^2 & \omega & \omega^2 & 1 & \omega^2 & 1 & \omega \\ 1 & \omega^2 & \omega & \omega & 1 & \omega^2 & \omega^2 & \omega & 1 \\ \hline 1 & 1 & 1 & \omega^2 & \omega^2 & \omega^2 & \omega & \omega & \omega \\ 1 & \omega & \omega^2 & \omega^2 & 1 & \omega & \omega & \omega^2 & 1 \\ 1 & \omega^2 & \omega & \omega^2 & \omega & 1 & \omega & 1 & \omega^2 \end{array}\right] \begin{bmatrix} v_{0,0} \\ v_{1,0} \\ v_{2,0} \\ v_{0,1} \\ v_{1,1} \\ v_{2,1} \\ v_{0,2} \\ v_{1,2} \\ v_{2,2} \end{bmatrix},$$

где разделение на подматрицы сделано для того, чтобы выявить структуру кронекеровского произведения. Кратко это равенство записывается в виде $\mathbf{V} = \mathbf{W}\mathbf{v}$, где $\mathbf{v}$ и $\mathbf{V}$ обозначают выписанные выше входной и выходной векторы соответственно, а элементы матрицы $\mathbf{W}$ равны степеням элемента ω. Нашей целью является построение разложения $\mathbf{W} = \mathbf{CBA}$, где $\mathbf{A}$ и $\mathbf{C}$ представляют собой соответственно матрицы предсложений и постсложений, а матрица $\mathbf{B}$ задает набор одномерных преобразований Фурье. В данном примере матрица $\mathbf{B}$ имеет вид

$$\mathbf{B} = \left[\begin{array}{ccc|ccc|ccc} 1 & 1 & 1 & & & & & & \\ 1 & \omega & \omega^2 & & & & & & \\ 1 & \omega^2 & \omega & & & & & & \\ \hline & & & 1 & 1 & 1 & & & \\ & & & 1 & \omega & \omega^2 & & & \\ & & & 1 & \omega^2 & \omega & & & \\ \hline & & & & & & 1 & 1 & 1 \\ & & & & & & 1 & \omega & \omega^2 \\ & & & & & & 1 & \omega^2 & \omega \end{array}\right],$$

где незаполненным блокам соответствуют нулевые (3×3)-матрицы.

Разложение $\mathbf{W} = \mathbf{CBA}$ замечательно похоже на запись малого БПФ-алгоритма Винограда. Сейчас, однако, мы находимся на совершенно ином уровне: стоящая в центре разложения блочно-диагональная матрица представляет собой пакет одномерных преобразований Фурье, а не пакет умножений. Коль скоро такое разложение уже получено, двумерное (3×3)-преобразование Фурье вычисляется последовательным выполнением предсложений, задаваемых матрицей $\mathbf{A}$, четырехкратным обращением к 3-точечному одномерному БПФ и выполнением постсложений, задаваемых матрицей $\mathbf{C}$. Как мы увидим позже, для некоторых одномерных преобразований Фурье надо вычислять не все члены; эта обсуждаемая дальше процедура названа *выколотыми* БПФ-алгоритмами и позволяет еще немного уменьшить вычислительную нагрузку.

Одномерное преобразование Фурье можно трактовать как процесс вычисления значений многочлена $v(x)$ в точках ω^k:

$$V_k = v(\omega^k), \quad k = 0, \ldots, n-1,$$

где

$$v(x) = \sum_{i=0}^{n-1} v_i x^i.$$

Эта же трактовка применима и к двумерному преобразованию Фурье, причем более плодотворным оказалось использование такого описания только вдоль одного из измерений двумерного преобразования. Запишем $(n \times n)$-таблицу в виде вектора длины n, компонентами которого являются многочлены

$$v_{i''}(x) = \sum_{i'=0}^{n-1} v_{i',i''} x^{i'}, \quad i'' = 0, \ldots, n-1.$$

Определим одномерное преобразование Фурье вектора многочленов равенствами

$$V_{k''}(x) = \sum_{i''=0}^{n-1} \omega^{i''k''} v_{i''}(x), \quad k'' = 0, \ldots, n-1.$$

Тогда двумерное преобразование Фурье запишется в виде

$$V_{k', k''} = R_{x-\omega^{k'}}[V_{k''}(x)] = V_{k''}(\omega^{k'}), \quad \begin{matrix} k' = 0, \ldots, n-1, \\ k'' = 0, \ldots, n-1. \end{matrix}$$

В этом определении оси двумерного преобразования Фурье неравноправны и играют разную роль.

Теперь у нас уже все готово для того, чтобы, используя некоторую разумную технику перестановок, начать переход от преобразования Фурье к полиномиальному преобразованию. Пусть r обозначает некоторое положительное целое число, взаимно простое с n. Пусть $k'' = ((kr))_n$ для некоторого k из множества $\{0, \ldots, n-1\}$. Это задает неявный способ перебора всех значений индекса k'', так как в силу взаимной простоты r и n, когда k пробегает все значения от 0 до $n-1$, число k'' также пробегает все значения от 0 до $n-1$. Тогда, так как порядок элемента ω равен n, $\omega^{k''} = \omega^{kr}$, и два предыдущие равенства можно для любого r, взаимно простого с n, переписать в виде

$$V_{((kr))}(x) = \sum_{i''=0}^{n-1} \omega^{i''kr} v_{i''}(x),$$

$$V_{k', ((kr))} = R_{x-\omega^{k'}}[V_{((kr))}(x)], \quad \begin{matrix} k' = 0, \ldots, n-1, \\ k = 0, \ldots, n-1. \end{matrix}$$

Если k' само взаимно просто с n, то можно положить r равным k' и тогда второе равенство принимает вид

$$V_{k', ((kk'))} = R_{x-\omega^{k'}}[V_{((kk'))}(x)], \quad \begin{matrix} k = 0, \ldots, n-1. \\ \text{НОД}(k', n) = 1. \end{matrix}$$

Каждая компонента выходной таблицы, первый индекс которой взаимно прост с n, может быть записана таким образом.

Завершив на этом вводную часть, переходим к построению полиномиального преобразования. Следующая теорема представляет собой скачок прямо к окончательному результату. Читая эту теорему, необходимо иметь в виду, что все те k', которые относятся к одному и тому же круговому многочлену, требуют одних и тех же вычислений, что один и тот же индекс k' используется и в качестве выходного индекса, и для формирования перестановки, и что деление на k' определено, так как k' и n взаимно просты.

Теорема 8.7.1. *Пусть k' — фиксированное целое число, взаимно простое с n. Пусть $Q(x)$ обозначает круговой многочлен, корнем*

которого является элемент $\omega^{k'}$. Тогда вычисление компонент $V_{k', k''}$ для $k'' = 0, \ldots, n-1$ может быть реализовано в виде следующих трех шагов: (1) *вычислить полиномиальное преобразование*

$$V'_k(x) = \sum_{i''=0}^{n-1} v_{i''}(x)\, x^{i''k} \pmod{Q(x)}, \quad k = 0, \ldots, n-1;$$

(2) *вычислить n преобразований Фурье*

$$V''_{k', k} = \sum_{i'=1}^{n-2} \omega^{i'k'} V'_{i', k}, \quad \begin{matrix} k'\text{: НОД}(k', n) = 1, \\ k = 0, \ldots, n-1; \end{matrix}$$

(3) *сделать перестановку*

$$V_{k', k''} = V''_{k', ((k''/k'))}, \quad \begin{matrix} k'\text{: НОД}(k', n) = 1, \\ k'' = 0, \ldots, n-1. \end{matrix}$$

Доказательство. Рассмотрим равенства

$$V_{((kk'))}(x) = \sum_{i''=0}^{n-1} \omega^{i''kk'} v_{i''}(x),$$

$$V_{k', ((kk'))} = R_{x-\omega^{k'}}[V_{((kk'))}(x)].$$

Так как $x - \omega^{k'}$ делит круговой многочлен $Q(x)$, то второе из равенств не нарушится, если первое вычислять по модулю $Q(x)$. Это позволяет переписать равенства следующим образом:

$$V_{((kk'))}(x) = \sum_{i''=0}^{n-1} v_{i''}(x)\, \omega^{k'i''k} \pmod{Q(x)},$$

$$V_{k', ((kk'))} = R_{x-\omega^{k'}}[V_{((kk'))}(x)] = \sum_{i'=0}^{n-2} \omega^{i'k'} V'_{i', ((kk'))}.$$

Но теперь, согласно второму равенству, элемент $\omega^{k'}$ сравним с x. Поэтому замена $\omega^{k'}$ на x в первом равенстве не приведет к изменению окончательного результата. Это позволяет еще раз переписать исходные равенства в виде

$$V'_{((kk'))}(x) = \sum_{i''=0}^{n-1} v_{i''}(x)\, x^{i''k} \pmod{Q(x)},$$

$$V_{k', ((kk'))} = R_{x-\omega^{k'}}[V_{((kk'))}(x)] = \sum_{i'=0}^{n-2} \omega^{i'k'} V'_{i', ((kk'))}.$$

Теперь для завершения доказательства теоремы остается уточнить обозначения и переписать равенства в окончательном виде. □

Эту теорему надо применить к трем различным случаям, соответствующим равенству числа точек по каждой размерности двумерного преобразования простому числу, степени нечетного про-

Длина n	Полиномиальные преобразования	Преобразования Фурье	Дополнительные сложения
Простое число p	p-точечное преобразование по модулю $(x^p-1)/(x-1)$	$p+1$ преобразований* длины p	p^3+p^2-5p+4
Степень простого числа p^2	p^3-точечное преобразование по модулю $(x^{p^2}-1)/(x^p-1)$	p^2+p преобразований** длины p^2	$2p^5+p^4-5p^3+p^2+6$
	p-точечное преобразование по модулю $(x^{p^2}-1)/(x^p-1)$	$p+1$ преобразований* длины p	
	p-точечное преобразование по модулю $(x^p-1)/(x-1)$		
Степень двойки 2^m	2^m-точечное преобразование по модулю $x^{2^{m-1}}+1$	$(3/2)\,2^m$ преобразований** длины 2^m	$(3m+5)\,2^{2t-2}$
	2^m-точечное преобразование по модулю $x^{2^{m-1}}+1$	Одно двумерное преобразование размера $2^{m-1}\times 2^{m-1}$	

* Из которых p выколоты.
** Из которых все выколоты.

Рис. 8.21. Подпрограммы, используемые в БПФ-алгоритмах Нуссбаумера—Квенделла.

стого числа и степени двух. Рассмотрим последовательно все три случая; результаты анализа подытожены на рис. 8.21.

Число r точек в каждом измерении просто. При простом числе точек теоремой можно воспользоваться для того, чтобы свести вычисление двумерного $(p\times p)$-преобразования Фурье к $p+1$ различным одномерным преобразованиям Фурье. В более общем случае метод позволяет свести N-мерное, p-точечное по каждому измерению преобразование Фурье к $(p^N-1)/(p-1)$ различным одномерным p-точечным преобразованиям Фурье. Это можно сравнить с прямым методом вычисления N-мерного p-точечного по каждому измерению преобразования Фурье, содержащим Np^{N-1} различных одномерных p-точечных преобразований Фурье.

Для простых p имеет место разложение

$$x^p-1=(x-1)\,(x^{p-1}+x^{p-2}+\ldots+x+1).$$

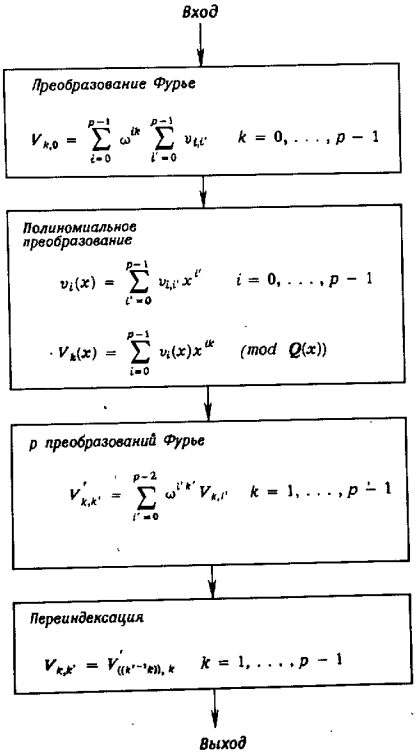


Рис. 8.22. Перестановочный алгоритм Нуссбаумера—Квенделла.

Каждое ненулевое k'' является корнем второго кругового многочлена. Согласно теореме 8.7.1, для вычисления компонент $V_{k',k''}$ при $k'=0,\ldots,p-1$ и $k''=1,\ldots,p$ надо вычислить одно полиномиальное преобразование и p одномерных преобразований Фурье длины p. Чтобы завершить вычисление, надо найти еще ком-

Выколотое преобразование Фурье

$$V_k = \sum_{i=1}^{p^{m-1}(p-1)-1} \omega^{ik} v_i, \quad k \not\equiv 0 \pmod{p},$$

где $\omega^n = 1$, $n = p^m$

Длина	Нормальное БПФ Винограда			Выколотое БПФ Винограда		
	Число умножений*		Число сложений	Число умножений*		Число сложений*
	общее	нетривиальных		общее	нетривиальных	
2	2	0	2			
3	3	2	6	2	2	4
4	4	0	8	2	0	2
5	6	5	17	5	5	15
7	9	8	36	8	8	34
8	8	2	26	4	2	10
9	11	10	44	8	8	28
16	18	10	74	10	8	32

* Для комплексных входных данных все цифры удваиваются

Рис. 8.23. Характеристики некоторых выколотых БПФ-алгоритмов.

поненты $V_{k',\,k''}$ с k'', равным нулю. Но они получаются сразу как результат еще одного преобразования Фурье:

$$V_{k',\,0} = \sum_{i'=0}^{p-1} \omega^{i'k'} \sum_{i''=0}^{p-1} v_{i',\,i''}, \quad k' = 0, \ldots, p-1.$$

Таким образом, всего получаем $p+1$ одномерных преобразований Фурье длины p. На рис. 8.22 приведена блок-схема алгоритма для простого p. На последнем шаге осуществляется обратная перестановка компонент, для чего используется обращение k' по модулю p.

Есть еще одно упрощение. У большинства преобразований Фурье некоторые коэффициенты отсутствуют, так что соответствующие преобразования можно упростить с помощью выкалывания — выбрасывая те входные компоненты, которые заведомо равны нулю, и те выходные компоненты, вычислять которые нет надобности. Как видно из блок-схемы на рис. 8.22, в основном шаге алгоритма, связанном с вычислением p-точечных преобразований Фурье векторов длины $p-1$, коэффициент старшего порядка всегда равен нулю и нужны только $p-1$ выходных компонент, а младший коэффициент можно отбросить. Такие выколотые БПФ-алгоритмы и их вычислительные характеристики представлены на рис. 8.23.

В качестве примера БПФ-алгоритма Нуссбаумера—Квенделла построим пример (3×3)-БПФ-алгоритма. Первый фрагмент выписывается проще всего:

$$\begin{bmatrix} V_{0,0} \\ V_{0,1} \\ V_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega \end{bmatrix} \begin{bmatrix} v_{0,0} + v_{1,0} + v_{2,0} \\ v_{0,1} + v_{1,1} + v_{2,1} \\ v_{0,2} + v_{1,2} + v_{2,2} \end{bmatrix}.$$

Теперь заметим, что круговой многочлен равен $Q(x) = x^2 + x + 1 = (x^3 - 1)/(x - 1)$ и

$$v_0(x) = v_{2,0}x^2 + v_{1,0}x + v_{0,0},$$
$$v_1(x) = v_{2,1}x^2 + v_{1,1}x + v_{0,1},$$
$$v_2(x) = v_{2,2}x^2 + v_{1,2}x + v_{0,2}.$$

Вычисление полиномиального преобразования по модулю $x^2 + x + 1$ приводит к многочленам первой степени. Хотя приведение по модулю $Q(x)$ можно и отложить, мы сразу заменим исходные многочлены их вычетами по модулю $Q(x)$:

$$v_0(x) = (v_{1,0} - v_{2,0})\,x + (v_{0,0} - v_{2,0}),$$
$$v_1(x) = (v_{1,1} - v_{2,1})\,x + (v_{0,1} - v_{2,1}),$$
$$v_2(x) = (v_{1,2} - v_{2,2})\,x + (v_{0,2} - v_{2,2}).$$

Полиномиальное преобразование равно

$$\begin{bmatrix} V_0(x) \\ V_1(x) \\ V_2(x) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & x & x^2 \\ 1 & x^2 & x \end{bmatrix} \begin{bmatrix} v_0(x) \\ v_1(x) \\ v_2(x) \end{bmatrix} \pmod{Q(x)}.$$

Это равенство можно переписать в виде

$$\left[\begin{array}{c} V_{0,0} \\ V_{1,0} \\ V_{0,1} \\ V_{1,1} \\ V_{0,2} \\ V_{1,2} \end{array}\right] = \left[\begin{array}{cc|cc|cc} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & -1 & -1 & 1 \\ 0 & 1 & 1 & -1 & -1 & 0 \\ \hline 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & -1 \end{array}\right] \left[\begin{array}{c} v_{0,0} - v_{2,0} \\ v_{1,0} - v_{2,0} \\ v_{0,1} - v_{2,1} \\ v_{1,1} - v_{2,1} \\ v_{0,2} - v_{2,2} \\ v_{1,2} - v_{2,2} \end{array}\right].$$

Следующий шаг состоит в вычислении результатов выколотых БПФ-алгоритмов. Вместо полного преобразования Фурье вида

$$\begin{bmatrix} V'_{0,k} \\ V'_{1,k} \\ V'_{2,k} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega \end{bmatrix} \begin{bmatrix} V_{0,k} \\ V_{1,k} \\ 0 \end{bmatrix}$$

нужны только выколотые преобразования Фурье

$$\begin{bmatrix} V''_{1,k} \\ V''_{2,k} \end{bmatrix} = \begin{bmatrix} 1 & \omega \\ 1 & \omega^2 \end{bmatrix} \begin{bmatrix} V'_{0,k} \\ V'_{1,k} \end{bmatrix},$$

при $k = 0, 1, 2$. Соединяя их вместе, получаем

$$\begin{bmatrix} V''_{1,0} \\ V''_{2,0} \\ V''_{1,1} \\ V''_{2,1} \\ V''_{1,2} \\ V''_{2,2} \end{bmatrix} = \left[\begin{array}{cc|cc|cc} 1 & \omega & 0 & 0 & 0 & 0 \\ 1 & \omega^2 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & \omega & 0 & 0 \\ 0 & 0 & 1 & \omega^2 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & \omega \\ 0 & 0 & 0 & 0 & 1 & \omega^2 \end{array}\right] \begin{bmatrix} V'_{0,0} \\ V'_{1,0} \\ V'_{0,1} \\ V'_{1,1} \\ V'_{0,2} \\ V'_{1,2} \end{bmatrix}.$$

Теперь надо выполнить обратную перестановку компонент:

$$\begin{bmatrix} V_{1,0} \\ V_{2,0} \\ V_{1,1} \\ V_{2,1} \\ V_{1,2} \\ V_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V''_{1,0} \\ V''_{2,0} \\ V''_{1,1} \\ V''_{2,1} \\ V''_{1,2} \\ V''_{2,2} \end{bmatrix}.$$

Наконец, собирая вместе все фрагменты, получаем окончательную форму алгоритма, показанную на рис. 8.24.

Число точек по осям равно степени простого нечетного числа. Для описания техники построения алгоритма достаточно рассмотреть случай, когда число точек по каждой оси двумерного преобразования равно квадрату нечетного простого числа. По существу техника остается той же, что и в первом случае, несколько усложняясь за счет того, что сначала надо отобрать индексы, идентифицирующие подлежащие отдельной обработке компоненты. В этом отношении очень полезной оказывается многочленная символика. Например, так как единственными делителями p^2 являются 1, p и p^2, то, согласно теореме 5.5.3, многочлен $x^{p^2} - 1$ разлагается по правилу

$$x^{p^2} - 1 = (x - 1)(x^{p-1} + x^{p-2} + \ldots + x + 1)(x^{p(p-1)} + \\ + x^{p(p-2)} + \ldots + x^p + 1) = Q_1(x)\, Q_p(x)\, Q_{p^2}(x)$$

на круговые многочлены. Индексы k разбиваются на три подмножества, соответствующие трем множествам сопряженных элементов, на которые корни из единицы, ω^k, разделяются круговыми многочленами. Воспользуемся теоремой 8.7.1 для обработки компонент, индексы которых входят в множество сопряженных элементов, соответствующее многочлену $Q_{p^2}(x)$. Это потребует выполнения одного полиномиального преобразования p^2 членов по модулю $Q_{p^2}(x)$ и p^2 выколотых p^2-точечных преобразований Фурье.

Для завершения этого фрагмента вычислений надо найти компоненты $V_{k',\,k''}$ с индексами k'', кратными p. Эти компоненты задаются равенствами

$$V_{k',lp} = \sum_{i'=0}^{p^2-1} \omega^{i'k'} \sum_{i''=0}^{p^2-1} \omega^{i''lp} v_{i',\,i''},$$

$$k' = 0, \ldots, p^2 - 1,$$

$$l = 0, \ldots, p - 1.$$

Внутреннюю сумму можно трансформировать в p-точечное преобразование Фурье. Обозначим через $\gamma = \omega^p$ корень степени p из единицы; пусть

$$v'_{i',\,r} = \sum_{l=0}^{p-1} v_{i',\,r+lp},$$

$$i' = 0, \ldots, p^2 - 1,$$

$$r = 0, \ldots, p - 1,$$

и пусть $V'_{k',\,l} = V_{k',lp}$. Тогда

$$V'_{k',\,l} = \sum_{i'=1}^{p^2-1} \omega^{i'k'} \sum_{r=0}^{p-1} \gamma^{rl} v'_{i',\,r},$$

$$k' = 0, \ldots, p^2 - 1,$$

$$l = 0, \ldots, p - 1.$$

Теперь поменяем порядок суммирования и опять применим теорему 8.7.1. Для этого нам потребуется еще одно полиномиальное преобразование p членов по модулю $Q_{p^2}(x)$ и еще p выколотых p^2-точечных преобразований Фурье. Остающееся при этом

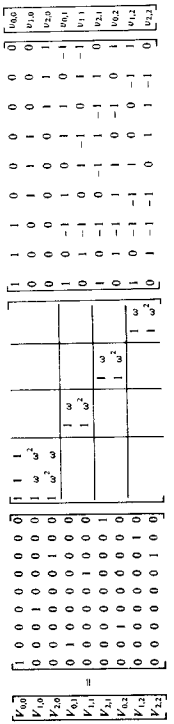


Рис. 8.24. (3×3)-БПФ-алгоритм Нуссбаумера—Квенделла.

двумерное $(p \times p)$-преобразование Фурье может быть вычислено как и ранее.

В общей сложности алгоритм содержит $p^2 + p$ одномерных p^2-точечных преобразований Фурье и $p + 1$ одномерных p-точечных преобразований Фурье. За исключением одного p-точечного преобразования все преобразования Фурье являются выколотыми.

Число точек по осям равно степени двух. Хотя конструкция для случая, когда число точек по осям двумерного преобразования равна степени двух, по существу совпадает с описанной выше, она все-таки отличается настолько, что заслуживает отдельного рассмотрения. Так как все нечетные меньшие чем 2^m целые числа взаимно просты с 2^m, то к компонентам с этими индексами опять применима теорема 8.7.1. Компоненты $V_{k', k''}$ для нечетных k' и $k'' = 0, \ldots, n - 1$ вычисляются применением n-точечного полиномиального преобразования по модулю $(x^{n/2} + 1)$ и n одномерных n-точечных преобразований Фурье при $n = 2^m$. В данном случае полиномиальное преобразование может быть организовано в виде БПФ-алгоритма Кули—Тьюки по основанию 2. Тогда число сложений будет пропорционально $n \log_2 n$. Все преобразования Фурье будут выколотыми с равными нулю входными компонентами с четными индексами.

Мы описали вычисление одной половины компонент $V_{k', k''}$, для которой k' нечетно и $k'' = 0, \ldots, n - 1$. Теперь поменяем ролями k' и k'' и вычислим оставшиеся невычисленными компоненты $V_{k', k''}$ с нечетными k'' и четными k'. Для этого нам понадобится еще одно полиномиальное преобразование и еще $n/2$ одномерных преобразований Фурье.

Наконец, остается вычислить компоненты $V_{k', k''}$, оба индекса которых четны. Но они образуют $(2^{m-1} \times 2^{m-1})$-точечное преобразование Фурье, вычисление которого можно организовать по той же схеме, что и вычисление $(2^m \times 2^m)$-точечного преобразования Фурье.

Задачи

8.1. Выписать таблицу для числа сложений и умножений алгоритма вычисления двумерного преобразования Фурье, использующего различные одномерные алгоритмы преобразования Фурье по строкам и по столбцам.

8.2. Для рассмотренных в разд. 8.1 БПФ-алгоритмов Кули — Тьюки по основанию 2 и 4 определить число необходимых сложений.

8.3. Пусть $A(n)$ и $M(n)$ обозначают число сложений и число умножений в БПФ-алгоритме Винограда длины n.

а. Доказать, что в гнездовом методе сочетания n'-точечного алгоритма с n''-точечным алгоритмом для минимизации числа сложений числа n' и n'' должны выбираться так, чтобы выполнялось условие

$$[M(n') - n']/A(n') \geqslant [M(n'') - n'']/A(n'').$$

б. Провести аналогичный анализ отдельно для числа предсложений и числа постсложений.

8.4. **а.** По заданным малым 2-точечному и 5-точечному БПФ-алгоритмам Винограда

$$\begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix},$$

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & -1 & 0 \\ 1 & 1 & -1 & 1 & 0 & 1 \\ 1 & 1 & -1 & -1 & 0 & -1 \\ 1 & 1 & 1 & -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & & & & & \\ & -1.25 & & & & \\ & & .559 & & & \\ & & & .951 & & \\ & & & & j.1538 & \\ & & & & & -j.363 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

построить большой 10-точечный БПФ-алгоритм Винограда.

б. Сколько вещественных умножений содержит этот алгоритм?

в. Сколько вещественных умножений будет содержать алгоритм, в котором 2-точечный и 5-точечный алгоритмы сочетаются по методу Гуда — Томаса?

г. Предположим, что этот 10-точечный БПФ-алгоритм используется по методу Кули — Тьюки для построения 1000-точечного БПФ-алгоритма. Сколько вещественных умножений будет содержать такой алгоритм? Как он соотносится с 1024-точечным БПФ-алгоритмом Кули — Тьюки по основанию 2?

д. Предположим, что вместо выписанного ранее 2-точечного алгоритма используется следующий алгоритм:

$$\begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix}$$

Что получится, проигрыш или выигрыш?

8.5. Для вычисления преобразований Фурье длин 72, 315 и 5040 можно воспользоваться большим БПФ-алгоритмом Винограда. Найти число необходимых сложений и число необходимых умножений в таких алгоритмах.

8.6. Построить алгоритм вычисления двумерного (256×256)-точечного преобразования Фурье, сочетая БПФ-алгоритм Кули — Тьюки с перестановочным алгоритмом Нуссбаумера — Квенделла. Сколько умножений и сложений содержит алгоритм?

8.7. Описать построение 1040-точечного БПФ-алгоритма, основанного на трехмерной циклической свертке. Сколько умножений необходимо?

8.8. Имеется много способов построения двумерного (63×63)-БПФ-алгоритма. Найти число умножений, необходимых в каждой из следующих процедур.

а. Гнездовой метод сочетания 63-точечного одномерного БПФ-алгоритма с самим собой. (Какие здесь имеются возможности выбора?)

б. Гнездовой метод сочетания двумерного (7×7)-БПФ-алгоритма с двумерным (9×9)-БПФ-алгоритмом.

в. Алгоритм разложения Нуссбаумера — Квенделла, сводящий задачу к четырехмерному (7×7×9×9)-БПФ-алгоритму.

8.9. Для каких значений n перестановочный алгоритм Нуссбаумера — Квенделла вычисления двумерного преобразования Фурье не приносит преимуществ по сравнению с гнездовым методом сочетания одномерных алгоритмов? Изменится ли вывод в случае трехмерных преобразований (если число точек простое)?

8.10. Описать структуру алгоритма разложения для вычисления двумерного (5×5)-преобразования Фурье, содержащую 230 сложений, и структуру, содержащую 236 сложений.

8.11. а. Доказать, что многочлен Рейдера от двух переменных $g(x, y)$ равен произведению многочлена от x на многочлен от y.

б. Сформулировать и доказать для двумерных преобразований Фурье аналог теоремы 4.6.2.

8.12. Построить (5×5)-БПФ-алгоритм Нуссбаумера — Квенделла.

8.13. Отталкиваясь от 4-точечного БПФ-алгоритма и двумерного (3×3)-БПФ-алгоритма, описанного в данной главе, построить двумерный (12×3)-БПФ-алгоритм.

Замечания

Простейший из алгоритмов вычисления двумерного преобразования Фурье сводится к последовательному выполнению одномерных преобразований по обеим осям двумерной таблицы. Поэтому все быстрые алгоритмы вычисления одномерных преобразований в готовом виде были перенесены на двумерный случай. Существенно двумерные алгоритмы возникли позже. Двумерное разбиение типа Кули — Тьюки было предложено Райвордом [1] (1977) и обобщено в работе Гарриса, Макклеллана, Чэна и Шусслера [2] (1977). Мерсере и Спик [3] (1981) описали изящную унификацию различных двумерных БПФ-алгоритмов Кули — Тьюки.

С целью создания более эффективной упаковки алгоритмов Виноград [4] (1976) предложил гнездовой метод соединения своих малых одномерных алгоритмов. Двумерные алгоритмы его интересовали как некоторое искусственное средство построения алгоритмов одномерных преобразований Фурье, но эта же техника применима и к возникающим естественным путем двумерным преобразованиям Фурье. Дальнейшее развитие гнездовой метод Винограда получил в работах Колбы и Паркса [5] (1977) и Сильвермена [6] (1977). Методы построения последовательности БПФ-алгоритмов, промежуточных по отношению к БПФ-алгоритмам Гуда — Томаса и БПФ-алгоритму Винограда, были рассмотрены Джонсоном и Баррасом [7] (1983). Использовать полиномиальное преобразование Нуссбаумера для вычисления преобразований Фурье предложили Нуссбаумер и Квенделл [8] (1979). Другую структуру алгоритма вычисления двумерного преобразования, основанную на связанных с циклическими структурами расширений полей Галуа перестановках, описали Ауслендер, Фейг и Виноград [10] (1983). Характеристики этого алгоритма близки к характеристикам БПФ-алгоритма Нуссбаумера — Квенделла.

Глава 9

АРХИТЕКТУРА ФИЛЬТРОВ И ПРЕОБРАЗОВАНИЙ

Сейчас, когда у нас уже есть такой большой набор алгоритмов цифровой обработки сигналов, пришло время обсудить их реализацию. Основной целью данной главы является рассмотрение вопросов реализации построенных алгоритмов на цифровых фильтрах. Будут рассмотрены также некоторые другие задачи, такие как интерполяция, прореживание и дискретное преобразование Фурье. Используя гнездовые и каскадные методы, мы построим из малых фрагментов большие структуры для обработки дискретных сигналов.

Наиболее важным устройством в обработке дискретных сигналов является КИО-фильтр. На фильтр поступает входной поток отсчетов цифровых данных, и на выходе фильтра формируется выходной поток отсчетов данных. Длины этих потоков данных очень велики; возможно, каждую секунду через фильтр проходят миллионы отсчетов. Быстрые алгоритмы фильтрации всегда разбивают поток входящих данных на пакеты, содержащие, возможно, сотни отсчетов. Единовременно обрабатывается только один пакет; результат обработки помещается в выходной буфер, данные из которого считываются с необходимой скоростью.

9.1. Вычисление свертки секционированием

В гл. 3 было построено много алгоритмов вычисления циклической свертки. Их можно использовать как непосредственно, так и в качестве модулей для построения хороших алгоритмов преобразования Фурье подобно тому, как это сделано в гл. 4. БПФ-алгоритмы, в свою очередь, могут быть использованы для построения эффективных алгоритмов вычисления свертки, особенно когда длина велика. В основе такого вычисления лежит теорема о свертке, согласно которой циклической свертке во временной области

$$s_i = \sum_{k=0}^{n-1} g_{((i-k))} d_k, \qquad i = 0, \ldots, n-1,$$

в частотной области соответствует произведение

$$S_k = G_k D_k, \qquad k = 0, \ldots, n-1.$$

Следовательно, одним из возможных путей вычисления циклической свертки является вычисление преобразования Фурье, покомпонентного произведения и обратного преобразования Фурье.

Обычно для такого вычисления длину преобразования Фурье выбирают равной длине свертки; но иногда этот способ оказывается полезным даже при нарушении столь разумного условия. Для БПФ-алгоритма может оказаться более подходящей другая длина преобразования.

Выберем удобную для построения преобразования Фурье длину n', такую, чтобы выполнялось условие $n' \geqslant 2n$, и удлиним векторы g, d и s так, чтобы компоненты с индексами $i = 0, \ldots, n-1$ принимали предписываемые им циклической сверткой длины n значения, несмотря на то, что вычисления производятся на удобной для преобразования Фурье длине n'. Например, определим

$$d'_i = \begin{cases} d_i, & i = 0, \ldots, n-1, \\ 0, & i = 0, \ldots, n'-1, \end{cases}$$

$$g'_i = \begin{cases} g_i, & i = 0, \ldots, n-1, \\ 0, & i = n, \ldots, n'-n-1, \\ g_{i+n-n'}, & i = n'-n, \ldots, n'-1, \end{cases}$$

и

$$s'_i = \sum_{k=1}^{n'-1} g'_{((i-k))} d'_k,$$

где теперь двойные скобки обозначают вычисление по модулю n'.

Тогда $s'_i = s_i$ для $i = 0, \ldots, n-1$. Остальные значения s'_i не представляют интереса и могут быть отброшены.

Имеется много методов сведения вычисления длинных линейных сверток к последовательности вычислений коротких циклических сверток. Повсеместно возникает необходимость фильтрации столь длинных последовательностей, что при прохождении через КИО-фильтр с конечным числом отводов их можно считать бесконечными. Такая фильтрация должна вычисляться по частям по ходу дела. Накопить все входные данные перед началом вычислений выходной последовательности невозможно, так как это приводит к задержкам и к тому же требует большой буферной памяти, и хорошие алгоритмы вычислений не используют всех входных данных сразу. Входные данные разделяются на сегменты, и, как только один сегмент становится полностью доступным, начинается его обработка.

Мы сначала опишем метод, известный под названием *метода перекрытия с накоплением*. Предположим, что у нас имеется устройство вычисления циклической свертки длины n, и что нам нужно умножить многочлен $g(x)$, степень A которого меньше n, на многочлен $d(x)$, степень B которого больше n. Обычно B очень велико и можно полагать его равным бесконечности. По многочлену $d(x)$ построим множество многочленов $\{d^{(0)}(x), d^{(1)}(x), \ldots\}$ степеней $n-1$ или меньше, полагая их коэффициенты равными

$$d_i^{(0)} = d_i, \qquad i = 0, \ldots, n-1,$$
$$d_i^{(1)} = d_{i+(n-A)}, \qquad i = 0, \ldots, n-1,$$
$$d_i^{(2)} = d_{i+2(n-A)}, \qquad i = 0, \ldots, n-1,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$

Многочленов должно быть достаточно для исчерпания всех коэффициентов многочлена $d(x)$. Заметим, что так построенные многочлены перекрывают друг друга. Для каждого l определим

$$s^{(l)}(x) = g(x)\, d^{(l)}(x) \pmod{x^n - 1}.$$

Тогда, так как $s(x) = g(x)\, d(x)$, то за исключением первых A коэффициентов, коэффициенты многочлена $s(x)$ находятся среди коэффициентов многочленов $s^{(l)}(x)$. А именно,

$$s_i = s_i^{(0)}, \qquad i = A, \ldots, n-1,$$
$$s_{i+(n-A)} = s_i^{(1)}, \qquad i = A, \ldots, n-1,$$
$$s_{i+2(n-A)} = s_i^{(2)}, \qquad i = A, \ldots, n-1$$

и так далее. Таким образом мы получаем на выходе фильтра бесконечную последовательность данных. Первые A младших коэффициентов многочлена $s(x)$ теряются. Во многих приложениях линейной свертки такая потеря в начале фильтрации является несущественной. Если же на выходе нужны все коэффициенты, то надо просто заменить многочлен $d(x)$ на многочлен $x^A d(x)$. При этом будут вычислены все коэффициенты свертки, но индексы их будут сдвинуты на величину A.

Каждая из рассмотренных циклических сверток за исключением последней вычисляет $n-A$ коэффициентов искомой линейной свертки и A бесполезных коэффициентов, которые просто отбрасываются. Поскольку степень последнего входного сегмента $d^{(r)}(x)$ может быть меньше $n-1$, то последняя свертка может содержать больше, чем $n-A$, значимых коэффициентов.

На самом деле в методе перекрытия с накоплением нет необходимости вычислять полную циклическую свертку, так как нужны не все ее коэффициенты, а только $n-A$ из них. Следовательно, подавляя ненужные коэффициенты, можно построить алгоритм, который несколько проще полного алгоритма вычисления циклической свертки. Алгоритмы описанного типа, независимо от того, проводится ли прямое вычисление или используется алгоритм вычисления циклической свертки, называются алгоритмами *секционной фильтрации*. Они будут изучаться в следующем разделе.

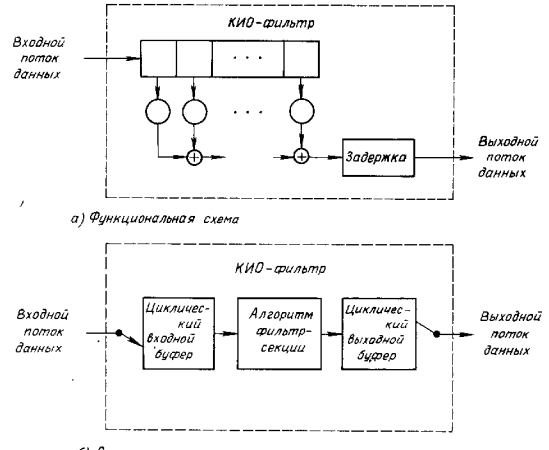


Рис. 9.1. Конструкция КИО-фильтра.

Основанная на методе перекрытия с накоплением конструкция КИО-фильтра показана на рис. 9.1. Функциональная схема такого фильтра похожа на линию задержки с отводами, но истинная конструкция может быть существенно другой. Поступающие входные слова подаются в циклическую буферную память, которая в типичных случаях вдвое больше секции фильтра. Во время фильтрации данных из одной части буфера, во вторую его часть записываются данные следующей секции, подготавливая блок данных для следующего шага вычислений. Текущая секция данных перекрывается с новой секцией. Фильтрация блока данных длины n занимает то же время, что и запись нового блока из $n - A$ точек.

Входной буфер является циклическим. Это означает, что после того, как его последний элемент заполнен, следующие входные символы записываются, начиная с первого элемента памяти. Таким образом, старые данные уничтожаются, а на их место записываются новые данные. Обычно удобно длину входного буфера выбирать равной степени двух. Записанное в двоичном регистре число указывает адрес, по которому запоминается следующий вход, и после каждого входа содержимое регистра увеличивается на единицу. Свойство цикличности буфера очень легко обеспечить тем,

что предусмотреть возможность переполнения адресного регистра после достижения состояния, в котором во всех его разрядах записаны единицы.

Секционный фильтр отыскивает в памяти блок данных, адреса которых равны $((b))$, ..., $((b + n - 1))$, где b — начальный адрес и двойные скобки обозначают вычисление по модулю, равному длине буфера памяти. После обработки секции данных b заменяется на $((b + A))$ и начинается фильтрация следующей секции.

Блок, формируемый на выходе секции фильтра, помещается в выходной буфер с помощью такой же техники, которая используется и во входном буфере. Поскольку для заполнения и очистки буферной памяти и для выполнения вычислений требуется время, то метод перекрытия с накоплением всегда приводит к некоторым задержкам. В приведенной на рис. 9.1 функциональной схеме эта задержка указана для того, чтобы сделать эквивалентными оба приведенных представления.

В методе перекрытия с накоплением используются перекрывающиеся блоки данных и часть каждого выходного блока отбрасывается. В основе метода лежит алгоритм вычисления циклической свертки той же самой длины, что и блоки входных данных. Другой метод, известный под названием *метода перекрытия с суммированием*, не использует перекрытия входных блоков данных, но основан на алгоритме вычисления линейной свертки, длина которой существенно больше длины входных блоков. В нем используются все компоненты выходных блоков и некоторое дополнительное множество вычислений, связывающих вместе отдельные выходные блоки. Оба эти метода иллюстрируются рис. 9.2.

Снова предположим, что у нас есть устройство вычисления линейной свертки, выходная длина которой равна n. Пусть нам надо умножить многочлен $g(x)$, степень A которого меньше n, на многочлен $d(x)$, степень B которого больше n. Величина B опять столь велика, что можно полагать ее равной бесконечности. По многочлену $d(x)$ построим последовательность многочленов степени не более $n - A - 1$, определяя их равенствами

$$d_i^{(1)} = d_i, \qquad i = 0, \ldots, n - A - 1,$$
$$d_i^{(2)} = d_{i+(n-A)}, \qquad i = 0, \ldots, n - A - 1,$$
$$d_i^{(3)} = d_{i+2(n-A)}, \qquad i = 0, \ldots, n - A - 1.$$
$$\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots$$

Отметим, что так построенные многочлены не перекрываются. Тогда многочлен $d(x)$ может быть записан в виде

$$d(x) = \sum_{l=1}^{\infty} d^{(l)}(x)\, x^{(l-1)(n-A)}$$

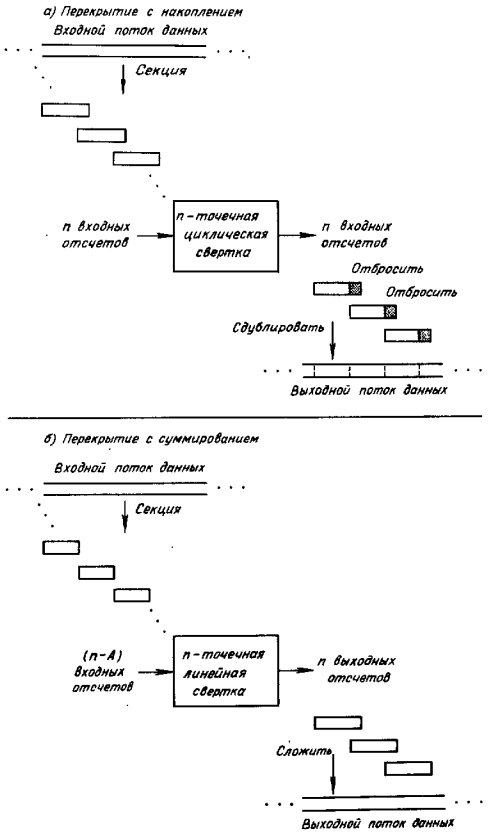


Рис. 9.2. Свертка по секциям.

и

$$s(x) = g(x)\, d(x) = \sum_{l=1}^{\infty} g(x)\, d^{(l)}(x)\, x^{(l-1)(n-A)}.$$

Для каждого l положим

$$s^{(l)}(x) = g(x)\, d^{(l)}(x).$$

Тогда коэффициенты многочлена $s(x)$ даются равенствами

$$s_i = s_i^{(1)}, \qquad i = 0, \ldots, n-A-1,$$

$$s_{i+(n-A)} = s_i^{(2)} + s_{i+(n-A)}^{(1)}, \qquad i = 0, \ldots, n-A-1,$$

$$s_{i+2(n-A)} = s_i^{(3)} + s_{i+(n-A)}^{(2)}, \qquad i = 0, \ldots, n-A-1,$$

и так далее. Этим исчерпываются все вычисления метода перекрытия с суммированием.

Метод перекрытия с суммированием основывается на алгоритме вычисления линейной свертки. Однако, поскольку $\deg g(x) = A$ и $\deg d^{(l)}(x) = n - A - 1$, то при желании можно воспользоваться и циклической сверткой. Циклическая свертка длины n на самом деле будет линейной сверткой; все выходные точки циклической свертки будут давать правильные значения коэффициентов линейной свертки.

Отметим, что и метод перекрытия с накоплением, и метод перекрытия с суммированием, для каждой вычисляемой n-точечной свертки используют $n - A$ выходных отсчетов. Метод перекрытия с накоплением обладает тем небольшим преимуществом, что после вычисления сверток не содержит сложений, и потому предпочтительнее. С другой стороны, метод перекрытия с суммированием основан на вычислении линейной свертки, вход которой содержит только $n - A$ ненулевых коэффициентов. Это позволяет построить алгоритм линейной свертки с меньшим числом сложений.

Например, в разд. 3.4 были рассмотрены алгоритмы итеративного построения линейных сверток большой длины из линейных сверток малой длины. В частности, для построения 256-точечной линейной свертки была использована итерация 2-точечной линейной свертки. Эта же задача может быть решена методом перекрытия с суммированием. Параметры алгоритма: $n = 511$ выходных точек; $A = 255$, что соответствует фильтру с 256 отводами; $n - A = 256$ входных точек. Каждая составляющая свертка дает пакет из 256 выходных точек, для вычисления которых в данной конструкции требуется 6561 умножений и 19171 сложений, считая дополнительные сложения алгоритма перекрытия с суммированием. Таким образом, реализация вычислений на КИО-фильтре с 256 отводами требует 25,6 умножений на одну точку на выходе.

9.2. Алгоритмы для коротких секций фильтра

Цифровые КИО-фильтры можно строить, используя описанные в предыдущем разделе методы перекрытия. Для применения этих методов нужны хорошие алгоритмы для отдельных секций фильтра. Сейчас мы дадим хорошие алгоритмы для коротких секций фильтра, а в разд. 9.3 опишем способы их сочетания для построения длинных секций фильтра.

Метод перекрытия с накоплением можно записать в матричном виде. Опуская отбрасываемые выходные компоненты, получаем следующее матричное уравнение:

$$\begin{bmatrix} s_{r-1} \\ s_r \\ \vdots \\ s_{n+r-2} \end{bmatrix} = \begin{bmatrix} d_{r-1} & \cdots & d_2 & d_1 & d_0 \\ d_r & \cdots & d_3 & d_2 & d_1 \\ d_{r+1} & & d_4 & d_3 & d_2 \\ & & & & \\ d_{n+r-2} & & & & d_{n-1} \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \end{bmatrix}.$$

Это уравнение описывает вычисление n компонент на выходе КИО-фильтра с r отводами. Такое вычисление будем называть *(r, n)-фильтр-секцией*, или, при $r = n$, *n-фильтр-секцией*. Эта задача представляет собой задачу вычисления усеченной свертки. Как будет доказано ниже, (r, n)-фильтр-секция требует $r + n - 1$ умножений, и поэтому алгоритм с $r + n - 1$ умножениями называется оптимальным. Этот термин, однако, может ввести в заблуждение, поскольку алгоритмы подобного сорта могут содержать слишком много сложений, если r и n не малы. Мы будем пользоваться оптимальными алгоритмами только при малых значениях r и n.

Алгоритм секционной фильтрации можно получить из алгоритма вычисления линейной свертки. Основой преобразования алгоритма для одной из этих задач в алгоритм для другой задачи является следующий общий принцип.

Теорема 9.2.1 (Трансформационный принцип). *Пусть задан алгоритм*

$$\mathbf{s} = \mathbf{Td} = \mathbf{CGAd},$$

где матрица $\mathbf{G}$ *является диагональной, а матрицы* $\mathbf{A}$ *и* $\mathbf{C}$ *содержат только малые константы. Тогда*

$$\mathbf{e} = \mathbf{A}^T\mathbf{GC}^T\mathbf{f}$$

является алгоритмом вычисления $\mathbf{e} = \mathbf{T}^T\mathbf{f}$. *Если исходный алгоритм вычисления вектора* $\mathbf{s}$ *оптимален, то второй алгоритм дает оптимальный алгоритм вычисления вектора* $\mathbf{e}$.

Доказательство. Алгоритм представляет собой факторизацию матрицы в виде

$$\mathbf{T} = \mathbf{CGA}.$$

Так как матрица $\mathbf{G}$ является диагональной, то $\mathbf{T}^T = \mathbf{A}^T\mathbf{GC}^T$ и, следовательно,

$$\mathbf{T}^T\mathbf{f} = \mathbf{A}^T\mathbf{GC}^T\mathbf{f}.$$

Вторая часть утверждения следует отсюда очевидным образом, так как если существует алгоритм с наименьшим числом умножений для вычисления вектора $\mathbf{e}$, то его трансформация дает алгоритм с тем же числом умножений для вычисления вектора $\mathbf{s}$. □

Воспользуемся трансформационным принципом для построения алгоритмов коротких фильтр-секций. Начнем с алгоритма линейной свертки

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} g_0 & 0 \\ g_1 & g_0 \\ 0 & g_1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_0 + g_1 & \\ & & g_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix}.$$

Согласно теореме 9.2.1 теперь можно записать

$$\begin{bmatrix} e_0 \\ e_1 \end{bmatrix} = \begin{bmatrix} g_0 & g_1 & 0 \\ 0 & g_0 & g_1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_0 + g_1 & \\ & & g_1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix}.$$

Для построения алгоритма 2-фильтр-секции заменим теперь (e_0, e_1) на (s_2, s_1) и (f_0, f_1, f_2) на (d_2, d_1, d_0):

$$\begin{bmatrix} s_2 \\ s_1 \end{bmatrix} = \begin{bmatrix} g_0 & g_1 & 0 \\ 0 & g_0 & g_1 \end{bmatrix} \begin{bmatrix} d_2 \\ d_1 \\ d_0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} g_0 & & \\ & g_0 + g_1 & \\ & & g_1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} d_2 \\ d_1 \\ d_0 \end{bmatrix}.$$

Число отводов	Число выходных точек	Число умножений	Число сложений
2	2	3	4
2	3	4	8
3	2	4	8
3	3	5	15
3	4	6	20
4	3	6	
4	4	7	
5	3	7	

Рис. 9.3. Характеристики некоторых алгоритмов коротких фильтр-секций.

При желании этот алгоритм после переупорядочивания можно переписать в виде

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} d_1 & d_0 \\ d_2 & d_1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} d_2 - d_1 & & \\ & d_1 & \\ & & d_0 - d_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}.$$

Для 2-фильтр-секции этот алгоритм оптимален; он содержит три умножения и четыре сложения, не считая суммирования $g_0 + g_1$, связывающего отводы фильтра.

Так же можно построить и другие алгоритмы коротких секций фильтра. Характеристики некоторых из таких алгоритмов приведены на рис. 9.3.

9.3. Итерирование секций фильтра

Итерируя алгоритмы для коротких секций фильтров, можно получать алгоритмы для секций фильтра большей длины, на выходе которых формируются длинные блоки отсчетов. Характеристики таких итерированных секций фильтра затабулированы на рис. 9.4. Используя описанный в разд. 9.1 метод перекрытия с накоплением, можно строить каскады произвольного числа этих секций фильтра, на выходе которых будет формироваться бесконечный поток данных.

В рассмотренных нами малых алгоритмах свертки не использовалось свойство коммутативности умножения чисел поля. Любой алгоритм линейной свертки,

$$s_t = \sum_{k=0}^{N-1} g_{t-k} d_k,$$

Число отводов	Число входных точек на секцию	Число умножений	Число сложений	Число умножений на точку на выходе	Число сложений на точку на выходе
3	3	5	15	1.33	5
9	9	25	120	2.78	13.33
16	16	81	260	5.06	16.25
27	27	125	735	4.63	27.22
32	32	243	844	7.59	26.37
81	81	625	4080	7.72	53.70

Рис. 9.4. Характеристики некоторых реальных алгоритмов КИО-фильтров.

в котором не используется ни деление, ни свойство коммутативности умножения, в такой же мере применим к элементам кольца, как к элементам поля. Пусть, например, надо вычислить свертку матриц

$$\mathbf{S}_t = \sum_{k=0}^{N-1} \mathbf{G}_{t-k} \mathbf{D}_k,$$

где теперь $\mathbf{D}_i$ представляет собой i-ю матрицу в списке из N матриц, а $\mathbf{G}_i$ представляет собой i-ю матрицу в списке из L матриц. Построенные нами ранее алгоритмы свертки применимы в данном случае так же, как и ранее. Сложение превращается в сложение матриц, а умножение становится умножением матриц.

Построим итерацию 2-фильтр-секций для формирования секций фильтра большей длины. Малый алгоритм возьмем в виде

$$\begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} (d_2 - d_1) & & \\ & d_1 & \\ & & (d_1 - d_0) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}.$$

Алгоритм содержит 1,5 умножений на каждый отсчет на выходе фильтра. Его можно использовать повторно для вычисления двух отсчетов на выходе одновременно, и такое вычисление всегда будет содержать 1,5 умножений и 2 сложения на один отсчет на выходе. 2-фильтр-секция, однако, не имеет существенного значения для практических приложений, а используется для итеративного построения алгоритмов больших фильтров с 2^m отводами, на выходе которых вычисляются одновременно 2^m отсчетов дискретного сигнала.

Для иллюстрации идеи рассмотрим построение алгоритма вычисления четырех точек на выходе секции фильтра с четырьмя

отводами (4-фильтр-секции). В матричном виде это вычисление записывается равенством

$$\begin{bmatrix} s_3 \\ s_4 \\ s_5 \\ s_6 \end{bmatrix} = \begin{bmatrix} d_3 & d_2 & d_1 & d_0 \\ d_4 & d_3 & d_2 & d_1 \\ d_5 & d_4 & d_3 & d_2 \\ d_6 & d_5 & d_4 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}.$$

Разбивая это вычисление на блоки размерности два на два, получаем

$$\begin{bmatrix} \mathbf{S}_1 \\ \mathbf{S}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{D}_0 \\ \mathbf{D}_2 & \mathbf{D}_1 \end{bmatrix} \begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \end{bmatrix},$$

где

$$\mathbf{D}_0 = \begin{bmatrix} d_1 & d_0 \\ d_2 & d_1 \end{bmatrix}, \quad \mathbf{D}_1 = \begin{bmatrix} d_3 & d_2 \\ d_4 & d_3 \end{bmatrix}, \quad \mathbf{D}_2 = \begin{bmatrix} d_5 & d_4 \\ d_6 & d_5 \end{bmatrix},$$

$$\mathbf{G}_0 = \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}, \quad \mathbf{G}_1 = \begin{bmatrix} g_2 \\ g_3 \end{bmatrix},$$

$$\mathbf{S}_1 = \begin{bmatrix} s_3 \\ s_4 \end{bmatrix}, \quad \mathbf{S}_2 = \begin{bmatrix} s_5 \\ s_6 \end{bmatrix}.$$

Теперь эта задача имеет точно такой же вид, как и предыдущая задача. Скалярные величины заменились матрицами, но алгоритм является системой тождеств и остается справедливым и в данном случае. Имеем

$$\begin{bmatrix} \mathbf{S}_1 \\ \mathbf{S}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{I} & -\mathbf{I} \\ \mathbf{I} & \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} (\mathbf{D}_2 - \mathbf{D}_1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (\mathbf{D}_1 - \mathbf{D}_0) \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \end{bmatrix}.$$

Как и ранее, сумма $\mathbf{G}_0 + \mathbf{G}_1$ должна быть вычислена заранее и не влияет на оценку числа необходимых в алгоритме операций. Алгоритм содержит четыре матричных сложения и три матричных умножения. Остановимся сначала на сложениях матриц. Два из них имеют вид

$$\mathbf{D}_2 - \mathbf{D}_1 = \begin{bmatrix} d_5 - d_3 & d_4 - d_2 \\ d_6 - d_4 & d_5 - d_3 \end{bmatrix},$$

$$\mathbf{D}_1 - \mathbf{D}_0 = \begin{bmatrix} d_3 - d_1 & d_2 - d_0 \\ d_4 - d_2 & d_3 - d_1 \end{bmatrix}.$$

Здесь имеется пять различных сложений. (Позже мы увидим, как одного из них можно избежать, так что останется только четыре сложения.) Еще здесь имеется два сложения 2-точечных векторов, выполняемых после завершения умножения; для их выполнения требуется четыре вещественных сложения.

Каждое из матричных умножений имеет тот же вид, что и 2-фильтр-секция, и, следовательно, может быть вычислено с помощью трех умножений и четырех сложений. Таким образом, алгоритм 4-фильтр-секции всего содержит девять умножений и 21 сложение.

Если алгоритм 4-фильтр-секции используется повторно для одновременного вычисления четырех отсчетов на выходе каждый раз, то одно из сложений пропадает, так как $d_6 - d_4$ в одном пакете равно $d_2 - d_0$ в следующем пакете.

Процесс итерации носит вполне общий характер. Если n четно, то матрица алгоритма n-фильтр-секции разбивается на четыре $((n/2) \times (n/2))$-блока. Используя теперь алгоритм 2-фильтр-секции, построим алгоритм n-фильтр-секции в виде трехкратного обращения к алгоритму $(n/2)$-фильтр-секции. При этом число дополнительных сложений, необходимых для вычисления матрицы $\mathbf{D}_1 - \mathbf{D}_0$, равно $n - 1$, а число дополнительных сложений, необходимых для вычисления матрицы $\mathbf{D}_2 - \mathbf{D}_1$, равно $n/2$. Это происходит потому, что все рассматриваемые матрицы являются теплицевыми $((n/2) \times (n/2))$-матрицами. Для вычисления матрицы $\mathbf{D}_1 - \mathbf{D}_0$ надо выполнить вычитания только в первой строке и левом столбце. Некоторые из полученных при этом величин повторяются в матрице $\mathbf{D}_2 - \mathbf{D}_1$, так что для вычисления последней требуется только $n/2$ вычитаний. Если алгоритм используется в режиме повторения, то некоторые из ранее вычисленных величин повторяются, так что для вычисления каждой из матриц $\mathbf{D}_1 - \mathbf{D}_0$ и $\mathbf{D}_2 - \mathbf{D}_1$ в каждом новом пакете требуется только $n/2$ сложений. Еще n дополнительных сложений требуется в выходном векторе. Подводя итог, получаем, что если алгоритм $(n/2)$-фильтр-секции содержит M умножений и A сложений, то построенный описанным образом алгоритм n-фильтр-секции будет содержать $3M$ умножений и $2n + 3A$ сложений.

Если $n = 2^m$, то эта процедура может быть проитерирована; алгоритм 2^m-фильтр-секции строится из алгоритмов 2^{m-1}-фильтр-секции, которые, в свою очередь, строятся из алгоритмов 2^{m-2}-фильтр-секции и т. д. Число умножений в таком алгоритме равно $3^m = n^{1,585}$, а число сложений описывается рекурсией $A(n) = 2n + 3A(n/2)$ при $A(1) = 0$.

Если n не равно степени двух, то в качестве блоков построения можно использовать секции фильтров других длин. Альтернативной возможностью является дополнение числа отводов фильтра до степени двух с помощью нулевых отводов.

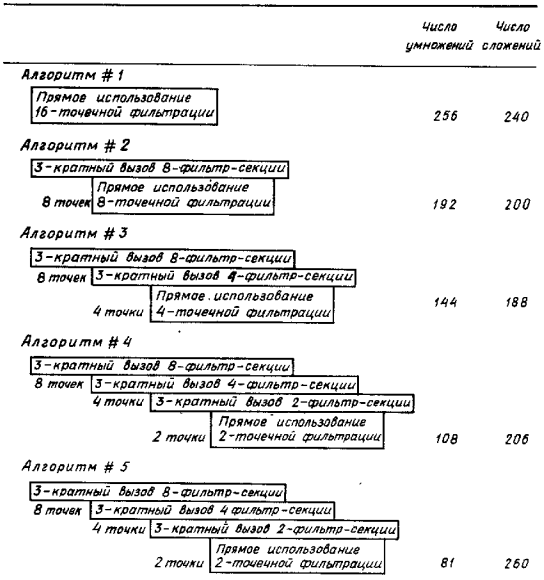


Рис. 9.5. Некоторые алгоритмы 16-точечных КИО-фильтров.

Если n достаточно велико, то рассматриваемый алгоритм с $n^{1,585}$ умножениями становится хуже алгоритма, основанного на БПФ и содержащего $O(n \log n)$ умножений. Значение n, для которого это происходит, можно сдвинуть, если итеративный алгоритм по основанию два заменить итеративным алгоритмом по основанию четыре, содержащим в своей начальной форме семь умножений. Тогда число умножений растет как $7^{m/2} = n^{1,404}$.

Для дальнейшей иллюстрации богатства конструктивных возможностей рассмотрим несколько алгоритмов вычисления 16 отсчетов фильтра с 16 отводами. Структура алгоритмов показана на рис. 9.5.

Алгоритм 1. Выходы фильтра вычисляются стандартным способом. Алгоритм содержит $16 \times 16 = 256$ умножений и $16 \times 15 = 240$ сложений.

Алгоритм 2. Алгоритм 2-фильтр-секции, в котором для вычисления выходов трех секций с 8 отводами используется любой подходящий алгоритм и 4×8 дополнительных сложений. Если выходы трех фильтров с 8 отводами вычисляются стандартным способом, то каждый из них требует $8 \times 8 = 64$ умножений и $8 \times 7 = 56$ сложений. В общей сложности получаем алгоритм с $3 \times 64 = 192$ умножениями и $32 + 3 \times 56 = 200$ сложениями.

Алгоритм 3. Алгоритм 2-фильтр-секции используется для сведения к трем 8-фильтр-секциям, каждая из которых реализуется в виде $4 \times 4 = 16$ сложений и трехкратного использования 4-фильтр-секций. Если каждая 4-фильтр-секция реализуется стандартным алгоритмом фильтрации, содержащим 16 умножений и 12 сложений, то такой алгоритм 16-фильтр-секции в общей сложности содержит $32 + 3 \times 16 + 9 \times 12 = 188$ сложений и $9 \times 16 = 144$ умножения.

Алгоритм 4 получается из алгоритма 3 модификацией алгоритмов для 4-фильтр-секций, в которой они заменены $4 \times 2 = 8$ дополнительными сложениями и трехкратным использованием алгоритма 2-фильтр-секции. Таким образом, этот алгоритм 16-фильтр-секции содержит $32 + 3 \times 16 + 9 \times 8 = 152$ сложения и 27-кратное вычисление 2-фильтр-секции. Стандартный алгоритм 2-фильтр-секции содержит два умножения и четыре сложения, так что в общей сложности получаем $152 + 27 \times 2 = 206$ сложений и $27 \times 4 = 108$ умножений.

Алгоритм 5 получается итеративным использованием 2-фильтр-секции на всех этапах; такой алгоритм содержит 260 сложений и 81 умножение.

Нельзя сказать, какой из этих алгоритмов предпочтительнее других. Только детальный анализ алгоритма в контексте его применения может сказать, что, например, алгоритм с 206 сложениями и 108 умножениями предпочтительнее, чем алгоритм с 260 сложениями и 81 умножениями. Все что мы можем сделать, это предложить конструктору различные альтернативы.

9.4. Симметрические и кососимметрические фильтры

Развитые в предыдущем разделе методы можно усилить, если коэффициенты задающего фильтр многочлена $g(x)$ обладают дополнительным специальным свойством, а именно, для построения хороших алгоритмов коротких секций фильтра можно воспользоваться симметрией коэффициентов многочлена фильтра. В методах, основанных на преобразовании Фурье, подобные тонкости использовать не удается.

Мы построим алгоритм для малых симметрических и кососимметрических фильтров. Комбинируя эти короткие секции, можно строить более длинные фильтры. Для увеличения размеров фильтра, к сожалению, нельзя использовать итерацию, так как фильтр перестает быть симметрическим. Возможен, однако, другой метод. Как мы увидим, вычет симметрического многочлена по модулю симметрического многочлена столь тесно связан с симметрией многочленов, что китайская теорема об остатках позволяет использовать эти свойства симметрии для построения алгоритмов для длинных симметрических фильтров из алгоритмов для коротких симметрических фильтров.

Фильтр с L отводами, описываемый многочленом $g(x)$, называется *симметрическим фильтром*, если многочлен $g(x)$ равен своему взаимному многочлену, т. е. если $g(x) = x^{L-1}g(x^{-1})$ или, эквивалентно, если $g_i = g_{L-1-i}$. Фильтр с L отводами, описываемый многочленом $g(x)$, называется *кососимметрическим фильтром*, если многочлен $g(x)$ равен своему взаимному многочлену со знаком минус, т. е. если $g(x) = -x^{L-1}g(x^{-1})$, или, эквивалентно, если $g_i = -g_{L-1-i}$. Мы начнем рассмотрение симметрических фильтров. Для симметрического фильтра с L отводами имеется очевидный алгоритм, содержащий для вычисления каждого выходного отсчета $(L-1)$ сложений и только $L/2$ умножений, если L четно, и $(L+1)/2$ умножений, если L нечетно. В этом очевидном алгоритме перед умножением на общий множитель g_i складываются точки с индексами i и $L-i-1$.

Алгоритм Винограда для свертки в случае симметрического фильтра строится точно так же, как и ранее. Построим алгоритм для вычисления выхода симметрического фильтра с тремя отводами, на вход которого подается 4-точечный вектор. Пусть

$$g(x) = g_2x^2 + g_1x + g_0,$$
$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

и

$$s(x) = g(x)\,d(x) \pmod{x^5 - x}\ (x - \infty)).$$

Поскольку $\deg s(x) = 5$, то приведение по такому модулю никак не сказывается на $s(x)$. Простыми делителями модуля являются четыре многочлена первой степени, каждый из которых в алгоритме приводит к одному умножению. Имеется также простой делитель $x^2 + 1$, который, как можно было бы ожидать, приводит к трем умножениям. Но, как мы увидим, в силу симметрии он вносит в алгоритм только два умножения. Пусть

$$s^0(x) = g^{(0)}(x)\,d^{(0)}(x) \pmod{x^2 + 1},$$

где

$$d^{(0)}(x) = (d_1 - d_3)\,x + (d_0 - d_2)$$

и

$$g^{(0)}(x) = g_1x + (g_0 - g_0) = g_1x.$$

Таким образом, для вычисления $s^{(0)}(x)$ нужно только два умножения. Остальная часть алгоритма остается без изменений и в окончательном виде алгоритм записывается равенством

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 & 0 & -1 \\ 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 \\ -1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} G_0 & & & & & \\ & G_1 & & & & \\ & & G_2 & & & \\ & & & G_3 & & \\ & & & & G_4 & \\ & & & & & G_5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix},$$

где

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & -\frac{1}{4} \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}.$$

Этот алгоритм содержит шесть умножений и 14 сложений. Его можно использовать в методе перекрытия с суммированием для реализации симметричной 3-фильтр-секции, содержащей 1.5 умножений и 4.5 сложений на каждую точку на выходе.

Даваемый теоремой 9.2.1 принцип трансформаций позволяет преобразовать этот алгоритм в алгоритм симметричной (4, 3)-фильтр-секции, содержащий 1.5 умножений и 4 сложения на каждую точку на выходе. Этот алгоритм дается равенством

$$\begin{bmatrix} s_5 \\ s_4 \\ s_3 \\ s_2 \end{bmatrix} = \begin{bmatrix} d_3 & d_4 & d_3 \\ d_4 & d_3 & d_2 \\ d_3 & d_2 & d_1 \\ d_2 & d_1 & d_0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} G_0 & & & & & \\ & G_1 & & & & \\ & & G_2 & & & \\ & & & G_3 & & \\ & & & & G_4 & \\ & & & & & G_5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & -1 & 1 & -1 & 1 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_3 \\ d_4 \\ d_3 \\ d_2 \\ d_1 \\ d_0 \end{bmatrix},$$

где

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & -\frac{1}{4} \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}.$$

Причина того, что в данном случае алгоритм для симметрического фильтра оказывается лучше, чем для несимметрического фильтра, кроется в том, что один коэффициент симметрического многочлена равен нулю. Это простейший пример общего феномена приведения симметрического многочлена по модулю симметрического многочлена. Пусть

$$g(x) = g_0x^4 + g_1x^3 + g_2x^2 + g_1x + g_0.$$

и рассмотрим вычеты

$$\begin{aligned} g^{(1)}(x) &= g(x) \pmod{x^2+1},\\ g^{(2)}(x) &= g(x) \pmod{x^2+x+1},\\ g^{(3)}(x) &= g(x) \pmod{x^2-x+1},\\ g^{(4)}(x) &= g(x) \pmod{x^4+1}. \end{aligned}$$

Легко проверить, что эти вычеты равны

$$\begin{aligned} g^{(1)}(x) &= 2g_0 - g_2,\\ g^{(2)}(x) &= (g_0+g_1-g_2)x + (g_0+g_1-g_2) =\\ &= (x+1)(g_0+g_1-g_2),\\ g^{(3)}(x) &= (-g_0+g_1+g_2)x - (-g_0+g_1+g_2) =\\ &= (x-1)(-g_0+g_1+g_2),\\ g^{(4)}(x) &= g_1x^3 + g_2x^2 + g_1x = x(g_1x^2 + g_2x + g_0). \end{aligned}$$

В каждом случае число независимых коэффициентов на два меньше, чем в соответствующем модуле $m^{(i)}(x)$. В силу симметрии один из независимых коэффициентов пропадает. Вообще говоря, вычет представляется в виде произведения симметрического многочлена степени $\deg m^{(i)}(x) - 2$ на дополнительный многочлен, не содержащий неопределенных коэффициентов. Так как нас интересуют только вычисления вида

$$s^{(i)}(x) = g^{(i)}(x)\, d^{(i)}(x) \pmod{m^{(i)}(x)},$$

то мы можем запрятать этот дополнительный множитель, «спрятав» его в переопределенном остатке $d^{(i)}(x)$. Пусть

$$\begin{aligned} d^{(1)}(x) &= d(x) \pmod{x^2+1},\\ d^{(2)}(x) &= (x+1)\,d(x) \pmod{x^2+x+1},\\ d^{(3)}(x) &= (x-1)\,d(x) \pmod{x^2-x+1},\\ d^{(4)}(x) &= x\,d(x) \pmod{x^4+1}. \end{aligned}$$

Тогда вычеты $g^{(i)}(x)$ переопределятся по правилам

$$\begin{aligned} g^{(1)}(x) &= 2g_0 - g_2,\\ g^{(2)}(x) &= g_0 + g_1 - g_2, \end{aligned}$$

$$\begin{aligned} g^{(3)}(x) &= -g_0 + g_1 + g_2,\\ g^{(4)}(x) &= g_1x^2 + g_2x + g_1. \end{aligned}$$

Для вычисления каждого из остатков $s^{(1)}(x)$, $s^{(2)}(x)$ и $s^{(3)}(x)$ потребуется два умножения, а остаток $s^{(4)}(x)$ надо вычислять как линейную свертку с последующим приведением по модулю x^4+1. Эта линейная свертка представляет собой прохождение 4-точечного вектора через симметрический фильтр с тремя отводами. Алгоритм такой фильтрации с шестью умножениями мы уже построили.

Используя построенные фрагменты, можно формировать алгоритмы линейной свертки для симметрического фильтра с пятью отводами. Предположим, что требуется построить алгоритм вычисления последовательности на выходе такого фильтра, если на его вход подается 6-точечная последовательность. Тогда выберем

$$m(x) = x(x-1)(x+1)(x-\infty)(x^2+1)(x^2-x+1) \times \\ \times (x^2+x+1).$$

Каждый из первых четырех многочленов приводит в алгоритме к одному умножению, а каждый из остальных трех — к двум умножениям. Окончательный алгоритм будет содержать десять умножений (1.67 умножений на один отсчет на выходе).

Предположим, что требуется пропустить через этот же фильтр 14-точечную последовательность. Тогда к использовавшемуся ранее многочлену $m(x)$ добавим еще один множитель, x^4+1. Это добавит в алгоритм еще шесть умножений. Окончательный алгоритм будет содержать 16 умножений (1.14 умножений на один отсчет на выходе).

В общем случае необходимы симметрические фильтры с числом отводов, большим пяти. Следующая теорема говорит о том, как надо выбирать модули для того, чтобы строить алгоритмы для больших симметрических фильтров из малых симметрических фильтров.

Теорема 9.4.1. *Пусть четное число t не меньше четного n и пусть $r(x)$ обозначает остаток от деления симметрического многочлена $g(x)$ степени t на симметрический многочлен $m(x)$ степени n над основным полем G. Тогда $r(x)$ можно записать в виде*

$$r(x) = f(x)\, r'(x) \pmod{m(x)},$$

где $r'(x)$ представляет собой симметрический многочлен степени $n-2$ и $f(x)$ — некоторый многочлен над основным полем G (не зависящий, следовательно, от коэффициентов многочлена $g(x)$).

Доказательство. Без потери общности можно полагать, что многочлен $m(x)$ приведен. Так как $m(x)$ является симметрическим

многочленом степени n, то $m(x) = x^n m(x^{-1})$. Если $n = t$, то положим $m'(x) = m(x)$; в противном случае, пусть

$$m'(x) = m(x) + x^{t-n} m(x).$$

Этот многочлен является симметрическим степени t, так как

$$x^t [m(x^{-1}) + x^{-t+n} m(x^{-1})] = x^{t-n} m(x) + m(x).$$

Определим многочлен $g'(x)$ равенством

$$xg'(x) = g(x) - g_0 m'(x).$$

Многочлен $xg'(x)$ при делении на $m(x)$ дает тот же остаток, что и многочлен $g(x)$, но многочлен $g'(x)$ является симметрическим и его степень равна четному числу, которое по меньшей мере на два меньше степени многочлена $g(x)$. Если степень многочлена $g'(x)$ не меньше n, то этот процесс можно повторить до тех пор, пока не получим многочлен $r'(x)$ степени $n-2$. Тогда остаток от деления многочлена $x^{(t-n+2)/2} r'(x)$ на $m(x)$ равен остатку от деления $g(x)$ на $m(x)$. Для завершения доказательства положим $f(x) = R_{m(x)}[x^{(t-n+2)/2}]$. □

Теперь мы перейдем к алгоритмам для кососимметрических фильтров. С этой задачей мы справимся значительно скорее, указав, как алгоритмы для симметрических фильтров можно превратить в алгоритмы для кососимметрических фильтров.

Теорема 9.4.2. *Пусть кососимметрический фильтр содержит L отводов; тогда, если L четно, то эта задача фильтрации может быть решена по алгоритму для симметрического фильтра с $L-1$ отводами, а если L нечетно, то эта задача может быть решена по алгоритму для симметрического фильтра с $L-2$ отводами.*

Доказательство. Предположим, что многочлен $g(x)$ описывает кососимметрический фильтр с четным числом L отводов. Тогда $g(1) = 0$, так как $g_i = -g_{L-1-i}$ для $i = 0, \ldots, (L/2) - 1$. Следовательно, $(x-1)$ делит многочлен $g(x)$. Определим фильтр

$$g'(x) = g(x)/(x-1).$$

Ясно, что многочлен $g'(x)$ задает симметрический фильтр с $L-1$ отводами. Симметричность фильтра вытекает из того, что равенство $g(x) = -x^{L-1} g(x^{-1})$ влечет за собой равенство

$$(x-1) g'(x) = -x^{L-1} \left(\frac{1}{x} - 1\right) g'\left(\frac{1}{x}\right),$$

или равенство

$$g'(x) = x^{L-2} g'(x^{-1}).$$

Таким образом, фильтр, описываемый многочленом $g(x)$, эквивалентен фильтру с многочленом $g'(x)$, за которым следует

(или которому предшествует) фильтр с двумя отводами, описываемый многочленом $(x-1)$.

Теперь предположим, что многочлен $g(x)$ задает кососимметрический фильтр с нечетным числом L отводов. Тогда 1 и -1 являются корнями многочлена $g(x)$, так как центральный коэффициент равен нулю и

$$g(\pm 1) = \left(\sum_{i \text{ чет.}} g_i + g_{L-1-i}\right) \pm \left(\sum_{i \text{ нечет}} g_i + g_{L-1-i}\right) = 0.$$

Следовательно, можно определить симметричный фильтр с $L-2$ отводами, задавая его многочленом

$$g'(x) = g(x)/(x^2 - 1).$$

Таким образом, фильтр, описываемый многочленом $g(x)$, эквивалентен фильтру, задаваемому многочленом $g'(x)$, которому предшествует (или за которым следует) фильтр с многочленом $x^2 - 1$. □

Фильтры, задаваемые многочленами $x-1$ или x^2-1, приводят к некоторым дополнительным сложениям на входе или выходе. Следовательно, изменяя соответствующим образом матрицу предложений или матрицу постсложений, мы преобразуем алгоритм для симметрического фильтра в алгоритм для кососимметрического фильтра. Как всегда, если этот алгоритм используется в методе перекрытия с суммированием, то его следует брать в форме алгоритма для линейной свертки. Напротив, если в конструкции используется метод перекрытия с накоплением, то следует воспользоваться даваемым теоремой 9.2.1 принципом трансформаций для преобразования алгоритма в алгоритм секционной фильтрации.

Как оказалось, тот же метод, который позволяет свести задачу для кососимметрических фильтров к задаче для симметрических фильтров, позволяет при нечетном числе L отводов свести задачу для симметрического фильтра с $L+1$ отводами к задаче для симметрического фильтра с L отводами.

Теорема 9.4.3. *Если L нечетно, то алгоритм для симметрического фильтра с $L+1$ отводами получается из алгоритма для симметрического фильтра с L отводами без дополнительных умножений.*

Доказательство. Пусть $(L+1)$-точечный фильтр задается симметрическим многочленом $g(x)$ нечетной степени L. Тогда $g(-1) = 0$, так что $(x+1)$ делит многочлен $g(x)$. Следовательно, $g(x) = (x+1) g'(x)$, где $g'(x)$ — симметрический многочлен степени $L-1$. Таким образом, фильтр, задаваемый многочленом $g(x)$, может быть реализован в виде каскада фильтров, задаваемых многочленами $x+1$ и $g'(x)$, без дополнительных умножений. □

9.5. Фильтры прореживания и интерполяции

Фильтром прореживания называется КИО-фильтр, который вычисляет только каждый r-й выходной отсчет; часто r равно 2 или 3. Все прочие выходные отсчеты нежелательны и подавляются, даже если они вычисляются. Можно надеяться на существование более простых алгоритмов, не вычисляющих ненужные выходные точки. *Фильтр интерполяции* действует противоположным образом: отсчеты на выходе такого фильтра формируются с большей скоростью, чем на входе. Прореживающий и интерполирующий фильтры иногда называются *фильтрами с подавлением* и *фильтром с восстановлением* соответственно.

Конструкции прореживающего и интерполирующего КИО-фильтров аналогичны общим конструкциям КИО-фильтров. Однако, их специфические особенности можно использовать для построения более эффективных алгоритмов. В первом случае более эффективные алгоритмы можно строить из-за выбрасываемых выходных точек, а во втором — из-за выбрасываемых входных точек.

Для фильтров прореживания алгоритмы линейной свертки и алгоритмы секционной фильтрации не связаны между собой описываемым теоремой 9.2.1 трансформационным принципом. Трансформационный принцип преобразует алгоритм свертки для прореживающего фильтра в алгоритм секционной фильтрации для интерполирующего фильтра.

Рассмотрим алгоритм линейной свертки для прореживающего в отношении 2 : 1 фильтра с пятью отводами, на вход которого подается 5-точечный вектор. Необходимый выходной вектор дается вычислением

$$\begin{bmatrix} s_0 \\ s_2 \\ s_4 \\ s_6 \\ s_8 \end{bmatrix} = \begin{bmatrix} g_0 & 0 & 0 & 0 & 0 \\ g_2 & g_1 & g_0 & 0 & 0 \\ g_4 & g_3 & g_2 & g_1 & g_0 \\ 0 & 0 & g_4 & g_3 & g_2 \\ 0 & 0 & 0 & 0 & g_4 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} g_0 & 0 & 0 \\ g_2 & g_0 & 0 \\ g_4 & g_2 & g_0 \\ 0 & g_4 & g_2 \\ 0 & 0 & g_4 \end{bmatrix} \begin{bmatrix} d_0 \\ d_2 \\ d_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ g_3 & 0 \\ g_1 & g_3 \\ 0 & g_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_3 \end{bmatrix}.$$

Алгоритм записан в виде вычисления 3-точечной линейной свертки, 2-точечной линейной свертки и трех дополнительных сложений, сочетающих результаты этих сверток. Таким образом, рассмотренный алгоритм прореживающей фильтрации содержит восемь умножений и 26 сложений на каждые пять точек на выходе. Если этот алгоритм использовать в методе перекрытия с суммированием, то для сочетания секций, каждая из которых содержит пять входных точек, потребуется еще четыре дополнительных сложения. Такой алгоритм будет содержать восемь умножений и 30 сло

жений на каждые пять выходных точек (1,6 умножений и 6 сложений на одну точку на выходе).

Теперь рассмотрим такую же задачу для секции прореживающего фильтра с пятью отводами и тремя выходными отсчетами. Алгоритм для такой секции можно записать равенством

$$\begin{bmatrix} s_4 \\ s_6 \\ s_8 \end{bmatrix} = \begin{bmatrix} d_4 & d_3 & d_2 & d_1 & d_0 \\ d_6 & d_5 & d_4 & d_3 & d_2 \\ d_8 & d_7 & d_6 & d_5 & d_4 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} = \begin{bmatrix} d_4 & d_2 & d_0 \\ d_6 & d_4 & d_2 \\ d_8 & d_6 & d_4 \end{bmatrix} \begin{bmatrix} g_0 \\ g_2 \\ g_4 \end{bmatrix} + \begin{bmatrix} d_3 & d_1 \\ d_5 & d_3 \\ d_7 & d_5 \end{bmatrix} \begin{bmatrix} g_1 \\ g_3 \end{bmatrix}.$$

В этом алгоритме с помощью трех дополнительных сложений сочетаются алгоритмы 3-фильтр-секции и (3, 2)-фильтр-секции, так что в общей сложности этот алгоритм прореживающей фильтрации содержит девять умножений и 26 сложений.

Посмотрим, какие алгоритмы получаются из построенных алгоритмов при их трансформации. Мы увидим, что в обоих случаях это приведет к алгоритмам интерполирующих фильтров. Трансформация выписанного выше алгоритма линейной свертки дается равенством

$$\begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} g_0 & g_2 & g_4 & 0 & 0 \\ 0 & g_1 & g_3 & 0 & 0 \\ 0 & g_0 & g_2 & g_4 & 0 \\ 0 & 0 & g_1 & g_3 & 0 \\ 0 & 0 & g_0 & g_2 & g_4 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}.$$

Заменим теперь $(e_0, e_1, e_2, e_3, e_4)$ на $(s_8, s_7, s_6, s_5, s_4)$ и $(f_0, f_1, f_2, f_3, f_4)$ на $(d_8, d_6, d_4, d_2, d_0)$. Тогда

$$\begin{bmatrix} s_8 \\ s_7 \\ s_6 \\ s_5 \\ s_4 \end{bmatrix} = \begin{bmatrix} g_0 & g_2 & g_4 & 0 & 0 \\ 0 & g_1 & g_3 & 0 & 0 \\ 0 & g_0 & g_2 & g_4 & 0 \\ 0 & 0 & g_1 & g_3 & 0 \\ 0 & 0 & g_0 & g_2 & g_4 \end{bmatrix} \begin{bmatrix} d_8 \\ d_6 \\ d_4 \\ d_2 \\ d_0 \end{bmatrix} = \begin{bmatrix} d_8 & 0 & d_6 & 0 & d_4 \\ 0 & d_6 & 0 & d_4 & 0 \\ d_6 & 0 & d_4 & 0 & d_2 \\ 0 & d_4 & 0 & d_2 & 0 \\ d_4 & 0 & d_2 & 0 & d_0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}.$$

Мы получили уравнения для интерполирующего фильтра, поскольку нулю теперь равны входные точки. Таким образом, применение трансформационного принципа к прореживающей в отношении 2 : 1 5-фильтр-секции привело к интерполирующей в отношении 2 : 1 5-фильтр-секции. Алгоритм содержит восемь умножений.

Таким же образом, алгоритм прореживающей 5-фильтр-секции, записываемый равенством

$$\begin{bmatrix} s_8 \\ s_6 \\ s_4 \end{bmatrix} = \begin{bmatrix} g_0 & g_1 & g_2 & g_3 & g_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & g_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 & g_4 \end{bmatrix} \begin{bmatrix} d_8 \\ d_7 \\ d_6 \\ d_5 \\ d_4 \\ d_3 \\ d_2 \\ d_1 \\ d_0 \end{bmatrix},$$

может быть трансформирован к алгоритму

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{bmatrix} = \begin{bmatrix} g_0 & 0 & 0 \\ g_1 & 0 & 0 \\ g_2 & g_0 & 0 \\ g_3 & g_1 & 0 \\ g_4 & g_2 & g_0 \\ 0 & g_3 & g_1 \\ 0 & g_4 & g_2 \\ 0 & 0 & g_3 \\ 0 & 0 & g_4 \end{bmatrix} \begin{bmatrix} d_0 \\ d_2 \\ d_4 \end{bmatrix}$$

Этот алгоритм является алгоритмом линейной свертки, входные элементы в которой через один равны нулю, — алгоритм интерполирующей фильтрации, содержащий девять умножений.

Следующим рассмотрим (на примере) алгоритм симметричной прореживающей фильтрации. Построим алгоритм симметрического прореживающего в отношении 2 : 1 15-фильтра. Как и в предыдущем примере, разобъем задачу на симметрическую линейную 8-точечную свертку и симметрическую линейную 7-точечную свертку. Согласно теореме 9.4.3 первая из них может быть вычислена с помощью алгоритма фильтрации 8-точечного входа в фильтре с 7 отводами.

Для вычисления обеих линейных симметрических сверток воспользуемся разработанным в предыдущем параграфе методом. Степень многочлена

$$m(x) = x(x-1)(x+1)(x-\infty)(x^2+1)(x^2+x+1)(x^2-x+1)(x^4+1)$$

равна 14 и, следовательно, им можно воспользоваться для построения алгоритма линейной фильтрации 8-точечного входа в фильтре с 7 отводами. В разд. 9.4 было определено число умножений, связанных с использованием в качестве модуля каждого из указанных симметрических многочленов. Это приводит к об-

щему числу умножений в алгоритме, равному 16. Аналогично, выбрасывая один линейный множитель, получаем алгоритм фильтрации 7-точечного входа в фильтре с 7 отводами, содержащий 15 умножений. Следовательно, для решения рассматриваемой задачи можно построить алгоритм, содержащий 31 умножение.

9.6. Автокорреляция и взаимная корреляция

Выражение вида

$$r_i = \sum_{k=0}^{N-1} g_{i+k} d_k, \qquad i = 0, \ldots, L+N-2,$$

называемое *корреляцией*, если $\mathbf{g}$ совпадает с $\mathbf{d}$, и *взаимной корреляцией*, если $\mathbf{g}$ и $\mathbf{d}$ различны, очень тесно связано со сверткой. Его можно сделать похожим на свертку, если просто прочитать одну из последовательностей в обратном порядке. В принципе, любой из рассмотренных нами методов вычисления линейной свертки годится для вычисления корреляции. Тем не менее, имеется одно практически существенное различие. Длина КИО-фильтра обычно намного меньше, чем длина фильтруемой последовательности данных, а входящий в корреляцию «фильтр» $g(x)$ на самом деле представляет собой другую последовательность данных, длина которой примерно равна длине последовательности, обозначаемой многочленом $d(x)$; часто длина записи обеих последовательностей неопределена и весьма велика — возможно, несколько сотен отсчетов. Хорошие алгоритмы разбивают $\mathbf{g}$ и $\mathbf{d}$ на секции.

Мы начнем рассмотрение с вычисления автокорреляции

$$r_i = \frac{1}{N} \sum_{k=0}^{N-1} d_k d_{k+i}, \quad i = 0, \ldots, n/2-1,$$

возможно, опуская в сумме деление на N. В интересующих нас задачах длина N блока данных значительно больше длины $n/2$ блока выходных данных, так что использование для вычислений преобразования Фурье длины N было бы слишком расточительным. Разобьем сумму на части по n слагаемых в каждой из частей, и будем решать задачу, используя преобразование Фурье длины n. Запишем

$$r_i = \sum_{l=0}^{L-1} r_i^{(l)},$$

где

$$r_i^{(l)} = \frac{1}{N} \sum_{k=0}^{n/2-1} d_{k+ln/2} d_{k+ln/2+i}, \qquad \begin{matrix} i = 0, \ldots, n/2-1, \\ l = 0, \ldots, L-1, \end{matrix}$$

и $L\,(n/2) = N$. Таким образом, задача свелась к вычислению вектора $\mathbf{r}^{(l)}$ для $l = 0, \ldots, L-1$, который мы будем находить, используя БПФ и подходящую циклическую свертку.

Определим два вида блоков длины n — один состоящий полностью из данных, и второй, состоящий наполовину из нулей. Позже будет показано, как избегать блоков первого вида. Пусть

$$d_i^{(l)} = \begin{cases} d_{l+ln/2}, & i = 0, \ldots, n/2-1, \\ 0, & i = n/2, \ldots, n-1, \end{cases}$$

и

$$g_i^{(l)} = d_{l+ln/2}, \qquad i = 0, \ldots, n-1.$$

Тогда

$$r_i^{(l)} = \sum_{k=0}^{n-1} d_k g_{k+i}, \quad i = 0, \ldots, n/2-1.$$

Пусть $\mathbf{D}^{(l)}$ и $\mathbf{G}^{(l)}$ обозначают векторы, полученные преобразованием Фурье из векторов $\mathbf{d}^{(l)}$ и $\mathbf{g}^{(l)}$. Тогда согласно задаче 1.10, $S_k^{(l)} = D_k^{(l)*} G_k^{(l)}$ равно преобразованию Фурье циклической корреляции

$$s_i^{(l)} = \sum_{k=0}^{n-1} d_k g_{((l+k))}, \quad i = 0, \ldots, n-1,$$

и половина из этих компонент дает искомые нами величины

$$r_i^{(l)} = \frac{1}{N} s_i^{(l)}, \quad i = 0, \ldots, n/2-1.$$

Таким образом, сначала мы вычисляем вектор

$$S_k = \sum_{l=0}^{L-1} D_k^{(l)*} G_k^{(l)}, \quad k = 0, \ldots, n-1,$$

а затем вычисляем его обратное преобразование Фурье $\mathbf{s}$ и полагаем $r_i = (1/N)\, s_i$ для $i = 0, \ldots, n/2-1$.

Для завершения описания алгоритма покажем, как исключить некоторые вычисления. Вместо использования БПФ для вычисления $\mathbf{G}^{(l)}$ воспользуемся формулой

$$G_k^{(l)} = D_k^{(l)} + (-1)^k D_k^{(l+1)}.$$

Эта формула является прямым следствием свойства задержки для преобразования Фурье; умножение в частотной области на ω^{kb} соответствует временно́му сдвигу на b позиций. Если $b = n/2$, то это соответствует умножению $D_k^{(l)}$ на $(-1)^k$. Во временно́й

области происходит сдвиг ненулевых позиций вектора $\mathbf{d}^{(l+1)}$ в нулевые позиции вектора $\mathbf{d}^{(l)}$; во временно́й области сумма равна $\mathbf{g}^{(l)}$, и, следовательно, в частотной области сумма равна $\mathbf{G}^{(l)}$.

Таким образом, это вычисление можно записать в виде

$$S_k = \sum_{l=0}^{L-1} D_k^{(l)} \left[D_k^{(l)} + (-1)^k D_k^{(l+1)}\right].$$

Алгоритм завершается вычислением обратного преобразования Фурье.

Блок-схема вычислений в окончательном виде показана на рис. 9.6. Отметим, что каждый цикл содержит только один БПФ-алгоритм, хотя в начальной точке и в последней точке алгоритма добавляется еще по одному БПФ. Таким образом, хотя вычисление одной циклической свертки содержит три преобразования Фурье, мы построили алгоритм, который в среднем на цикл содержит только одно БПФ.

Все L циклов на рисунке показаны одинаковыми, хотя в последнем из них вычисляется преобразование Фурье для нулевого вектора, добавленного в конец последовательности данных; это преобразование Фурье можно исключить, модифицировав последний цикл вычислений.

Аналогичный метод применим и к задаче вычисления взаимной корреляции. Обе последовательности данных разобьем на отрезки, комбинируя которые в частотной области, мы уменьшаем число необходимых обратных преобразований Фурье.

Теперь на одном очень подробном примере покажем, как можно использовать другие методы в рассматриваемом способе вычисления корреляций. Запишем корреляцию $s_i = \sum_{k=0}^{r-1} d_{k+i} g_k$ в матричном виде,

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{n-1} \end{bmatrix} = \begin{bmatrix} d_0 & d_1 & d_2 & \ldots & d_{r-1} \\ d_1 & d_2 & d_3 & \ldots & d_r \\ d_2 & d_3 & d_4 & \ldots & d_{r+1} \\ \vdots & & & & \\ d_{n-1} & d_n & d_{n+1} & \ldots & d_{r+n-2} \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{r-1} \end{bmatrix},$$

где r намного больше n.

Мы опишем структуру алгоритма для частного случая $n = 120$ и $r = 10\,000$. Основываясь на некотором опыте, примем произвольное решение разбить последовательность из 120 выходных точек на четыре пакета по 30 выходных точек в каждом, и организуем вычисления, опираясь на 60-точечный БПФ-алгоритм Винограда.

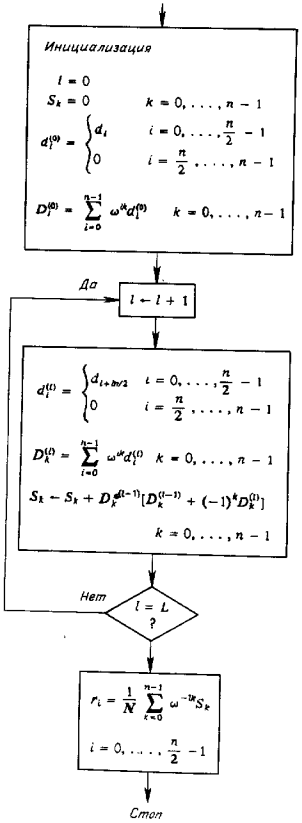


Рис. 9.6. Вычисление автокорреляций.

В таком блочном виде матричное уравнение принимает форму

$$\begin{bmatrix} s_0 \\ \vdots \\ s_{29} \\ s_{30} \\ \vdots \\ s_{59} \\ s_{60} \\ \vdots \\ s_{89} \\ s_{90} \\ \vdots \\ s_{119} \end{bmatrix} = \begin{bmatrix} d_0 & \ldots & d_{29} & d_{30} & \ldots & d_{59} & \ldots & d_{9990} & \ldots & d_{10019} \\ \vdots & & \vdots & \vdots & & \vdots & & & & \\ d_{29} & \ldots & d_{59} & d_{60} & \ldots & d_{89} & \ldots & d_{10019} & \ldots & d_{10049} \\ d_{30} & \ldots & d_{60} & d_{90} & \ldots & d_{119} & & & & \\ \vdots & & \vdots & \vdots & & \vdots & & & & \\ d_{59} & \ldots & d_{89} & d_{119} & \ldots & d_{148} & & & & \end{bmatrix} \begin{bmatrix} g_0 \\ \vdots \\ g_{29} \\ g_{30} \\ \vdots \\ g_{59} \\ \vdots \\ g_{9990} \\ \vdots \\ g_{10019} \end{bmatrix},$$

где к $g(x)$ и к $d(x)$ добавлены 20 нулевых компонент так, чтобы новое значение $r = 10\,020$ делилось на 30. Каждый из малых блоков, например, первый

$$\begin{bmatrix} s_0 \\ \vdots \\ s_{29} \end{bmatrix} = \begin{bmatrix} d_0 & \ldots & d_{29} \\ \vdots & & \vdots \\ d_{29} & \ldots & d_{58} \end{bmatrix} \begin{bmatrix} g_0 \\ \vdots \\ g_{29} \end{bmatrix}$$

можно вычислить, используя 60-точечный БПФ-алгоритм Винограда и взяв в качестве ответа первые 30 выходных точек.

Теперь рассмотрим блочную структуру

$$\begin{bmatrix} \mathbf{S}_0 \\ \mathbf{S}_1 \\ \mathbf{S}_2 \\ \mathbf{S}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{D}_0 & \mathbf{D}_1 & \mathbf{D}_2 & \ldots & \mathbf{D}_{333} \\ \mathbf{D}_1 & \mathbf{D}_2 & \mathbf{D}_3 & \ldots & \mathbf{D}_{334} \\ \mathbf{D}_2 & \mathbf{D}_3 & \mathbf{D}_4 & \ldots & \mathbf{D}_{335} \\ \mathbf{D}_3 & \mathbf{D}_4 & \mathbf{D}_5 & \ldots & \mathbf{D}_{336} \end{bmatrix} \begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \\ \mathbf{G}_2 \\ \vdots \\ \mathbf{G}_{333} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{D}_0 & \mathbf{D}_1 & \mathbf{D}_2 & \mathbf{D}_3 \\ \mathbf{D}_1 & \mathbf{D}_2 & \mathbf{D}_3 & \mathbf{D}_4 \\ \mathbf{D}_2 & \mathbf{D}_3 & \mathbf{D}_4 & \mathbf{D}_5 \\ \mathbf{D}_3 & \mathbf{D}_4 & \mathbf{D}_5 & \mathbf{D}_6 \end{bmatrix} \begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \\ \mathbf{G}_2 \\ \mathbf{G}_3 \end{bmatrix} + \begin{bmatrix} \mathbf{D}_4 & \mathbf{D}_5 & \mathbf{D}_6 & \mathbf{D}_7 \\ \mathbf{D}_5 & \mathbf{D}_6 & \mathbf{D}_7 & \mathbf{D}_8 \\ \mathbf{D}_6 & \mathbf{D}_7 & \mathbf{D}_8 & \mathbf{D}_9 \\ \mathbf{D}_7 & \mathbf{D}_8 & \mathbf{D}_9 & \mathbf{D}_{10} \end{bmatrix} \begin{bmatrix} \mathbf{G}_4 \\ \mathbf{G}_5 \\ \mathbf{G}_6 \\ \mathbf{G}_7 \end{bmatrix}$$

$$+ \ldots + \begin{bmatrix} \mathbf{D}_{332} & \mathbf{D}_{333} & \mathbf{D}_{334} & \mathbf{D}_{335} \\ \mathbf{D}_{333} & \mathbf{D}_{334} & \mathbf{D}_{335} & \mathbf{D}_{336} \\ \mathbf{D}_{334} & \mathbf{D}_{335} & \mathbf{D}_{336} & \mathbf{D}_{337} \\ \mathbf{D}_{335} & \mathbf{D}_{336} & \mathbf{D}_{337} & \mathbf{D}_{338} \end{bmatrix} \begin{bmatrix} \mathbf{G}_{332} \\ \mathbf{G}_{333} \\ \mathbf{G}_{334} \\ \mathbf{G}_{335} \end{bmatrix},$$

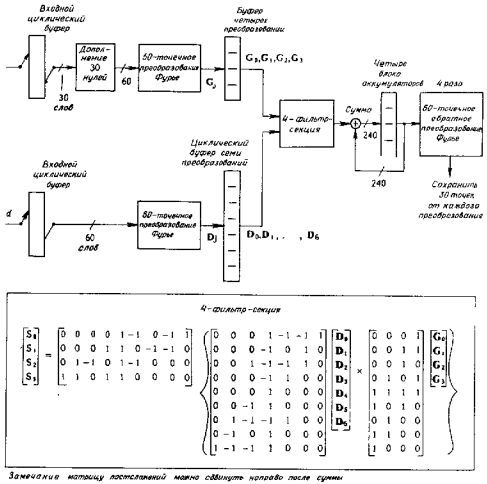


Рис. 9.7. Коррелятор.

где опять дописаны нулевые блоки так, чтобы получалось число блоков, кратное четырем. Теперь надо построить алгоритм вычисления вида

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} D_0 & D_1 & D_2 & D_3 \\ D_1 & D_2 & D_3 & D_4 \\ D_2 & D_3 & D_4 & D_5 \\ D_3 & D_4 & D_5 & D_6 \end{bmatrix} \begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \end{bmatrix}.$$

Это вычисление идентифицируется как 4-фильтр-секция. То, что элементами, над которыми выполняются вычисления, являются матрицы, роли не играет; можно воспользоваться построенным в разд. 9.2 алгоритмом, содержащим девять умножений и 21 сложение. Каждое из умножений является на самом деле 30-фильтр-секцией, которую мы уже решили вычислять через 60-точечный БПФ-алгоритм Винограда. Это вычисление должно быть повторено 84 раза, после чего для вычисления нужного ответа надо просуммировать все полученные ответы.

Окончательная форма этого алгоритма вычисления корреляции показана на рис. 9.7. Символами $\mathbf{D}_j$ и $\mathbf{G}_j$ на рисунке обозначены преобразования Фурье блоков отрезков данных $\mathbf{d}_j$ и $\mathbf{g}_j$. Те преобразования, которые используются более одного раза, сохраняются, а не перевычисляются. Большинство сложений выполняются в частотной области, хотя мы описывали их во временной области. Это позволяет уменьшить число необходимых обратных преобразований Фурье. Можно еще несколько уменьшить число требуемых сложений, если умножение на матрицу постсложений в каждом из алгоритмов секций отложить до того момента, пока не будут просуммированы выходы всех фильтр-секций. Тогда эти сложения будут выполняться только один раз.

Теперь обсудим сложность такой реализации рассматриваемого вычисления, имея в виду число умножений. В общей сложности алгоритм содержит 676 обращений к 60-точечному БПФ-алгоритму Винограда, что приводит к 48 672 умножениям. Алгоритм содержит также 84-кратное использование алгоритма для 4-фильтр-секции, каждое из которых содержит девять векторных умножений, представляющих собой покомпонентное произведение 60-точечных комплексных векторов, для вычисления каждого из которых, в свою очередь, необходимо три вещественных умножения. Это приводит к 146 080 умножениям. Таким образом, полное число умножений в рассматриваемом алгоритме вычисления 120 точек корреляции последовательностей из 10 000 отсчетов равно 194 752. Для задач, в которых требуется вычисление высокоскоростной корреляции, — типа задач обработки радиолокационных сигналов, — приведенный на рис. 9.7 алгоритм можно реализовать в виде дискретно-логического устройства.

9.7. Устройства для вычисления БПФ

Хороший БПФ-алгоритм можно использовать успешно только тогда, когда удается реализовать заложенные в нем возможности. Конструируя аппаратурный модуль или подпрограмму, необходимо тщательно анализировать структуру алгоритма.

БПФ-алгоритмы Кули—Тьюки по основанию два и четыре обладают регулярной структурой и допускают относительно прямую реализацию. Алгоритм задается несколькими инструкциями. Поэтому, если в программной реализации длина памяти, в которой записываются инструкции, является более важным параметром, чем время выполнения программы, то БПФ-алгоритму Кули—Тьюки следует отдать предпочтение перед другими, более быстрыми алгоритмами.

Те же соображения применимы и к аппаратурной реализации. Регулярность архитектуры устройства для вычисления преобразования может оказаться важнее общего числа компонент. Тогда опять следует отдать предпочтение БПФ-алгоритму Кули—

Тьюки в силу его регулярности. Регулярность БПФ-алгоритма Кули—Тьюки очевидным образом вытекает из его блок-схемы, показанной на рис. 9.8 для случая прореживания по времени. На всех шагах итерации используется простое 2-точечное преобразование Фурье. Основной вычислительный модуль состоит из 2-точечного дискретного преобразования Фурье и фазовращателя. Показанная на рис. 9.9 диаграмма этого модуля определила его название: он называется *2-точечной бабочкой*. Алгоритм Кули—Тьюки по основанию два с прореживанием по частоте, показанный на рис. 9.10, также строится на основе подобной бабочки, диаграмма которой приведена на рис. 9.11.

Если 2-точечная бабочка задана, в виде ли программной реализации, в виде ли аппаратурного модуля, то БПФ-алгоритм Кули—Тьюки по основанию два с прореживанием по времени сводится просто к определенной последовательности прохождения входных данных через 2-точечную бабочку. При каждом таком прохождении вычисляется соответствующее преобразование Фурье, но индексы данных переставляются. На рис. 9.8 указана перестановка на входных данных, которая предопределяет вычисление выходных данных в естественном порядке. При желании можно входные данные подавать в естественном порядке, но тогда пересечения линий на диаграмме примут более сложный вид.

При написании программы БПФ-алгоритма Кули—Тьюки удобнее выбрать форму, в которой вычисляемые выходные данные записываются в памяти в переставленном виде. Перед использованием этих данных надо выполнить обратную перестановку. Другой возможностью является написание программы алгоритма в таком виде, когда на каждом шаге итерации вычисленные данные возвращаются в память в таком порядке, что адресация данных на следующем шаге итерации совпадает с адресацией на прошлом шаге. Имеется много способов таких перестановок; их детализация является существенной частью структуры программы или аппаратурного модуля. Конструктор должен решить, какая часть данных будет переиндексирована при запоминании, какая часть данных будет переиндексирована при вызове, и какая часть рабочей памяти может быть использована сверх того объема, который отведен для запоминания входного массива данных.

Большой БПФ-алгоритм Винограда не является столь регулярным, так что его реализация не может иметь такого аккуратного вида. Однако, если приступая к программной или аппаратурной реализации, тщательно проанализировать структуру алгоритма, то ее можно сделать достаточно упорядоченной. Проиллюстрируем некоторые возможности на примере построения 63-точечного БПФ-алгоритма Винограда из 7-точечного и 9-точечного БПФ-алгоритмов Винограда. Предположим, что в малых БПФ-алгоритмах Винограда матрицы профакторизованы в виде $\mathbf{W}_7 =$

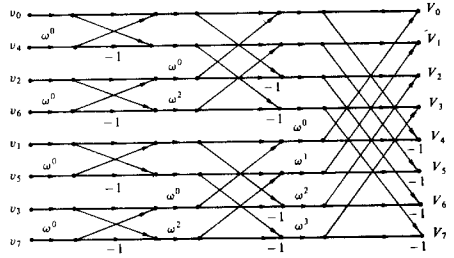


Рис. 9.8. БПФ-алгоритм Кули—Тьюки по основанию два с прореживанием по времени.

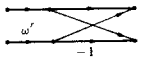


Рис. 9.9. Двуточечная бабочка.

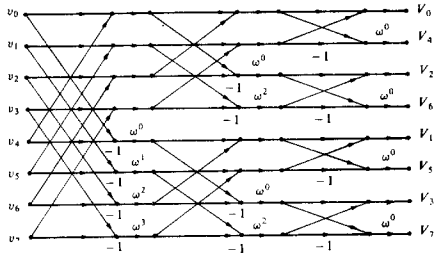


Рис. 9.10. БПФ-алгоритм Кули—Тьюки по основанию два с прореживанием по частоте.

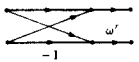


Рис. 9.11. Двуточечная бабочка.

$= \mathbf{C}_7\mathbf{B}_7\mathbf{A}_7$ и $\mathbf{W}_9 = \mathbf{C}_9\mathbf{B}_9\mathbf{A}_9$, где $\mathbf{B}_7$ и $\mathbf{B}_9$ представляют собой диагональные матрицы соответственно размерностей 9 и 11. Тогда большой БПФ-алгоритм Винограда записывается в виде

$$\mathbf{W}_{63} = (\mathbf{C}_7 \times \mathbf{C}_9)(\mathbf{B}_7 \times \mathbf{B}_9)(\mathbf{A}_7 \times \mathbf{A}_9) = (\mathbf{C}_7 \times \mathbf{C}_9)\mathbf{B}_{63}(\mathbf{A}_7 \times \mathbf{A}_9).$$

Каждый из крайних множителей мы будем использовать в виде кронекеровского произведения, не производя вычисления новой матрицы, равной этому кронекеровскому произведению. Член $\mathbf{B}_7 \times \mathbf{B}_9$ заменим, однако, на диагональную матрицу $\mathbf{B}_{63}$.

Реализация рассматриваемого БПФ-алгоритма начинается с перезаписи данных в виде двумерной (7×9)-таблицы. Первое умножение каждого из 7 столбцов таблицы на матрицу $\mathbf{A}_9$ удлиняет столбцы до 11. Следующее за этим умножение каждой из 11 строк на матрицу $\mathbf{A}_7$ приводит к их удлинению до 9. Теперь данные записаны в виде двумерной (9×11)-таблицы. Умножим поэлементно эту матрицу на 99 констант матрицы $\mathbf{B}_{63}$, которые заранее должны быть записаны в постоянном запоминающем устройстве. (Более точно, $\mathbf{B}_{63}$ представляет собой диагональную матрицу размерности 99, но для простоты ее здесь лучше рассматривать как (9×11)-таблицу, элементами которой служат упорядоченные соответствующим образом элементы диагонали матрицы $\mathbf{B}_{63}$.)

Полученную таблицу данных уменьшим теперь в объеме, умножая сначала каждый из 9 столбцов на $\mathbf{C}_9$, а затем каждую из 9 строк на матрицу $\mathbf{C}_7$. Это приведет к вычислению всех 63 компонент преобразования Фурье входного вектора, записаниых в виде (7×9)-таблицы. Остается только переупорядочить компоненты, одновременно записывая их в виде одномерного массива.

На рис. 9.12 иллюстрируется структура использования малых преобразований Фурье в 15-точечном БПФ-алгоритме Винограда. Из рисунка ясно, как простые логические цепи многократно используются для реализации указанных функций. Аналогичную структуру можно использовать и для других длин преобразований. В тех приложениях, в которых необходимо вычислять n-точечное преобразование Фурье непрерывного потока данных, можно воспользоваться конвейерной схемой. Показанная на рис. 9.13 конвейерная архитектура относится к высокоэффективному 1008-точечному БПФ-алгоритму. Такая структура может быть предложена для гипотетической радиолокационной задачи, в которой необходимо вычислять 1008-точечное преобразование Фурье бесконечной последовательности данных. Этот способ позволяет достигать скоростей вычисления порядка миллионов отсчетов в секунду. Входные данные распределяются по банкам 16-точечных модулей БПФ, пройдя через которые они поступают в банк 63-точечных модулей БПФ. Во время обработки данных в 63-точечных модулях в 16-точечных модулях начинается обработка следующего пакета данных. Этим достигается одновре-

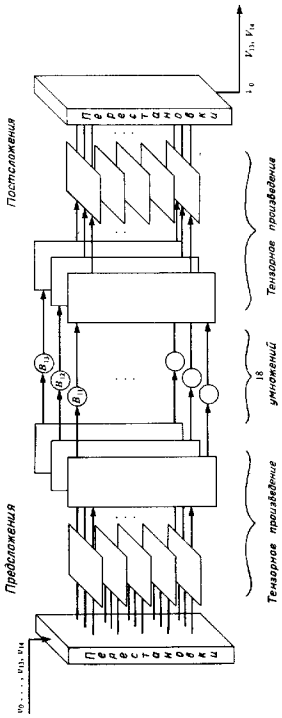


Рис. 9.12. Принципиальная схема большого БПФ-алгоритма Винограда.

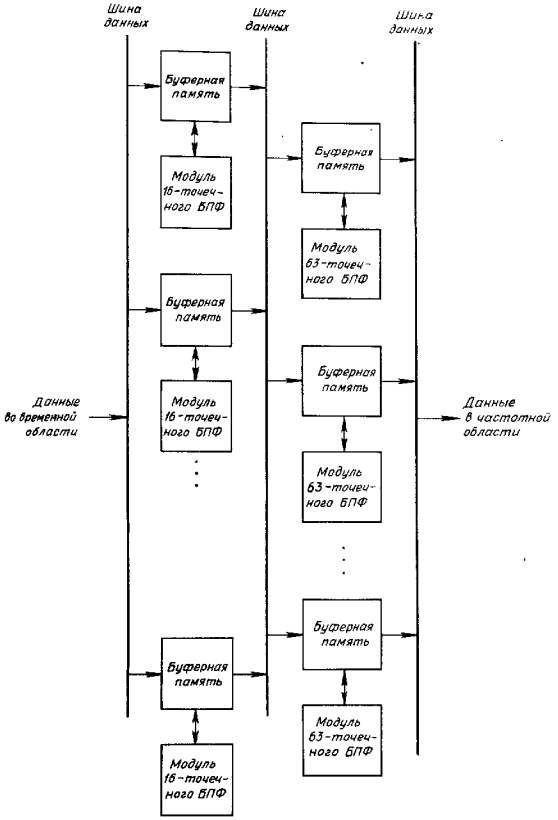


Рис. 9.13. Архитектура 1008-точечного БПФ.

менность вычисления обоих БПФ. В такую конвейерную архитектуру можно добавить параллельно работающие преобразователи, реализующие необходимые перестановки компонент данных. Тогда конвейер будет содержать четыре каскада — два для вычислений и два для перестановок компонент.

Может оказаться удобным, например, выбрать для первого банка девять 16-точечных модулей, а для второго — восемнадцать 63-точечных модулей. Тогда каждый 16-точечный модуль должен в пределах одного пакета выполнять 7 вычислений. Выбор соответствующих семи векторов с шины данных регулируется простыми методами управления памятью и циклическими выборками из памяти. Аналогично, передача данных с выхода 16-точечных модулей на центральную шину данных управляется некоторым фиксированным протоколом. Определенный протокол управляет также обработкой данных в 63-точечных модулях.

9.8. Преобразование Фурье ограниченного диапазона

Предположим, что при n временны́х компонентах преобразования Фурье интерес представляют меньше чем n частотных компонент. Скажем, имеется приложение, в котором обработке подлежат 10 000 временны́х компонент. Это позволяет вычислить 10 000 компонент спектра, из которых нужны не все, а, возможно, только 100. Воспользовавшись 10 000-точечным БПФ, можно вычислить все спектральные компоненты, а затем ненужные отбросить. Но лучше воспользоваться прореживающей фильтрацией и преобразованием Фурье ограниченного диапазона.

Преобразование Фурье ограниченного диапазона относится скорее всего к другой области цифровой обработки сигналов; оно не представляет собой быстрого алгоритма в том смысле, который вкладывается в это понятие в данной книге. Мы только хотим упомянуть его как некоторый другой метод, характер которого сильно отличается от изучаемых нами алгоритмов. В своем большинстве наши алгоритмы носят характер математических тождеств. Вопросы точности относятся к реализации алгоритмов, но не к их построению. Если ответ неточен, то это не вина алгоритма, как такового, а скорее связано с невозможностью записи чисел бесконечной длины при выполнении сложения и умножения.

В преобразовании Фурье ограниченного диапазона появляется новый феномен. Оно представляет собой не математическое тождество, а всего лишь приближенное вычисление, хотя и дает сколь угодно точное приближение. Общий вид преобразования Фурье ограниченного диапазона представлен на рис. 9.14.

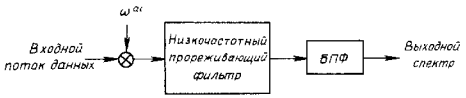


Рис. 9.14. Вычисление преобразования Фурье ограниченного диапазона.

Центральной теоремой в исследованиях прореживающих фильтров является теорема отсчетов. Скорость отсчетов временного дискретного сигнала не может быть меньше скорости Найквиста. Если скорость отсчетов сделать меньше, чем удвоенная максимальная частота ненулевой компоненты, то эти компоненты начинают «накладываться» на компоненты с меньшими частотами, создавая фальшивый «образ».

При вычислении преобразования Фурье ограниченного диапазона данные сначала пропускаются через низкочастотный прореживающий фильтр, который подавляет ненужные спектральные компоненты и сохраняет те спектральные компоненты, которые представляют интерес. Практически используемые низкочастотные фильтры, построенные методами дискретной техники, не являются совершенными низкочастотными фильтрами. Они немного изменяют нужные спектральные компоненты и не полностью подавляют нежелательные спектральные компоненты, так что преобразование Фурье ограниченного диапазона не является точным.

После выполнения прореживающей фильтрации для вычисления нужных спектральных компонент можно воспользоваться БПФ-алгоритмом малой длины. Если полоса нужных спектральных компонент начинается не с нуля, то надо просто ввести показанный на рис. 9.14 модулирующий множитель ω^{at}. Это обеспечивает такой частотный сдвиг представляющих интерес частотных компонент, что они начинаются с нуля.

Задачи

9.1. Доказать, что если n равно степени двух, то n-фильтр-секция, построенная итерацией 2-фильтр-секции, содержит $M(n) = n^{\log_2 3}$ умножений и число сложений, описываемое рекурсией

$$A(n) = 2n + 3A(n/2)$$

с начальным условием $A(1) = 0$.

9.2. Отталкиваясь от описанного в гл. 3 алгоритма двумерной 3-точечной линейной свертки, построить методом итерации двумерную 9-точечную линейную свертку.

9.3. Для линейной свертки можно предложить метод перекрытия, который по своему духу будет чем-то средним между методом перекрытия с накоплением и методом перекрытия с суммированием. Выписать уравнения для этого гибридного метода. Каковы его преимущества и каковы недостатки?

9.4. Доказать, что алгоритм, вычисляющий три последовательных отсчета на выходе симметрического фильтра с четырьмя отводами, должен содержать по меньшей мере пять умножений.

9.5. Допустим, что имеется устройство (в виде жесткого модуля или подпрограммы) для вычисления 315-точечной *циклической* свертки, и что необходимо пропустить вектор, содержащий 1000 отсчетов данных, через КИО-фильтр с 100 отводами. Описать, как нужно разбить данные и организовать вход и выход устройства свертки для того, чтобы произвести необходимые вычисления.

9.6. а. Построить содержащий семь умножений алгоритм 4-точечной симметричной фильтрации вектора с четырьмя компонентами.

б. Построить содержащий шесть умножений алгоритм 4-точечной кососимметрической фильтрации вектора с четырьмя компонентами.

9.7. Если последовательность данных d является комплексной, то автокорреляцию обычно определяют равенствами

$$r_l = \sum_{k=0}^{N-1} d_{l+k}^* d_k, \qquad l = 0, \ldots, N + L - 2.$$

Показать, как надо изменить блок-схему на рис. 9.6 для того, чтобы она стала пригодной для обработки комплексных данных.

9.8. Для вычисления сверток можно использовать метод перекрытия с накоплением и короткие БПФ-алгоритмы, добавляя к ним некоторые поправочные члены. Используя 48-точечный БПФ-алгоритм Винограда, применить этот метод для вычисления 25 точек на выходе КИО-фильтра с 25 отводами. Сколько сложений и умножений потребуется в таком алгоритме?

9.9. Выписать алгоритм 3-фильтр-секции для поля комплексных чисел. Сколько вещественных сложений и умножений содержит алгоритм?

Замечания

Методы перекрытия для вычисления данных линейных сверток разбиением их на отрезки представляются естественными и использовались, по-видимому, независимо во многих работах. Стокхэм первым (1966) отметил, что сочетание БПФ-алгоритма Кули — Тьюки с теоремой о свертке дает хороший способ вычисления циклических сверток. Агарвал и Баррас (1974) показали, как можно одномерную свертку преобразовать в многомерную свертку, используя схему переиндексации, которую мы назвали гнездовым или итеративным методом. Эта конструкция представляет собой одну из форм секционирования по методу перекрытия с накоплением, в котором секции упорядочены в виде матрицы. Та же самая идея, но основанная на методе перекрытия с суммированием, была предложена Дюбуа и Венецианопулосом (1978) для вычисления больших циклических сверток. Конструкции хороших алгоритмов коротких секций фильтров, включая симметрические и кососимметрические фильтры, были описаны Виноградом (1979, 1980). Конструкции для интерполирующих секций фильтров являются непосредственным приложением методов Винограда к этим частным задачам. Трансформационный принцип описан Хопкрофтом и Мюзинским (1973). Метод секционирования для вычисления автокорреляции принадлежит Рейдеру (1970). Имеется очень много публикаций, посвященных вопросам организации памяти для построения БПФ-алгоритмов. Одной из недавних работ является работа Барраса и Эшенбечера (1979).

Использование прореживания для вычисления преобразования Фурье было предложено Лю и Минцером [1978]. Они пошли дальше нашего краткого обзора

и рассмотрели многокаскадные конструкции прореживающих фильтров. Обзор работ по интерполяции и прореживанию можно найти в статье Крошьера и Ребинера (1981).

За исключением числа умножений и числа сложений, мы не проводили сравнений характеристик различных БПФ-алгоритмов, так как, даже если отбросить аппаратурную реализацию и рассматривать только программную реализацию алгоритма, эти характеристики зависят от точности выбранного варианта алгоритма, уровня языка программирования, архитектуры процессора и искусства программиста. Некоторые исследования характеристик были проведены Колбой и Парксом (1977), Силверменом (1977), Моррисом (1978), Навабом и Макклелланом (1979), Паттерсоном и Макклелланом (1978) и Пандой, Полом и Чаттерджи (1983). Было бы опрометчиво заявить, что из всех этих разнообразных работ можно сделать единое заключение.

Глава 10

БЫСТРЫЕ АЛГОРИТМЫ, ОСНОВАННЫЕ НА СТРАТЕГИИ ДУБЛИРОВАНИЯ

Некоторые хорошие алгоритмы удается строить на основе стратегии, которая дублирует алгоритм решения половины задачи. Выберем объем задачи n в качестве параметра и разобьем, если можно, задачу на две подзадачи объема $n/2$ той же самой структуры, что и исходная задача. Если решения этих подзадач можно скомбинировать в решение исходной задачи, то получается алгоритм решения данной задачи, который часто оказывается эффективным. БПФ-алгоритм Кули—Тьюки по основанию два можно понимать как алгоритм, полученный разбиением задачи на две половины и дублированием, так как n-точечный алгоритм строится из двух $(n/2)$-точечных БПФ-алгоритмов. Алгоритм итерирования фильтр-секций также имеет эту структуру, так как n-фильтр-секция строится из двух $(n/2)$-фильтр-секций. В настоящей главе строятся другие быстрые алгоритмы, основанные на делении задачи пополам и дублировании. Они иллюстрируют способ построения алгоритмов, пригодных для многих видов задач.

10.1. Стратегия деления пополам и дублирования

Рассмотрим задачу нахождения многочлена $p(x)$ степени n, заданного своими корнями $\beta_0, \beta_1, \ldots, \beta_{n-1}$. Мы должны найти коэффициенты многочлена $p(x)$ вида

$$p(x) = (x - \beta_0)(x - \beta_1) \ldots (x - \beta_{n-1}).$$

Наиболее естественным путем решения этой задачи является последовательное умножение на одночлены, начиная с какого-нибудь конца, скажем правого, согласно процедуре

$$p^{(i)}(x) = (x - \beta_i)\, p^{(i-1)}(x), \quad i = 1, \ldots, n-1,$$

при начальной точке $p^{(0)}(x) = (x - \beta_0)$. На i-м шаге итерации такая процедура требует i умножений и i сложений, что в общей сложности приводит к $(1/2)\, n\, (n-1)$ умножениям и такому же числу сложений.

Деление задачи пополам и дублирование позволяет построить более эффективный алгоритм. Предположим, что n равно степени двух, а именно $n = 2^m$. (Процедуру легко модифицировать для других значений n, пополняя данные фиктивными входами.) Положим теперь

$$p'(x) = \prod_{i=0}^{n/2-1} (x - \beta_i), \quad p''(x) = \prod_{i=0}^{n/2-1} (x - \beta_{n/2+i})$$

и

$$p(x) = p''(x)\, p'(x).$$

Последнее равенство в том виде, в котором оно записано, требует $(n/2)^2$ умножений. Если $p'(x)$ и $p''(x)$ вычисляются непосредственным образом, то каждое из этих вычислений требует $(1/2)\,(n/2)\,((n/2) - 1)$ умножений. Полное число умножений равно

$$\left(\frac{n}{2}\right)^2 + 2 \cdot \frac{1}{2}\left(\frac{n}{2}\right)\left(\frac{n}{2} - 1\right) = \frac{1}{2} n (n - 1),$$

что не лучше, чем при прямом методе вычислений.

Для извлечения из стратегии дублирования какой-то выгоды надо применить лучшие способы сочетания двух частей вычисления $p(x) = p''(x)\, p'(x)$. Но эта задача представляет собой линейную свертку, которую мы столь интенсивно изучали и которую можно вычислить не более чем с $An \log_2 n$ операциями, где A — некоторая малая константа. Следовательно, полное число умножений в таком алгоритме не превосходит величины

$$M(n) = An \log_2 n + \frac{n}{2}\left(\frac{n}{2} - 1\right),$$

которая при больших n уже лучше выписанной ранее. Это число можно уменьшать еще дальше, применяя к вычислению каждого из многочленов $p'(x)$ и $p''(x)$ ту же самую идею. Каждая из этих задач может быть разбита на две и вычислена по алгоритму, сочетающему содержащие $A\,(n/2)\log_2 (n/2)$ операций решения двух половин, что дает в итоге меньше чем $An \log_2 (n/2)$ операций. Продолжая таким образом, мы уменьшим число умножений до

$$M(n) = A \sum_{i=1}^{m} n \log_2 (n/2i) = A \frac{n}{2} (\log_2^2 n - \log_2 n).$$

При не очень малых n эта величина меньше, чем $(1/2)\, n\, (n - 1)$, так что стратегия разбиения задачи пополам и дублирования позволила построить улучшенный алгоритм.

На рис. 10.1 приведена схема организации вычисления произведения многочленов с помощью рассмотренной процедуры дублирования. Эта схема является хорошим примером так называемых *рекурсивных процедур*. Рекурсивная процедура представ-

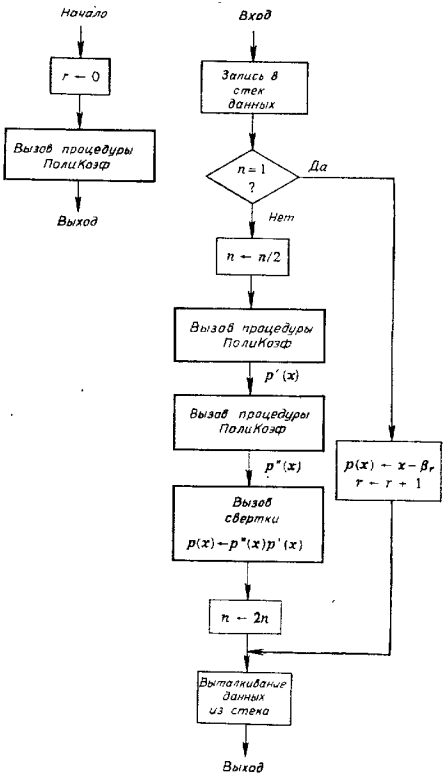


Рис. 10.1. Процедура ПолиКоэф.

Замечания: r — глобальная переменная.

Процедура ПолиКоэф вычисляет $\prod_{i=r}^{r+n-1} (x - \beta_i)$ и увеличивает значение r.

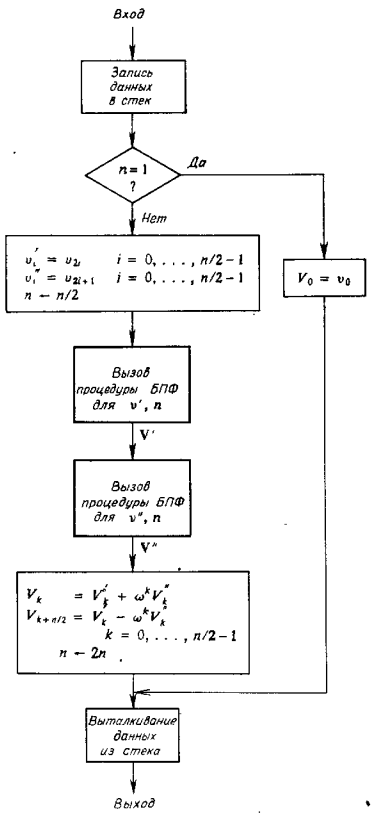


Рис. 10.2. Процедура БПФ.

ляет собой искусный программный модуль, который не содержит точного описания каждого уровня вычислений, а описывает только один уровень, содержащий копию той же самой процедуры. Процедура обращается с вызовом к самой себе. Это предполагает запись текущих данных в виде стека, который будет описан в следующем разделе. При каждом вызове процедуры происходит проталкивание вниз существующего стека данных с целью освобождения чистого рабочего пространства.

Подобная стратегия дублирования может использоваться для решения многих задач. Если задача некоторого вычисления зависит от некоторого целого числа n, то при $n = 2^m$ можно попытаться получить ответ из ответов для двух подзадач с $n = 2^{m-1}$. Рекурсивная форма БПФ-алгоритма Кули—Тьюки с дублированием выписана на рис. 10.2. Полезно проследить, насколько отличаются последовательности операций на рис. 9.8 и 10.2. Рекурсивный алгоритм приводит к расточительному использованию текущей памяти, так как создает временный стек для запоминания результатов вычисления преобразований Фурье разных объемов. С другой стороны, рекурсивную форму алгоритма удобно использовать в единой программе, позволяющей вычислять произвольное преобразование Фурье по основанию два. Рекурсивная форма может быть также удобна для организации БПФ-алгоритмов по смешанному основанию, так как легко разветвляется в другие подпрограммы.

Стратегию дублирования обычно удается распространить на задачи, в которых параметр n не равен степени двух. Один из способов состоит в добавлении достаточного для превращения числа n в ближайшую степень двух числа фальшивых итераций, хотя иногда, как, например, в случае преобразования Фурье, этот способ не проходит. Можно также разбивать задачу на подзадачи другого объема, равного, скажем, трети или пятой части объема исходной задачи, но, если задача допускает, разбиение на половинки, как правило, дает лучший результат.

10.2. Структуры данных

Входящий в некоторое вычисление набор данных перед началом обработки должен быть соответствующим образом упорядочен. Данные промежуточных вычислений также подлежат определенному запоминанию; в рассмотренных в предыдущем параграфе алгоритмах дублирования для этого использовался обратный стек. Двумя основными методами организации данных являются *списки* и *деревья*. *Списком* длины L называется упорядоченное множество из L позиций; каждая позиция сама может представлять собой сложный набор данных, даже, в частности, тоже содержать списки.

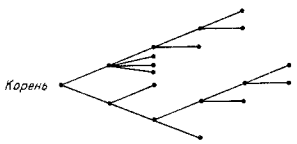


Рис. 10.3. Дерево.

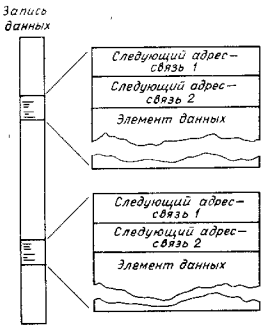


Рис. 10.4. Двойной связанный список.

Список, все элементы которого являются числами, часто называется *вектором*. Если все элементы списка принадлежат некоторому конечному алфавиту, список иногда называется *цепочкой*. Цепочка не обязательно имеет фиксированную длину, но длина вектора обычно фиксирована. Различие между вектором и цепочкой лежит скорее в применении, чем в существе дела.

Списком переменной длины называется список, длина которого не фиксирована, а со временем растет или уменьшается. Список переменной длины может расти или укорачиваться путем добавления новой позиции в любую точку списка или удалением позиции из любой точки списка, но во многих случаях позиции добавляются или удаляются только на двух концах списка. Список, добавление или удаление позиций в котором происходит только на одном конце, — скажем, в вершине — называется *стеком*, или *обратным списком*, или *последним-пришел-первым-ушел-буфером*. Список, в котором добавление позиций происходит на одном конце, а удаление на другом, называется *очередью*, или *первым-пришел-первым-ушел-буфером*.

Деревом называется такая структура данных, в которой за каждой позицией могут следовать одна или более позиций, называемых *потомками*; пример графического изображения такой структуры приведен на рис. 10.3. Каждому узлу соответствует позиция данных, и каждая позиция имеет в качестве своих потомков другие позиции, с которыми она связана. Возможна ситуация, в которой несколько узлов дерева соответствуют одной и той же позиции данных. В противоположность дереву каждая позиция списка может иметь только одного потомка.

Позиция данных в списке или в дереве может быть достаточно сложной, включая, возможно, текст на естественном языке; но

с точки зрения исследования структуры списков каждая позиция рассматривается как некая единица. В памяти вычислительного устройства списки и деревья запоминаются с помощью некоторых имен, присваиваемых позициям; хорошими именами являются адреса ячеек памяти, в которых начинаются данные позиций. Список (или дерево) запоминается в виде последовательности имен позиций, входящих в список. Список не обязательно запоминать в непосредственной близости к позициям данных.

Другим, иногда более удобным методом, является метод непрямой адресации, называемый *связанным списком*; он показан на рис. 10.4. Позиции данных появляются в произвольном порядке, и каждый вход начинается с адреса следующей позиции данных списка. На рис. 10.4 показан *двойной связанный список*. В двойном связанном списке данные записаны двумя способами — скажем, в алфавитном и хронологическом порядке — но не обязательно запоминаются дважды. Если список часто пересматривается, то удобнее пользоваться связанным списком, так как это позволяет не двигать данные.

Организация стека даже из сложных позиций данных реализуется простым присваиванием адреса первой позиции данных и привязыванием к каждой позиции данных адреса следующей по списку позиции данных. Для записи новой позиции данных в вершину стека надо просто привязать адрес предыдущей вершины к новой вершине и изменить адрес первой позиции данных на адрес новой позиции. Для извлечения позиции данных из вершины стека надо обратить процедуру.

10.3. Быстрые алгоритмы сортировки

Задача сортировки формулируется следующим образом. Задана произвольная последовательность из n элементов, выбранных из множества, на котором имеется некоторый естественный порядок. Необходимо переупорядочить заданную последовательность в том же естественном порядке. Алгоритм сортировки может относиться непосредственно к позициям данных и перезаписи их в памяти, а может и не затрагивать позиции данных, а оперировать только с некоторыми привязанными к ним параметрами. Данные сортируются переупорядочиванием списка непрямых адресов. Сортировка переупорядочиванием адресов полезна в тех случаях, когда объем данных в каждой позиции сам по себе очень велик.

Предположим, что с каждой позицией данных связан некоторый целочисленный параметр, и необходимо упорядочить позиции данных так, чтобы целочисленный параметр менялся от своего наибольшего значения к наименьшему. Любая задача сортировки может быть сформулирована в таком виде.

Наивная процедура сортировки состоит в просмотре одной позиции за другой и выборе той, которой соответствует наибольшее значение параметра. Затем так же просматриваются оставшиеся позиции и находится та из них, которой соответствует наибольшее значение параметра. Это продолжается до тех пор, пока не будут рассортированы все позиции. В повседневной жизни, когда n мало, это вполне удовлетворительная процедура. Однако, среднее число шагов в ней пропорционально n^2. Для сортировки больших списков можно действовать намного лучше.

Хорошие алгоритмы сортировки основываются на делении списка пополам и дублировании. Одним из них является алгоритм *сортировки слиянием*; его сложность пропорциональна величине $n \log_2 n$. Разобьем список из n позиций данных на две половины и упорядочим каждую из них. Из этих упорядоченных списков половин организуем общий упорядоченный список, сливая их вместе по следующему правилу. Сравним верхние элементы в обоих списках. Выберем из них тот, который соответствует бо́льшему значению параметра, и разместим его в качестве следующей позиции объединенного списка, выбросив из той половины, где он находился раньше. Так как каждая позиция попадает в объединенный список после операции сравнения, то в общей сложности придется сделать n сравнений. Следовательно, сложность $C(n)$ алгоритма сортировки слиянием описывается рекурсией

$$C(n) \leqslant 2C\left(\frac{n}{2}\right) + n.$$

Эта рекурсия дает оценку сложности для алгоритма сортировки слиянием:

$$C(n) \leqslant n \log_2 n.$$

Так как имеется только одно значение аргумента, при котором здесь выполняется равенство, то при слиянии некоторые сравнения не нужны, так что алгоритм можно немного улучшить. Тем не менее, мы полагаем, что эта оценка является достаточно точной.

Имеются другие способы разбиения задачи сортировки. В алгоритме «*быстрой сортировки*» случайно выбирается один из параметров данных в качестве основы для разбиения списка. Характеристики такого алгоритма являются случайными величинами, достаточно хорошими в среднем, но очень медленными в наихудших случаях.

Так как в алгоритме «быстрой сортировки» параметр разбиения списка выбирается непосредственно из данных, то этот параметр должен быть случаен. В противном случае можно было бы построить тестовый список, для которого «быстрый алгоритм» оказался бы очень плохим. Например, если в качестве параметра разбиения списка всегда выбирать первый вход данных, то «быстрый алгоритм» будет сортировать список чрезвычайно медленно.

«Быстрый алгоритм» работает следующим образом. Выберем случайно из данных некоторый элемент с параметром a. Все данные, за исключением a, разделим на два подмножества: те данные, параметр которых меньше a, и те данные, параметр которых не меньше a. Упорядочим каждое из этих подмножеств и соединим их в один список, расположив позицию a между ними. Каждое из двух подмножеств упорядочивается по такому же правилу.

Два списка, на которые разбивается исходный список, содержат по $n-1-i$ и i элементов, где i — случайная величина, равновероятно распределенная на множестве $\{0, \ldots, n-1\}$. Сложность формирования этих множеств пропорциональна n. Средняя сложность сортировки исходного списка из n точек равна

$$C(n) = An + \frac{1}{n}\sum_{i=0}^{n-1} C(i) + \frac{1}{n}\sum_{i=0}^{n-1} C(n-1-i),$$

где A — некоторая константа. Для $n > 2$ это приводит к следующей рекурсивной формуле для сложности $C(n)$ алгоритма «быстрой сортировки»:

$$C(n) = An + \frac{2}{n}\sum_{i=0}^{n-1} C(i)$$

при начальных условиях $C(0) = C(1) = 0$. (Иногда предпочитают полагать $C(0)$ и $C(1)$ некоторыми малыми константами, но это мало что меняет и никак не влияет на асимптотику сложности.) Покажем, что при $n > 2$ величина $C(n)$ меньше, чем $2An \log n$. Доказательство проведем индукцией: предположим, что для всех $i < n$ величина $C(i)$ меньше, чем $2Ai \ln_e i$. Это справедливо для $n = 2$. Тогда

$$C(n) < An + \frac{4A}{n}\sum_{i=2}^{n-1} i \ln_e i.$$

Так как функция $i \ln_e i$ выпукла, то правую часть можно ограничить величиной вида

$$C(n) < An + \frac{4A}{n}\int_2^n x \ln_e x \, dx.$$

Следовательно,

$$C(n) < An + \frac{4A}{n}\left[\frac{n^2 \ln_2 n}{2} - \frac{n^2}{4}\right] = 2An \ln_e n.$$

Средняя характеристика алгоритма «быстрой сортировки» асимптотически лучше средней характеристики алгоритма сортировки слиянием, но в наихудшем случае характеристика становится намного хуже.

10.4. Рекурсивное быстрое преобразование Фурье по основанию два

Комплексное преобразование Фурье по основанию два определяется формулой

$$V_k = \sum_{l=0}^{n-1} \omega^{lk} v_l, \qquad k = 0, \ \ldots, \ n-1,$$

где $n = 2^m$, m — целое число и $\mathbf{v}$ — вектор с комплексными компонентами. В гл. 4 мы интересовались построением алгоритмов, представляющих собой малый пакет уравнений, непосредственное применение которых позволяет вычислить преобразование Фурье радиуса два. Этот способ очень подходит для длин n, равных 8 или 16, но может оказаться неподходящим при больших n. В настоящем параграфе мы выпишем алгоритм по основанию два в рекурсивной форме. Входные данные в комплексном виде мы выбираем потому, что при $n = 32$ и более эта форма алгоритма применительно к комплексным данным эффективнее, чем двойное использование БПФ-алгоритма для вещественного входа.

Так же, как и в гл. 4, воспользуемся методом Винограда для разбиения вычисления преобразования Фурье на отдельное вычисление компонент с четными индексами и компонент с нечетными индексами. Для обработки компонент с четными индексами заменим k на $2k$ и запишем

$$V_{2k} = \sum_{l=0}^{n/2-1} (v_l + v_{l+n/2})\, \omega^{2lk}, \quad k = 0, \ \ldots, \ n/2-1.$$

Мы получили $n/2$-точечное преобразование Фурье нового вектора, компоненты которого равны $v_l + v_{l+n/2}$, так что для их вычисления требуется $n/2$ комплексных сложений. Для обработки компонент с нечетными индексами заменим k на $2k+1$ и запишем

$$V_{2k+1} = \sum_{l=0}^{n/2-1} v_{2l} \omega^{2l\,(2k+1)} + \sum_{l=0}^{n/2-1} v_{2l+1} \omega^{(2l+1)\,(2k+1)}, \ k = 0, \ \ldots, \ n/2-1.$$

Эти две суммы будут рассматриваться отдельно. Для их сочетания потребуется еще $n/2$ комплексных сложений.

В первой сумме выписанного уравнения необходимо вычислить только первые компоненты для $k = 0, \ \ldots, \ n/4-1$, так как в дальнейшем они повторяются. Эти компоненты представляют собой $(n/4)$-точечное преобразование Фурье вектора с компонентами $v_{2l}\omega^{2l}$; для вычисления последнего вектора требуется $(3/4)\,n - 8$ вещественных умножений (так как это вычисление содержит $n/4 - 4$ комплексных умножений, каждое из которых может быть выполнено с тремя вещественными умножениями, и два комплексных умножения, для вычисления каждого из которых нужно только два вещественных умножения) и столько же сложений.

Основная часть рассуждений связана с вычислением промежуточного результата,

$$t_{2k+1} = \sum_{l=0}^{n/2-1} v_{2l+1} \omega^{(2l+1)\,(2k+1)}, \quad k = 0, \ \ldots, \ n/2-1.$$

Так как порядок элемента ω равен n, то вычисления в показателе степени должны проводиться по модулю n, где $n = 2^m$. Все эти вычисления представляют собой умножения нечетных чисел по модулю 2^m. Согласно теореме 5.1.8, относительно операции умножения по модулю 2^m множество нечетных чисел образует группу, изоморфную группе $Z_2 \times Z_2^{m-2}$ и, следовательно, имеет две образующие. В качестве таких образующих можно выбрать 3 и -1; тогда множество показателей степени образует мультипликативную группу $\{3^l (-1)^{l'}\colon l = 0, \ \ldots, \ 2^{m-2}-1;\ l' = 0, 1\}$, групповой операцией в которой является умножение по модулю 2^m. На входных и выходных индексах это задает соответственно подстановки

$$2i+1 = 3^l(-1)^{l'} \ \text{ и } \ 2k+1 = 3^{-r}(-1)^{-r'}.$$

При таком представлении индексов предыдущее уравнение приводится к виду

$$t_{r,\,r'} = \sum_{l'=0}^{1} \sum_{l=0}^{2^{m-2}-1} v_{l,\,l'} \omega^{3^{l-r}(-1)^{l'-r'}}, \qquad \begin{matrix} r = 0, \ \ldots, \ 2^{m-2}-1, \\ r' = 0, \ 1, \end{matrix}$$

где элементы $t_{r,\,r'}$ и $v_{l,\,l'}$ двумерных таблиц равны тем компонентам t_{2k+1} и v_{2i+1}, индексы которых определяются выписанными выше подстановками. Рассматриваемое уравнение преобразуется при этом в двумерную циклическую свертку:

$$t(x,\ y) = g(x,\ y)\, v(x,\ y) \pmod{x^{2^{m-2}}-1} \pmod{y^2-1},$$

где $g(x, y)$ равен обобщенному многочлену Рейдера,

$$g(x,\ y) = \sum_{l'=0}^{1} \sum_{l=0}^{2^{m-2}-1} \omega^{3^l(-1)^{l'}} x^l y^{l'},$$

а $v(x, y)$ и $t(x, y)$ представляют собой многочлены от двух переменных, задаваемые входными и выходными данными соответственно равенствами

$$v(x, y) = \sum_{l'=0}^{1} \sum_{l=0}^{2^{m-2}-1} v_{l, l'} x^l y^{l'},$$

$$t(x, y) = \sum_{l'=0}^{1} \sum_{l=0}^{2^{m-2}-1} t_{l, l'} x^l y^{l'}.$$

Чтобы задать многочлен $v(x, y)$ по компонентам v_{2i+1}, нужно выполнить только перестановки. Аналогично, для вычисления компонент t_{2k+1} по многочлену $t(x, y)$ нужны только перестановки.

Структуру легче увидеть, если переписать многочлены в виде

$$v(x, y) = v_0(x) + y v_1(x), \qquad t(x, y) = t_0(x) + y t_1(x)$$

и

$$g(x, y) = \sum_{l=0}^{2^{m-2}-1} \omega^{3l} x^l + \sum_{l=0}^{2^{m-2}-1} \omega^{-3l} x^l y = g(x) + y g^*(x),$$

где $g(x) = \sum_{l=0}^{2^{m-2}-1} \omega^{3l} x^l$.

Тогда

$$\begin{bmatrix} t_0(x) \\ t_1(x) \end{bmatrix} = \begin{bmatrix} g(x) & g^*(x) \\ g^*(x) & g(x) \end{bmatrix} \begin{bmatrix} v_0(x) \\ v_1(x) \end{bmatrix}.$$

Если входные данные вещественны, то $t_1(x) = t_0^*(x)$ и вычислять надо только

$$t_0(x) = g(x) v_0(x) + g^*(x) v_1(x) \pmod{x^{n/4} - 1}.$$

На следующем шаге воспользуемся разложением

$$x^{n/4} - 1 = (x^{n/8} - 1)(x^{n/8} + 1)$$

и китайской теоремой об остатках. Пусть

$$g^{(0)}(x) = g(x) \pmod{x^{n/8} - 1},$$
$$g^{(1)}(x) = g(x) \pmod{x^{n/8} + 1},$$

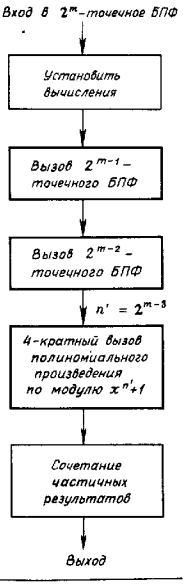
и остальные необходимые многочлены определены аналогичным образом. Так как согласно теореме 4.6.3 $g^{(0)}(x) = 0$, то вычислять надо только члены, содержащие $g^{(1)}(x)$:

$$\begin{bmatrix} t_0^{(1)}(x) \\ t_1^{(1)}(x) \end{bmatrix} = \begin{bmatrix} g^{(1)}(x) & g^{(1)*}(x) \\ g^{(1)*}(x) & g^{(1)}(x) \end{bmatrix} \begin{bmatrix} v_0^{(1)}(x) \\ v_1^{(1)}(x) \end{bmatrix}.$$

Далее воспользуемся алгоритмом 2-точечной циклической свертки

$$\begin{bmatrix} t_0^{(1)}(x) \\ t_1^{(1)}(x) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{2}\left[g^{(1)}(x) + g^{(1)*}(x)\right] & \\ & \frac{1}{2}\left[g^{(1)}(x) - g^{(1)*}(x)\right] \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} v_0^{(1)}(x) \\ v_1^{(1)}(x) \end{bmatrix}.$$

Рис. 10.5. Рекурсивная форма БПФ-алгоритма Винограда по основанию два.



Многочлен $\frac{1}{2}[g^{(1)}(x) + g^{(1)*}(x)]$ содержит только вещественные коэффициенты, а многочлен $\frac{1}{2}[g^{(1)}(x) - g^{(1)*}(x)]$ содержит только чисто мнимые коэффициенты. Следовательно, задача свелась к вычислению четырех произведений по модулю $x^{n/8} + 1$ многочленов с вещественными коэффициентами. Теоретически, каждая из этих задач требует $n/4 - 1$ вещественных умножений; практически известные алгоритмы требуют несколько большего числа умножений.

На рис. 10.5 приведена блок-схема описанного рекурсивного вычисления; на рис. 10.6 затабулированы получаемые при этом характеристики.

Рис. 10.6. Характеристики некоторых БПФ-алгоритмов Винограда по основанию два.

Длина	Вещественные входные данные			Комплексные входные данные		
	Число вещественных умножений		Число вещественных сложений	Число вещественных умножений		Число вещественных сложений
	нетривиальных	полное		нетривиальных	полное	
2	0	2	2	0	4	4
4	0	4	8	0	8	16
8	2	8	26	4	16	52
16	10	18	74	20	36	148
32	36	50		68	96	
64	102	124		188	232	
128	258	294		468	540	
256	608	666		1092	1208	
512	1370	1464		2444	2632	
1024	2994	3146		5316	5620	

10.5. Быстрая транспозиция

Во время обработки больших двумерных таблиц, подобных оцифрованным изображениям, процессор может воспринять только малую часть этой таблицы. Например, для запоминания (1024×1024)-таблицы требуется более, чем миллион ячеек памяти и более, чем два миллиона, если таблица комплексная. В большинстве случаев таблица запоминается во внешней памяти системы и считывается по частям в оперативную память процессора. Как правило, $(n \times n)$-таблица запоминается по столбцам (или по строкам) во внешней памяти, и в каждый момент времени между оперативной памятью и внешней памятью происходит обмен одним столбцом. Обмен двух элементов из двух различных столбцов столь же трудоемок, сколь обмен двух целых столбцов. Таким образом, для формирования одной строки матрицы приходится считывать n столбцов, а для формирования всех строк приходится считывать n^2 столбцов, так что прямое транспонирование матрицы приводит к считыванию n^2 столбцов.

В тех приложениях, в которых оперативная память мала, для перестановки строк и столбцов во внешней памяти можно пользоваться алгоритмом быстрой транспозиции. Мы рассмотрим случай, когда оперативная память допускает запоминание двух столбцов из квадратной таблицы. Это наиболее интересный и позволяющий продемонстрировать все основные идеи случай. Для транспонирования $(n \times n)$-таблицы во внешней памяти алгоритм делает $n \log_2 n$ считываний столбцов в оперативную память. Если в оперативную память нельзя записать двух столбцов, то алгоритм приходится дублировать, что снижает его эффективность; если оперативная память допускает запись более двух столбцов, то алгоритм можно улучшить на некоторую константу.

Транспонирование матрицы, записанной в оперативной памяти прямого доступа, является тривиальной задачей, так как реализуется простым изменением адресов при вызове данных. Мы будем полагать, что транспонирование малых матриц сводится к считыванию таблицы в оперативную память и последующему считыванию ее во внешнюю память. Пусть $(2^m \times 2^m)$-матрица $\mathbf{A}$ разбита на блоки размерности 2^{m-1} в виде

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}.$$

Предположим, что у нас имеется способ вычисления матрицы

$$\begin{bmatrix} \mathbf{A}_{11}^T & \mathbf{A}_{12}^T \\ \mathbf{A}_{21}^T & \mathbf{A}_{22}^T \end{bmatrix}.$$

Тогда для вычисления матрицы

$$\mathbf{A}^T = \begin{bmatrix} \mathbf{A}_{11}^T & \mathbf{A}_{21}^T \\ \mathbf{A}_{12}^T & \mathbf{A}_{22}^T \end{bmatrix}$$

достаточно переставить матрицы $\mathbf{A}_{12}^T$ и $\mathbf{A}_{21}^T$. Это можно сделать, считывая и одновременно переставляя один столбец из $\mathbf{A}_{12}^T$ с одним столбцом из $\mathbf{A}_{21}^T$. Для полной перестановки матриц потребуется переставить $n/2$ пар столбцов (или n столбцов).

Теперь применим эту идею рекурсивно к вычислению $\mathbf{A}_{11}^T$, $\mathbf{A}_{21}^T$, $\mathbf{A}_{12}^T$ и $\mathbf{A}_{22}^T$, разбивая каждую из них на подблоки. Так как $\mathbf{A}_{11}^T$ и $\mathbf{A}_{21}^T$ расположены в одних и тех же столбцах таблицы, то входящие в них транспозиции выполняются перестановкой одних и тех же столбцов. Таким образом, каждый уровень рекурсии затрагивает n столбцов, а всего уровней имеется $\log_2 n$. Следовательно, в противоположность прямому методу транспонирования матрицы, содержащему n^2 перестановок столбцов, построенный быстрый алгоритм транспозиции содержит $n \log_2 n$ перестановок столбцов.

10.6. Умножение матриц

Произведение матриц размерности два

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

можно вычислить по правилам

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21}, \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22}, \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21}, \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22}, \end{aligned}$$

из которых следует, что так организованное вычисление содержит восемь умножений и четыре сложения. *Алгоритм Штрассена* позволяет так организовать вычисление, что оно будет содержать семь умножений.

В алгоритме Штрассена сначала выполняются следующие вычисления:

$$\begin{aligned} m_1 &= (a_{12} - a_{22})(b_{21} + b_{22}), \\ m_2 &= (a_{11} + a_{22})(b_{11} + b_{22}), \\ m_3 &= (a_{11} - a_{21})(b_{11} + b_{12}), \\ m_4 &= (a_{11} + a_{12})\, b_{22}, \end{aligned}$$

$$m_5 = a_{11}(b_{12} - b_{22}),$$
$$m_6 = a_{22}(b_{21} - b_{11}),$$
$$m_7 = (a_{21} + a_{22})\, b_{11}.$$

Затем элементы произведения матриц вычисляются по формулам

$$c_{11} = m_1 + m_2 - m_4 + m_6,$$
$$c_{12} = m_4 + m_5,$$
$$c_{21} = m_6 + m_7,$$
$$c_{22} = m_2 - m_3 + m_5 - m_7.$$

В матричном виде алгоритм Штрассена записывается равенством

$$\begin{bmatrix} c_{11} \\ c_{12} \\ c_{21} \\ c_{22} \end{bmatrix} = \left[\begin{array}{cccccccc} 1 & 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 & 0 & -1 & 0 \end{array}\right] \left[\begin{array}{cccccccc} G_1 & & & & & & & \\ & G_2 & & & & & & \\ & & G_3 & & & & & \\ & & & G_4 & & & & \\ & & & & G_5 & & & \\ & & & & & G_6 & & \\ & & & & & & G_7 & \end{array}\right] \left[\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{bmatrix},$$

в котором элементы стоящей в центре диагональной матрицы вычисляются по правилу

$$\begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \\ G_6 \\ G_7 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{bmatrix}.$$

Алгоритм Штрассена содержит семь умножений и 18 сложений. Если одна из двух перемножаемых матриц является постоянной и используется много раз, то некоторые сложения можно вычислить заранее вне процесса умножения и тогда число сложений становится равным 13.

По сравнению со стандартным алгоритмом умножения матриц алгоритм Штрассена даже в лучшем случае меняет одно умножение на девять сложений. Практически этот алгоритм умножения матриц размерности два не дает преимуществ.

Теперь рассмотрим задачу умножения двух матриц размерности n. Прямой метод такого вычисления требует n^3 умножений и $(n-1)\,n^2$ сложений. Предположим, что n равно степени двух: $n = 2^m$ для некоторого целого m; в противном случае дополним матрицу справа таким количеством нулевых столбцов и снизу таким количеством нулевых строк, чтобы n стало равным степени двух.

Произведение матриц $\mathbf{C} = \mathbf{AB}$ можно разбить на блоки

$$\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix},$$

где каждый блок представляет собой матрицу размерности 2^{m-1}. Вычисление блоков слева прямым умножением блоков справа содержит восемь умножений матриц размерности $n/2$ и четыре сложения матриц размерности $n/2$. Если все эти вычисления проводить стандартными способами, то полное число умножений будет равно

$$M(n) = 8\left(\frac{n}{2}\right)^3 = n^3,$$

$$A(n) = 8\left(\frac{n}{2} - 1\right)\left(\frac{n}{2}\right)^2 + 4\left(\frac{n}{2}\right)^2 = (n-1)\,n^2.$$

Мы получили те же самые величины, что и раньше, так что стратегия дублирования не приводит к увеличению эффективности, если не воспользоваться некоторыми другими возможностями. Такие возможности дает алгоритм Штрассена.

Алгоритм Штрассена можно применить на уровне блоковых матриц, так как он не зависит от свойства коммутативности умножения. Итак, применим сначала алгоритм Штрассена в следующем виде:

$$\mathbf{M}_1 = (\mathbf{A}_{12} - \mathbf{A}_{22})(\mathbf{B}_{21} + \mathbf{B}_{22}),$$
$$\mathbf{M}_2 = (\mathbf{A}_{11} + \mathbf{A}_{22})(\mathbf{B}_{11} + \mathbf{B}_{22}),$$
$$\mathbf{M}_3 = (\mathbf{A}_{11} - \mathbf{A}_{21})(\mathbf{B}_{11} + \mathbf{B}_{12}),$$
$$\mathbf{M}_4 = (\mathbf{A}_{11} + \mathbf{A}_{12})\mathbf{B}_{22},$$
$$\mathbf{M}_5 = \mathbf{A}_{11}(\mathbf{B}_{12} - \mathbf{B}_{22}),$$
$$\mathbf{M}_6 = \mathbf{A}_{22}(\mathbf{B}_{21} - \mathbf{B}_{11}),$$
$$\mathbf{M}_7 = (\mathbf{A}_{21} + \mathbf{A}_{22})\mathbf{B}_{11}.$$

Блоки матрицы $\mathbf{C}$ вычисляются по правилам

$$\mathbf{C}_{11} = \mathbf{M}_1 + \mathbf{M}_2 - \mathbf{M}_4 + \mathbf{M}_6,$$
$$\mathbf{C}_{12} = \mathbf{M}_4 + \mathbf{M}_5,$$
$$\mathbf{C}_{21} = \mathbf{M}_6 + \mathbf{M}_7,$$
$$\mathbf{C}_{22} = \mathbf{M}_2 - \mathbf{M}_3 + \mathbf{M}_5 - \mathbf{M}_7.$$

Если матрица $\mathbf{A}$ является матрицей констант, то вместо имевшихся ранее восьми умножений и четырех сложений матриц размерности $n/2$, мы получаем семь умножений и тринадцать сложений матриц размерности $n/2$. Если каждое из этих вычислений проводить стандартным способом, то для полного числа умножений получаем

$$M(n) = 7\left(\frac{n}{2}\right)^3 = \frac{7}{8}\,n^3,$$

что меньше n^3. Для полного числа сложений имеем

$$A(n) = 7\left(\frac{n}{2} - 1\right)\left(\frac{n}{2}\right)^2 + 13\left(\frac{n}{2}\right)^2 = \left(\frac{7}{8}n + \frac{3}{2}\right)n^2,$$

что при $n > 20$ также меньше, чем $(n - 1)\,n^2$.

Еще большее улучшение дает рекурсивное использование алгоритма Штрассена, основанное на разбиении блоков на более мелкие подблоки и использовании тех же самых уравнений. В этом случае число умножений равно

$$M(n) = 7^m = 7^{\log_2 n} = n^{\log_2 7} = n^{2.81}.$$

Число сложений вычисляется сложнее. Оно удовлетворяет рекурсии

$$A(n) = 7A\left(\frac{n}{2}\right) + 13\left(\frac{n}{2}\right)^2.$$

Число сложений больше числа умножений, но также растет как $n^{2\,81}$. При достаточно больших n это число меньше числа сложений, необходимых в прямом методе вычисления. При $n = 1024$ число необходимых в алгоритме сложений примерно равно числу сложений прямого метода вычислений, но число умножений равно примерно одной четверти от числа умножений прямого метода вычисления произведения матриц размерности 1024.

10.7. Рекурсивный алгоритм Евклида

Стратегия дублирования позволяет ускорить алгоритм Евклида и построить алгоритм, сложность которого имеет порядок $n \log^2 n$. Основные вычисления в алгоритме Евклида описываются уравнениями

$$Q^{(r)}(x) = \begin{bmatrix} s^{(r-1)}(x) \\ t^{(r-1)}(x) \end{bmatrix}$$

$$\begin{bmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} \begin{bmatrix} s^{(r-1)}(x) \\ t^{(r-1)}(x) \end{bmatrix} = \mathbf{A}^{(r)}(x) \begin{bmatrix} s(x) \\ t(x) \end{bmatrix},$$

где

$$\mathbf{A}^{(r)}(x) = \prod_{l=r}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{bmatrix},$$

и алгоритм останавливается, когда многочлен $t^{(r+1)}(x)$ обращается в нуль.

Ясно, что вычисление матрицы $\mathbf{A}^{(r)}(x)$ имеет ту же самую структуру, что и показанное на рис. 10.1 вычисление произведения многочленов, которое оказалось столь пригодным для стратегии дублирования. Поэтому можно ожидать, что стратегия деления задачи пополам и дублирования позволит улучшить характеристики алгоритма Евклида. Нам надо набор итераций разделить на две группы. Если это деление удастся осуществить так, что каждая из групп будет похожа (или может быть сделана похожей) на исходную задачу, то мы получим рекурсивную структуру.

Имеется, однако, несколько деталей, о которых следует позаботиться. Первая трудность состоит в том, что мы не знаем заранее число итераций, так что невозможно сказать, когда половина итераций завершена. Эту трудность можно исключить, прерывая вычисления в точке, в которой выполнена примерно половина итераций.

Другая трудность состоит в том, что $Q^{(r)}(x)$ зависит от $\mathbf{A}^{(r-1)}(x)$, и, в свою очередь, $\mathbf{A}^{(r)}(x)$ зависит от $Q^{(r)}(x)$. Следовательно, заранее не известны все участвующие в вычислении $\mathbf{A}^{(r)}(x)$ его делители. При разбиении на группы необходимо позаботиться о том, чтобы ни один делитель не использовался раньше, чем он станет известным. Теорема 10.7.1 утверждает, что если разбиение алгоритма происходит в правильной точке, то структура обеих групп аналогична структуре исходной задачи.

Пусть матрица $\mathbf{A}^{(r+1)}(x)$ факторизуется в виде

$$\mathbf{A}^{(r)}(x) = \mathbf{A}^{(r,\,r'+1)}(x)\,\mathbf{A}^{(r')}(x),$$

где

$$\mathbf{A}^{(r,\,r'+1)}(x) = \prod_{l=r}^{r'+1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{bmatrix}$$

и

$$\mathbf{A}^{(r')}(x) = \prod_{l=r'}^{1} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{bmatrix}$$

для некоторого фиксированного r'. Первая группа уравнений состоит из r' итераций, совпадающих с первыми r' итерациями исходной задачи. Вторая группа содержит уравнения для $r = r' + 1, \ldots, R$:

$$Q^{(r)}(x) = \begin{bmatrix} s^{(r-1)}(x) \\ t^{(r-1)}(x) \end{bmatrix}, \quad \begin{bmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \mathbf{A}^{(r,r'+1)}(x) \begin{bmatrix} s^{(r')} \\ t^{(r')} \end{bmatrix}.$$

Эта группа вычислений, очевидно, имеет ту же форму, что и исходная задача.

Первая группа также имеет эту же структуру, но если для ее вычисления использовать тот же путь решения, то стратегия деления задачи не даст ожидаемого улучшения. Чтобы уменьшить сложность, надо упростить вычисление первой группы. Такое упрощение

является источником значительной экономии числа вычислений и основывается на следующей теореме, описывающей условия, при которых возможно усечение делимого и делителя, не изменяющее частотного и остатка.

Теорема 10.7.1. *Пусть два заданных многочлена $f(x)$ и $g(x)$, степени которых связаны соотношением* $\deg g(x) < \deg f(x)$, *записаны в виде* $f(x) = f'(x)\,x^k + f''(x)$ *и* $g(x) = g'(x)\,x^k + g''(x)$, *где степени многочленов $f''(x)$ и $g''(x)$ меньше некоторого числа k, удовлетворяющего неравенству* $k \leqslant 2 \deg g(x) - \deg f(x)$. *Пусть алгоритм деления приводит соответственно к равенствам* $f(x) = Q(x)\,g(x) + r(x)$ *и* $f'(x) = Q'(x)\,g'(x) + r'(x)$. *Тогда*

(i) $Q(x) = Q'(x)$,

(ii) $r(x) = r'(x)\,x^k + r''(x)$, *где степень многочлена $r''(x)$ меньше, чем* $k + \deg f(x) - \deg g(x)$.

Доказательство. Утверждение теоремы является легко доказываемым следствием однозначности алгоритма деления. Начнем с равенства

$$f'(x) = Q'(x)\,g'(x) + r'(x).$$

Тогда

$$f'(x)\,x^k + f''(x) = Q'(x)\,g'(x)\,x^k + r'(x)\,x^k + f''(x),$$

$$f(x) = Q'(x)\,g(x) + r'(x)\,x^k + f''(x) - Q'(x)\,g''(x).$$

В силу однозначности алгоритма деления мы сможем заключить, что

$$Q(x) = Q'(x) \text{ и } r(x) = r'(x)\,x^k + f''(x) - Q'(x)\,g''(x),$$

если покажем, что

$$\deg [r'(x)\,x^k + f''(x) - Q'(x)\,g''(x)] < \deg g(x).$$

Но, используя условия теоремы, это легко проверить для каждого из трех многочленов, стоящих в левой части неравенства:

(i) $\deg [r'(x)\,x^k] < \deg g'(x) + k = \deg g(x)$,

(ii) $\deg f''(x) \leqslant k - 1 < \deg g(x)$,

(iii) $\deg [Q'(x)\,g''(x)] < \deg f'(x) - \deg g'(x) + k =$
$= (\deg f(x) - k) - (\deg g(x) - k) + k \leqslant \deg g(x)$.

Второе утверждение теоремы получится, если заметить, что из неравенств (ii) и (iii) следует, что степень многочлена $f''(x) - Q'(x)\,g''(x)$ меньше, чем $k + \deg f(x) - \deg g(x)$. □

Следствие 10.7.2. *Пусть целое число k удовлетворяет неравенству*

$$k \leqslant 2 \deg g(x) - \deg f(x).$$

Тогда частное $Q(x)$ при делении многочлена $f(x)$ на многочлен $g(x)$ не зависит от k младших коэффициентов многочленов $f(x)$ и $g(x)$ и k младших коэффициентов многочленов $f(x)$ и $g(x)$ оказывают влияние только на те коэффициенты остатка $r(x)$, индексы которых меньше, чем $k + \deg f(x) - \deg g(x)$. □

Согласно теореме 10.7.1, и многочлен-частное $Q(x)$ и сегмент многочлена-остатка $r(x)$ могут быть получены при усечении многочленов $f(x)$ и $g(x)$ до более коротких многочленов. Мы сейчас покажем, что отбрасывание сегмента многочлена-остатка $r(x)$ не затрагивает некоторые последующие итерации алгоритма Евклида. На самом деле, если выбрать k разумно, то примерно половина итераций может быть вычислена без знания отброшенного сегмента остатка $r(x)$.

Теорема 10.7.3. *Пусть* $f(x) = f'(x)\,x^k + f''(x)$ *и* $g(x) = g'(x)\,x^k + g''(x)$, *где* $\deg f''(x) < k$ *и* $\deg g''(x) < k$. *Пусть* $\deg f(x) = n$, $\deg g(x) < \deg f(x)$, *и пусть* $\mathbf{A}^{(r)}(x)$ *и* $\mathbf{A}^{(r)}(x)$ *обозначают матрицы из алгоритма Евклида, вычисленные соответственно для штрихованных и нештрихованных переменных. Тогда если* $\deg g'^{(r)}(x) \geqslant (n-k)/2$, *то для каждого r выполняется равенство* $\mathbf{A}^{(r)}(x) = \mathbf{A}'^{(r)}(x)$. *(Иными словами, частные остаточных последовательностей, порождаемых в процессе выполнения алгоритма Евклида, для нештрихованных и штрихованных переменных совпадают по меньшей мере до тех пор, пока не будет достигнут остаток $g'^{(r)}(x)$, степень которого меньше, чем половина степени $f'(x)$.)*

Доказательство. Доказательство сводится к применению следствия 10.7.2 на каждом шаге итерации алгоритма Евклида для нештрихованных и штрихованных переменных при начальных условиях $f^{(0)}(x) = f(x)$, $g^{(0)}(x) = g(x)$ и $f'^{(0)}(x) = f'(x)$, $g'^{(0)}(x) = g'(x)$.

Шаг 1. Следствие 10.7.2 можно применять на каждом шаге при условии, что выполняется неравенство $k \leqslant 2 \deg g^{(r)}(x) - \deg f^{(r)}(x)$. Но нам дано, что

$$\deg g'^{(r)}(x) \geqslant \frac{n-k}{2}.$$

Мы покажем, что это условие эквивалентно нужному условию, связывая степени штрихованных многочленов со степенями нештрихованных многочленов. Эти связи следуют из равенств

$$\begin{bmatrix} f^{(r)}(x) \\ g^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} \begin{bmatrix} f^{(r-1)}(x) \\ g^{(r-1)}(x) \end{bmatrix}, \quad \begin{bmatrix} f'^{(r)}(x) \\ g'^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} \begin{bmatrix} f'^{(r-1)}(x) \\ g'^{(r-1)}(x) \end{bmatrix}$$

и

$$\deg f'^{(0)}(x) = \deg f^{(0)}(x) - k,$$

$$\deg g'^{(0)}(x) = \deg g^{(0)}(x) - k.$$

Следовательно, поскольку многочлен-частное является одним и тем же многочленом в штрихованном и нештрихованном случаях, имеют место равенства

$$\deg f'^{(r)}(x) = \deg f^{(r)}(x) - k,$$

$$\deg g'^{(r)}(x) = \deg g^{(r)}(x) - k.$$

Таким образом, нам дано, что

$$\deg g^{(r)}(x) - k \geqslant \frac{n-k}{2} \geqslant \frac{\deg f^{(r)}(x) - k}{2},$$

так как $n = \deg f^{(0)}(x) \geqslant \deg f^{(r)}(x)$. Это неравенство сразу сводится к условию

$$k \leqslant 2 \deg g^{(r)}(x) - \deg f^{(r)}(x),$$

так что следствие 10.7.2 можно применять на каждом шаге итерации, если только на предыдущем шаге итерации многочлен $Q(x)$ вычислен правильно.

Шаг 2. Согласно следствию 10.7.2, каждый многочлен-частное вычисляется правильно, если только предыдущий многочлен-частное был вычислен правильно и последний полученный многочлен-остаток содержит достаточное число правильных коэффициентов. Чтобы проверить последнее условие, опять воспользуемся следствием 10.7.2, заменив число k числом

$$k^{(r)} = k + \deg f^{(r-1)}(x) - \deg g^{(r-1)}(x)$$

и полагая $k^{(0)} = k$. Согласно следствию 10.7.2 к началу r-го шага итерации многочлен-остаток будет правильно вычислен за исключением, возможно, младших $k^{(r)}$ членов, что согласно тому же следствию не влияет на вычисление многочлена-частного при условии, что $k^{(r)} \leqslant 2 \deg g^{(r)}(x) - \deg f^{(r)}(x)$. Тот факт, что это неравенство выполняется, проверяется следующим образом:

$$k^{(r)} - 2 \deg g^{(r)}(x) + \deg f^{(r)}(x) = k + \deg f^{(r-1)}(x) - \deg g^{(r-1)}(x) -$$

$$- 2 \deg g^{(r)}(x) + \deg f^{(r)}(x) = k + \deg f^{(r-1)}(x) - 2 \deg g^{(r)}(x) \leqslant$$

$$\leqslant k + n - 2\left(\frac{n+k}{2}\right) = 0,$$

где использованы неравенства $\deg f^{(r-1)}(x) \leqslant n$ и

$$\deg g^{(r)}(x) = \deg g'^{(r)}(x) + k \leqslant \frac{n+k}{2}.$$

Таким образом,

$$k^{(r)} \leqslant 2 \deg g^{(r)}(x) - \deg f^{(r)}(x)$$

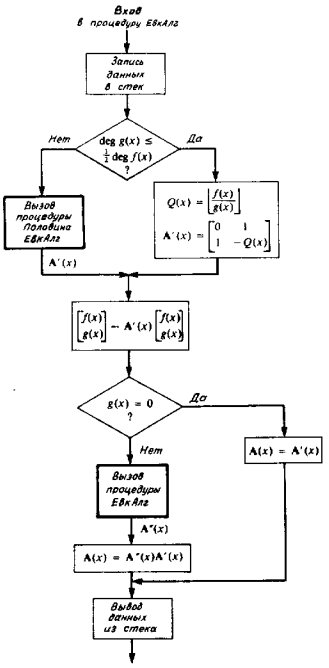


Рис. 10.7. Процедура ЕвкАлг.

и, следовательно, на r-м шаге итерации многочлен-частное не зависит от неизвестных коэффициентов. Это завершает доказательство теоремы. □

Условие «$\deg g'^{(r)}(x) \geqslant (n-k)/2$» теоремы для нештрихованных переменных можно переписать в виде «$\deg g^{(r)}(x) \geqslant (n+k)/2$». Эквивалентность этих условий используется в формулировке рекурсивной процедуры.

Разбиение алгоритма Евклида пополам показано на диаграмме на рис. 10.7. В основной точке ветвления алгоритма выносится ре-

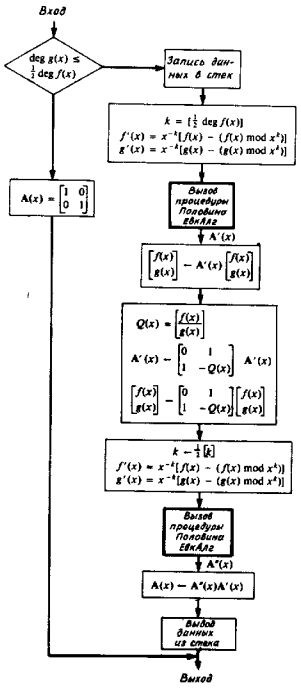


Рис. 10.8. Процедура Половина ЕвкАлг.

шение о немедленном разбиении задачи или о выполнении стандартной итерации алгоритма Евклида, поскольку процедура разбиения и дублирования неприменима. Последнее решение принимается только в том случае, когда степень многочлена $g(x)$ меньше половины степени многочлена $f(x)$. В этом случае стандартная итерация алгоритма Евклида на половину снижает размерность задачи, и мы ничего не теряем, поскольку процедура дублирования неприменима.

Форма полученной при разбиении второй половины задачи совпадает с формой исходной задачи, так что для ее вычисления можно опять вызвать процедуру ЕвкАлг. Первая половина задачи имеет аналогичную, но не идентичную форму, так как вычисления следует закончить, если степени многочленов слишком малы. Половину задачи можно также разбить пополам, так что мы приходим к формулировке вычислений в рекурсивном виде. Такая рекурсивная процедура показана на рис. 10.8. Обоснование этой процедуры можно провести методом индукции. Нам надо показать, что вычисляются те же самые итерации алгоритма Евклида, которые описываются теоремой 10.7.3. Это всегда выполняется при n равном 1 и 2, так как в этом случае мы движемся по левой ветви на рис. 10.8. При движении по правой ветви в результате первого обращения к процедуре «Половина ЕвкАлг» согласно предположению индукции степень $\deg f(x)$ уменьшается до $(n + k)/2$, что составляет примерно 3/4 исходной степени многочлена $f(x)$. Второе обращение к процедуре «Половина ЕвкАлг» завершает описываемые теоремой 10.7.3 итерации.

10.8. Вычисление тригонометрических функций

Для вычисления тригонометрических функций обычно используется некоторое разложение в степенной ряд. Такие методы сильно отличаются от рассматриваемых нами методов. Имеется, однако, ряд приложений с привкусом стратегии дублирования, для которых более подходящими являются описываемые нами методы.

Первый из рассматриваемых нами методов дается алгоритмом одновременного вычисления значений функций $\sin\theta$ и $\cos\theta$ при заданном угле θ. Процесс вычисления основывается на тригонометрических тождествах для удвоения угла

$$\sin 2\theta = 2\sin\theta\cos\theta,$$
$$\cos 2\theta = 1 - 2\sin^2\theta,$$

или

$$\sin 2\theta = 2\sin\theta - 2\sin\theta\,\mathrm{vers}\,\theta,$$
$$\mathrm{vers}\,2\theta = 2\sin^2\theta,$$

где $\mathrm{vers}\,\theta = 1 - \cos\theta$.

Для вычисления начальных значений данный угол θ делится на некоторую степень двух и используются приближенные формулы для малых углов: $\sin\frac{\theta}{2^m} = \frac{\theta}{2^m}$ и $\mathrm{vers}\frac{\theta}{2^m} = 0$.

Точность алгоритма управляется выбором числа m. Величина угла θ выражается в радианах и расположена в пределах $\pm\pi$. Алгоритм автоматически помещает результат в правильный квад-

рант, так что не требуется никакого специального определения квадранта. Окончательным результатом работы алгоритма является вычисление $\sin\theta$ и $1 - \cos\theta$.

Для того, чтобы числовые величины не становились слишком малыми, введем новые переменные

$$X_n = \frac{2^m}{2^n}(\sin\theta)_n, \qquad Y_n = \frac{2^m}{2^n}(\operatorname{vers}\theta)_n.$$

При этом рекуррентные уравнения принимают вид

$$X_n = X_{n-1} - 2^{n+1-m}X_{n-1}Y_{n-1},$$
$$X_n = 2^{n+1-m}X_{n-1}^2,$$

где m обозначает полное число шагов рекурсии, и начальные условия равны $X_0 = \theta$, $Y_0 = 0$. На первой итерации точность алгоритма определяется погрешностью начального приближения и равна $(X_0)_e = -(1/6)(\theta/2^m)^3$. Эта погрешность приводит к финальным погрешностям, ограниченным сверху величиной $(1/6)\,\pi^3 2^{-2m}$:

$$(\sin\theta)_e \leqslant \frac{1}{6}\pi^3 2^{-2m}, \quad (\cos\theta)_e \leqslant \frac{1}{6}\pi^3 2^{-2m}.$$

Следовательно, если при умножении обеспечено достаточное число точных битов, каждый шаг итерации увеличивает точность в двух битах.

Второй тригонометрический алгоритм представляет собой метод поворота. Его можно использовать для вычисления декартовых координат точки, повернутой на угол θ,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix},$$

или для вычисления полярных координат точки, заданной декартовыми координатами,

$$\theta = \operatorname{arctg}\frac{x}{y}, \quad r = \sqrt{x^2 + y^2}.$$

Алгоритм состоит из некоторых вычислений и проверки логических условий. В основе алгоритма лежит тот факт, что тригонометрические тождества

$$\sin\theta = \frac{\operatorname{tg}\theta}{\sqrt{1+\operatorname{tg}^2\theta}} \text{ и } \cos\theta = \frac{1}{\sqrt{1+\operatorname{tg}^2\theta}}$$

позволяют легко вычислить поворот вектора на конкретный угол $\theta = \operatorname{arctg}(2^{-k})$. В этом случае

$$\sin[\operatorname{arctg}2^{-k}] = \frac{2^{-k}}{\sqrt{1+(2^{-k})^2}} \text{ и } \cos[\operatorname{arctg}2^{-k}] = \frac{1}{\sqrt{1+(2^{-k})^2}},$$

Номер интерации k	Масштаб 2^{-k}	Приращение угла $\theta_k = \operatorname{tan}^{-1}2^{-k}$
−1	0	90°
0	1	45°
1	1/2	26.565051°
2	1/4	14.036243°
3	1/8	7.125016°
4	1/16	3.576334°
5	1/32	1.789911°
6	1/64	0.895174°
7	1/128	0.447614°
8	1/256	0.223811°
9	1/512	0.111906°
10	1/1024	0.055953°

Рис. 10.9. Алгоритм поворота координат.

так что поворот вектора (x, y) на угол $\theta_k = \operatorname{arctg}2^{-k}$ задается равенствами

$$x' = \frac{1}{\sqrt{1+(2^{-k})^2}}[x + 2^{-k}y],$$
$$y' = \frac{1}{\sqrt{1+(2^{-k})^2}}[y - 2^{-k}x].$$

Для поворота на отрицательный угол $\operatorname{arctg}2^{-k}$ используются те же уравнения, но с заменой знака у 2^{-k}. Это не влияет на величину стоящего перед скобкой множителя. Длина вектора умножается на фиксированную константу.

Произвольный угол θ может быть представлен в виде

$$\theta = \pm 90° + \sum_{k=0}^{\infty}(\pm\operatorname{arctg}2^{-k})$$

или

$$\theta = \xi_{-1}\,90° + \sum_{k=0}^{\infty}\xi_k \operatorname{arctg}2^{-k} = \sum_{k=-1}^{\infty}\xi_k\theta_k,$$

где $k = -1, 0, 1, \ldots$, ξ_k равно 1 или -1, а θ_k затабулированы на рис. 10.9. Поворот на угол θ просто сводится к поворотам на каждый из углов θ_k с направлением, определяемым ξ_k.

Независимо от знака каждого индивидуального поворота после m итераций усиление равно

$$\prod_{k=1}^{m}\sqrt{1+2^{-2k}}.$$

В окончательном виде алгоритм показан на рис. 10.10. Правило поворота вектора на первом шаге на угол $\pm 90°$ несколько отличается от правила остальных поворотов. В дальнейшем решение

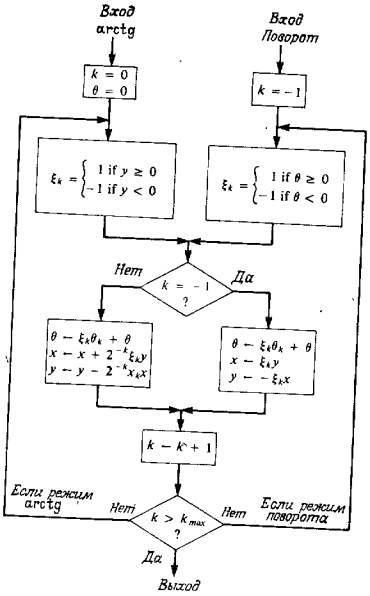


Рис. 10.10.

о повороте на угол θ_k или $-\theta_k$ принимается в зависимости от знаков координат x и y. Это определяет дальнейшее сложение или вычитание с умножением на скаляр. Так как скалярный множитель равен 2^{-k}, то умножение сводится к двоичному сдвигу, так что алгоритм почти не содержит умножений. Даваемое алгоритмом представление величины усиления известно. Оно может быть нивелировано умножением на обратную величину после выполнения всех шагов алгоритма. В больших задачах еще лучше упрятать его в константы, встречающиеся в других местах.

Для вычисления arctg (x/y) вектор (x, y) на каждой итерации вращается в том направлении, которое уменьшает текущее значение координаты y, а формирование угла θ проводится суммированием величин θ_k с учетом их знаков.

Задачи

10.1. Найти алгоритм дублирования для умножения вектор-столбца на теплицеву матрицу.

10.2. Провести сортировку списка {3, 1, 5, 2, 7, 3, 9, 8, 2, 6, 1, 4, 9, 2, 5, 1}, используя алгоритм слияния.

10.3. **а.** Привести блок-схему БПФ-алгоритма Кули — Тьюки по основанию два с прореживанием по частоте в виде алгоритма дублирования.
б. Привести блок-схему алгоритма дублирования в рекурсивной форме для вычисления фильтр-секции, длина которой равна степени двух.
в. Привести блок-схему алгоритма дублирования в рекурсивной форме для вычисления произведения многочленов от двух переменных.

10.4. Пусть n равно степени двух и пусть a и b — два произвольных целых числа, запись которых содержит n позиций. Найти эффективный алгоритм умножения чисел a и b, используя дублирование.

10.5. Разработать алгоритм транспортирования матрицы на процессоре, который может одновременно запоминать четыре столбца из таблицы. Построить блок-схему алгоритма. Сколько необходимо обменов столбцами?

10.6. Сколько умножений потребуется для вычисления следующего преобразования координат?

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

10.7. Используя приведенный на рис. 10.10 алгоритм, вычислить arctg (2). Установить точность алгоритма, сравнивая вычисленное значение с истинным.

10.8. Используя алгоритм Штрассена, вычислить произведение матриц

$$\begin{bmatrix} 1 & 5 \\ 7 & 3 \end{bmatrix} \begin{bmatrix} 4 & 8 \\ 6 & 2 \end{bmatrix}.$$

Замечания

Стратегия деления задачи пополам и дублирования является одной из основных и появилась во многих работах. Данная короткая глава только затрагивает этот раздел общей теории алгоритмов. Более широкое введение можно найти в работах Кнута (1968) и Ахо, Хопкрофта и Ульмана (1974). Общеизвестные примеры алгоритмов дублирования и сортировки, такие как рассмотренный алгоритм слиянием, или как алгоритмы пирамидальной сортировки и быстрой сортировки, были введены Уильямсоном (1964) и Хоаром (1962) соответственно. Известны и многие другие алгоритмы сортировки. Описание БПФ-алгоритма Кули—Тьюки с точки зрения стратегии дублирования предложил Стейглиц (1974). Фидуччиа (1972) применил к построению БПФ-алгоритма Кули — Тьюки полиномиальные идеи алгоритма Герцеля, позволившие использовать стратегию деления задачи пополам и дублирования. Стратегию дублирования для задачи транспонирования матрицы использовал Эклунд (1972). Первая усиленная с помощью стратегии дублирования форма алгоритма Евклида появилась в книге Ахо, Хопкрофта и Ульмана в связи с вычислением наибольшего общего знаменателя.

Алгоритм поворота вектора используется уже многие годы под названием «cordic algorithm» и обычно приписывается Волверу (1959); детальная его проработка выполнена Волтером (1971). Алгоритм дублирования для вычисления синусов и косинусов взят из работы Блейхута и Волдекера (1970).

Глава 11

БЫСТРЫЕ МЕТОДЫ РЕШЕНИЯ ТЕПЛИЦЕВЫХ СИСТЕМ

Стандартный метод решения системы n линейных уравнений с n неизвестными состоит в записи этой системы в матричном виде, $\mathbf{Af} = \mathbf{g}$, и отыскании решения либо с помощью обращения матрицы, $\mathbf{f} = \mathbf{A}^{-1}\mathbf{g}$, либо с помощью метода Гаусса. Сложность стандартных методов обращения матрицы пропорциональна n^3. Иногда матрица имеет такой частный вид, что им можно воспользоваться для построения более быстрого алгоритма.

Система линейных уравнений называется теплицевой, если ее матрица $\mathbf{A}$ теплицева. Задача решения теплицевой системы уравнений возникает во многих приложениях, включающих спектральное оценивание, линейное предсказание, построение авторегрессионных фильтров и теорию кодов, контролирующих ошибки. Так как теплицевы системы уравнений имеют очень частный вид, то для их решения существуют методы, которые намного лучше общих методов решения систем линейных уравнений. Эти методы, составляющие предмет исследования в данной главе, справедливы для произвольного поля.

Рассматриваемые в настоящей главе алгоритмы несколько отличаются от тех алгоритмов, которые мы построили для вычисления свертки или преобразования Фурье. Свертка и преобразование по существу сводятся к матричному умножению, а в данной главе рассматривается задача решения системы линейных уравнений. Поэтому неудивительно, что мы не сможем прямо воспользоваться построенными ранее алгоритмами, хотя такой метод, как дублирование, окажется полезным.

11.1. Алгоритмы Левинсона и Дурбина

Задачи обращения свертки или спектрального оценивания содержат задачу решения теплицевой системы уравнений, задаваемой матричным равенством

$$\mathbf{Af} = \mathbf{g},$$

где $\mathbf{A}$ представляет собой теплицеву $(n \times n)$-матрицу

$$\mathbf{A} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_{-1} & a_0 & a_1 & \cdots & a_{n-2} \\ a_{-2} & a_{-1} & a_0 & & \vdots \\ \vdots & & & & \vdots \\ a_{-n+1} & & & & a_0 \end{bmatrix}.$$

Вычислительная задача состоит в вычислении вектора $\mathbf{f}$ по заданным вектору $\mathbf{g}$ и теплицевой матрице $\mathbf{A}$. Одним из возможных методов решения, конечно, является обращение матрицы $\mathbf{A}$, однако, практически n может быть больше 100 или даже 1000, так что важно отыскать более эффективные методы решения задачи.

Алгоритм Левинсона является эффективным алгоритмом, применимым в том частном случае, когда матрица $\mathbf{A}$ является одновременно теплицевой и симметричной. В этом случае система уравнений записывается в виде

$$\left[\begin{array}{cccc|c} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_1 & a_0 & a_1 & \cdots & a_{n-2} \\ a_2 & a_1 & a_0 & \cdots & a_{n-3} \\ \vdots & & & & \vdots \\ a_{n-2} & a_{n-3} & & \cdots & a_1 \\ \hline a_{n-1} & a_{n-2} & & & a_0 \end{array}\right] \left[\begin{array}{c} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-2} \\ \hline f_{n-1} \end{array}\right] = \left[\begin{array}{c} g_0 \\ g_1 \\ g_2 \\ \vdots \\ g_{n-2} \\ \hline g_{n-1} \end{array}\right].$$

Разделение матрицы и векторов на блоки здесь сделано для того, чтобы подчеркнуть лежащую в основе алгоритма повторяемость структуры. На каждом шаге (итерации) к матрице $\mathbf{A}$ добавляется «половина границы», состоящая из новой строки снизу и нового столбца справа. Используя обменную матрицу $\mathbf{J}$, уравнение $\mathbf{Af} = \mathbf{g}$ можно переписать в виде $\mathbf{JAJJf} = \mathbf{Jg}$, и $\mathbf{JAJ} = \mathbf{A}$, так как матрица $\mathbf{A}$ является симметричной и теплицевой. Следовательно, имеет место также равенство

$$\begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_1 & a_0 & a_1 & \cdots & a_{n-2} \\ a_2 & a_1 & a_0 & \cdots & a_{n-3} \\ \vdots & & & & \vdots \\ a_{n-1} & a_{n-2} & & & a_0 \end{bmatrix} \begin{bmatrix} f_{n-1} \\ f_{n-2} \\ \vdots \\ f_1 \\ f_0 \end{bmatrix} = \begin{bmatrix} g_{n-1} \\ g_{n-2} \\ \vdots \\ g_1 \\ g_0 \end{bmatrix}.$$

Эта форма уравнения также будет использоваться при построении алгоритма Левинсона. Алгоритм оперирует с $(r \times r)$-подматрицами матрицы $\mathbf{A}$ вида

$$\mathbf{A}^{(r)} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{r-1} \\ a_1 & a_0 & a_1 & \cdots & a_{r-2} \\ a_2 & a_1 & a_0 & \cdots & a_{r-3} \\ \vdots & & & & \vdots \\ a_{r-1} & & & & a_0 \end{bmatrix}.$$

которые получаются из матрицы A отбрасыванием последних $n - r$ столбцов и такого же числа нижних строк.

Алгоритм Левинсона является итеративным; эти итерации индексируются буквой r. На r-м шаге алгоритм Левинсона вычисляет решение r-го усечения задачи:

$$\begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} \\ a_1 & a_0 & a_1 & \dots & a_{r-2} \\ \vdots & & & & \\ a_{r-1} & a_{r-2} & a_{r-3} & \dots & a_0 \end{bmatrix} \begin{bmatrix} f_0^{(r)} \\ f_1^{(r)} \\ \vdots \\ f_{r-1}^{(r)} \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \end{bmatrix}.$$

где верхний индекс r указывает на то, что вектор $\mathbf{f}^{(r)}$ является решением r-го усечения задачи. Ясно, что если даже $\mathbf{f}^{(r)}$ и является решением усеченной задачи, он не равен усечению решения $\mathbf{f}$ исходной задачи. Алгоритм Левинсона рекурсивно модернизирует $\mathbf{f}^{(r)}$ таким образом, что $\mathbf{f}^{(n)}$ равно решению исходной задачи.

Помимо вектора $\mathbf{f}^{(r)}$ в итерациях алгоритма Левинсона участвует несколько рабочих переменных, задаваемых следующим образом. Скалярные рабочие переменные обозначаются через α_r, β_r и γ_r; рабочий вектор длины r обозначается через $\mathbf{t}^{(r)}$. Все эти рабочие переменные выбираются на каждом r-м шаге так, чтобы выполнялось следующее дополнительное матричное равенство:

$$\begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} \\ a_1 & a_0 & a_1 & \dots & a_{r-2} \\ a_2 & a_1 & a_0 & \dots & a_{r-3} \\ \vdots & & & & \\ a_{r-1} & a_{r-2} & & \dots & a_0 \end{bmatrix} \begin{bmatrix} t_0^{(r)} \\ t_1^{(r)} \\ \vdots \\ t_{r-1}^{(r)} \end{bmatrix} = \begin{bmatrix} \alpha_r \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

где в стоящем в правой части равенства векторе все за исключением одной компоненты равны нулю. Введение рабочего вектора $\mathbf{t}^{(r)}$ и дополнительного уравнения является разумной идеей, позволяющей продолжать итеративный процесс. Мы хотим итерации организовать так, чтобы все уравнения имели одну и ту же форму. Следующая итерация начинается с уравнения

$$\left[\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{r-2} & a_{r-1} \\ \vdots & & & & & \\ a_{r-1} & a_{r-2} & & \dots & a_0 & a_1 \\ \hline a_r & a_{r-1} & & & a_1 & a_0 \end{array}\right] \begin{bmatrix} f_0^{(r)} \\ f_1^{(r)} \\ \vdots \\ f_{r-1}^{(r)} \\ 0 \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \\ \gamma_r \end{bmatrix},$$

которое определяет γ_r, и с уравнения

$$\left[\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{r-2} & a_{r-1} \\ \vdots & & & & & \\ a_{r-1} & a_{r-2} & & \dots & a_0 & a_1 \\ \hline a_r & a_{r-1} & & & a_1 & a_0 \end{array}\right] \begin{bmatrix} t_0^{(r)} \\ t_1^{(r)} \\ \vdots \\ t_{r-1}^{(r)} \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha_r \\ 0 \\ \vdots \\ 0 \\ \beta_r \end{bmatrix},$$

которое определяет β_r и в правой части которого стоит вектор, все внутренние компоненты которого равны нулю. Если $\gamma_r = g_r$ и $\beta_r = 0$, то $\mathbf{f}^{(r+1)}$ и $\mathbf{t}^{(r+1)}$ равны соответственно $\mathbf{f}^{(r)}$ и $\mathbf{t}^{(r)}$, но с добавленными нулевыми компонентами, увеличивающими их длину. В этом случае итерация завершается. В противном случае векторы $\mathbf{f}^{(r)}$ и $\mathbf{t}^{(r)}$ должны быть модифицированы.

Как всегда в рекурсивном алгоритме, выберем начальные значения всех переменных так, чтобы при $r = 0$ выполнялись все уравнения, и предположим, что эти уравнения выполняются к концу $(r-1)$-го шага итераций. Нам надо только показать, как надо модифицировать рабочие переменные, чтобы эти уравнения выполнялись и к концу r-го шага. Но мы можем также записать уравнения

$$\begin{bmatrix} a_r & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{r-2} & a_{r-1} \\ \vdots & & & & & \\ a_{r-1} & a_{r-2} & & \dots & a_0 & a_1 \\ a_r & a_{r-1} & & & a_1 & a_0 \end{bmatrix} \begin{bmatrix} 0 \\ f_{r-1}^{(r)} \\ \vdots \\ f_1^{(r)} \\ f_0^{(r)} \end{bmatrix} = \begin{bmatrix} \gamma_r \\ g_{r-1} \\ \vdots \\ g_1 \\ g_0 \end{bmatrix}$$

и

$$\begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & \dots & a_{r-2} & a_{r-1} \\ \vdots & & & & & \\ a_{r-1} & a_{r-2} & & \dots & a_0 & a_1 \\ a_r & a_{r-1} & & \dots & a_1 & a_0 \end{bmatrix} \begin{bmatrix} 0 \\ t_{r-1}^{(r)} \\ \vdots \\ t_1^{(r)} \\ t_0^{(r)} \end{bmatrix} = \begin{bmatrix} \beta_r \\ 0 \\ \vdots \\ 0 \\ \alpha_r \end{bmatrix}.$$

Из этих уравнений можно сформировать следующую итерацию. Пусть для подлежащих дальнейшему выбору констант k_1 и k_2 выполняется

$$\begin{bmatrix} t_0^{(r+1)} \\ t_1^{(r+1)} \\ \vdots \\ t_{r-1}^{(r+1)} \\ t_r^{(r+1)} \end{bmatrix} = k_1 \begin{bmatrix} t_0^{(r)} \\ t_1^{(r)} \\ \vdots \\ t_{r-1}^{(r)} \\ 0 \end{bmatrix} + k_2 \begin{bmatrix} 0 \\ t_{r-1}^{(r)} \\ \vdots \\ t_1^{(r)} \\ t_0^{(r)} \end{bmatrix}.$$

Тогда

$$\begin{bmatrix} a_0 & a_1 & \dots & a_r \\ a_1 & a_0 & \dots & a_{r-1} \\ \vdots & & & \vdots \\ a_{r-1} & a_{r-2} & \dots & a_1 \\ a_r & a_{r-1} & \dots & a_0 \end{bmatrix} \begin{bmatrix} t_0^{(r+1)} \\ t_1^{(r+1)} \\ \vdots \\ t_{r-1}^{(r+1)} \\ t_r^{(r+1)} \end{bmatrix} = k_1 \begin{bmatrix} \alpha_r \\ 0 \\ \vdots \\ 0 \\ \beta_r \end{bmatrix} + k_2 \begin{bmatrix} \beta_r \\ 0 \\ \vdots \\ 0 \\ \alpha_r \end{bmatrix} = \begin{bmatrix} \alpha_{r+1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix},$$

откуда следует, что k_1 и k_2 должны выбираться так, чтобы выполнялись равенства

$$0 = k_1\beta_r + k_2\alpha_r \text{ и } \alpha_{r+1} = k_1\alpha_r + k_2\beta_r.$$

В силу произвольности выбора k_1 положим $k_1 = \alpha_r$. (Различные возможности выбора k_1 приводят, однако, к различной точности.) Тогда $k_2 = -\beta_r$ и $\alpha_{r+1} = \alpha_r^2 - \beta_r^2$. Наконец, положим

$$\begin{bmatrix} f_0^{(r+1)} \\ f_1^{(r+1)} \\ \vdots \\ f_{r-1}^{(r+1)} \\ f_r^{(r+1)} \end{bmatrix} = \begin{bmatrix} f_0^{(r)} \\ f_1^{(r)} \\ \vdots \\ f_{r-1}^{(r)} \\ 0 \end{bmatrix} + k_3 \begin{bmatrix} t_r^{(r+1)} \\ t_{r-1}^{(r+1)} \\ \vdots \\ t_1^{(r+1)} \\ t_0^{(r+1)} \end{bmatrix},$$

где константа k_3 также подлежит вычислению; тогда

$$\begin{bmatrix} a_0 & a_1 & \dots & a_r \\ a_1 & a_0 & \dots & a_{r-1} \\ \vdots & & & \vdots \\ a_{r-1} & & & a_1 \\ a_r & & & a_0 \end{bmatrix} \begin{bmatrix} f_0^{(r+1)} \\ f_1^{(r+1)} \\ \vdots \\ f_{r-1}^{(r+1)} \\ f_r^{(r+1)} \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \\ \gamma_r \end{bmatrix} + k_3 \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \alpha_{r+1} \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{r-1} \\ g_r \end{bmatrix}$$

где для обеспечения правого равенства k_3 должно быть выбрано так, чтобы $\gamma_r + k_3\alpha_{r+1} = g_r$. Это завершает итерацию.

Алгоритм Левинсона подытожен на рис. 11.1. На нем векторы $\mathbf{t}^{(r)}$ и $\mathbf{f}^{(r)}$ представлены соответственно многочленами $t(x) = t_r^{(r)}x^r + t_{r-1}^{(r)}x^{r-1} + \dots + t_1^{(r)}x + t_0^{(r)}$ и $f(x) = f_r^{(r)}x^r + f_{r-1}^{(r)}x^{r-1} + \dots + f_1^{(r)}x + f_0^{(r)}$, в которых опущен индекс r. Вычисления не нуждаются в фактическом вычислении матрицы $\mathbf{A}^{(r)}$; во время итераций достаточно вычислять многочлены $f(x)$ и $t(x)$.

Сложность r-го прохода по ветви пропорциональна r, и всего имеется n проходов. Следовательно, сложность алгоритма Левинсона пропорциональна n^2. Шаг рекурсии приводит к неудаче только тогда, когда возникает деление на нуль, что происходит только тогда, когда одна из главных подматриц вырождена.

Алгоритм Левинсона применим в любом поле. В частности, в поле комплексных чисел его можно использовать именно в том виде, в котором он был выписан. Однако в связанных с комплексным нолем приложениях симметричная теплицева матрица возникает не часто; более распространенной в этом случае является эрмитова теплицева матрица. Алгоритм Левинсона применим и в этом случае, но с переходом в соответствующих местах вычислений к комплексно сопряженным величинам. Реконструкцию алгоритма для этого случая сделать легко.

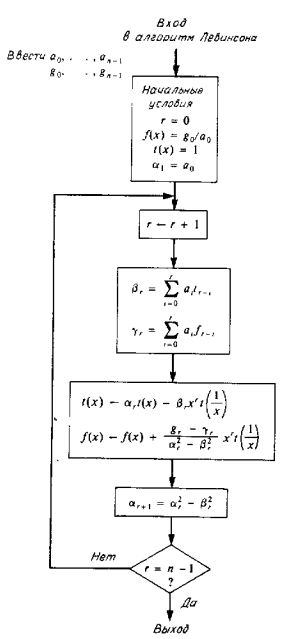


Рис. 11.1. Алгоритм Левинсона.

Иногда вместо алгоритма Левинсона лучше воспользоваться так называемым *алгоритмом Дурбина*. Это возможно в тех случаях, когда матрица симметрична и теплицева, а вектор в правой части формируется из элементов теплицевой матрицы так, что уравнение принимает следующий вид:

$$\left[\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \dots & a_{n-2} & a_{n-1} \\ a_1 & a_0 & a_1 & & a_{n-3} & a_{n-2} \\ a_2 & a_1 & a_0 & & a_{n-4} & a_{n-3} \\ \vdots & & & & & \vdots \\ a_{n-2} & & & & a_0 & a_1 \\ \hline a_{n-1} & & & & a_1 & a_0 \end{array}\right] \begin{bmatrix} f_0 \\ f_1 \\ \\ \vdots \\ f_{n-2} \\ f_{n-1} \end{bmatrix} = -\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}.$$

Матрицу и оба вектора можно опять разбить на блоки, чтобы выявить лежащую в основе алгоритма повторяющуюся блочную структуру. Специальные свойства стоящего справа вектор-столбца, формируемого из элементов теплицевой матрицы, позволяют вдвое уменьшить необходимые в алгоритме Левинсона вычисления, так как в итерации включается только один многочлен. В рассматриваемом виде теплицевы матрицы часто встречаются в задачах спектрального анализа, и в этих случаях полезен алгоритм Дурбина.

r-й шаг алгоритма Дурбина начинается с решения следующей усеченной задачи:

$$\begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{r-1} \\ a_1 & a_0 & a_1 & & a_{r-2} \\ \vdots & & & & \\ a_{r-1} & a_{r-2} & & \dots & a_0 \end{bmatrix} \begin{bmatrix} f_0^{(r)} \\ f_1^{(r)} \\ \vdots \\ f_{r-1}^{(r)} \end{bmatrix} = - \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_r \end{bmatrix}.$$

Следующая итерация начинается с уравнения

$$\left[\begin{array}{ccccc|c} a_0 & a_1 & a_2 & \dots & a_{r-1} & a_r \\ a_1 & a_0 & a_1 & & a_{r-2} & a_{r-1} \\ \vdots & & & & & \vdots \\ a_{r-1} & a_{r-2} & & & a_0 & a_1 \\ \hline a_r & a_{r-1} & & & a_1 & a_0 \end{array}\right] \begin{bmatrix} f_0^{(r)} \\ f_1^{(r)} \\ \vdots \\ f_{r-1}^{(r)} \\ 0 \end{bmatrix} = - \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_r \\ \gamma_r \end{bmatrix},$$

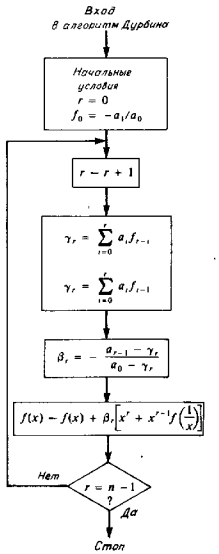
определяющего переменную γ_r. Целью итерации является такая модификация вектора $\mathbf{f}^{(r)}$, при которой величина γ_r становится равной a_{r+1}. Пусть $\mathbf{f}^{(r+1)}$ задается равенством

$$\begin{bmatrix} f_0^{(r+1)} \\ f_1^{(r+1)} \\ \vdots \\ f_{r-1}^{(r+1)} \\ f_r^{(r+1)} \end{bmatrix} = \begin{bmatrix} f_0^{(r)} \\ f_1^{(r)} \\ \vdots \\ f_{r-1}^{(r)} \\ 0 \end{bmatrix} + k_r \begin{bmatrix} f_{r-1}^{(r)} \\ f_{r-2}^{(r)} \\ \vdots \\ f_0^{(r)} \\ 0 \end{bmatrix} + \beta_r \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

Если нам удастся выбрать k_r и β_r так, чтобы регенерировать желаемый вид уравнений, то мы построим хороший алгоритм. Но при некоторых γ_r и γ_r' имеет место равенство

$$\left[\begin{array}{ccc|c} a_0 & \dots & a_{r-1} & a_r \\ \vdots & & & \vdots \\ a_{r-1} & \dots & a_0 & a_1 \\ \hline a_r & & a_1 & a_0 \end{array}\right] \begin{bmatrix} f_0^{(r+1)} \\ \vdots \\ f_{r-1}^{(r+1)} \\ f_r^{(r+1)} \end{bmatrix} = - \begin{bmatrix} a_1 \\ \vdots \\ a_r \\ \gamma_r \end{bmatrix} - k_r \begin{bmatrix} a_r \\ \vdots \\ a_1 \\ \gamma_r' \end{bmatrix} + \beta_r \begin{bmatrix} a_r \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = - \begin{bmatrix} a_1 \\ \vdots \\ a_r \\ a_{r+1} \end{bmatrix}.$$

Рис. 11.2. Алгоритм Дурбина.



Для обеспечения нужного равенства k_r и β_r в окончательном виде надо выбрать так, чтобы выполнялись условия

$$k_r - \beta_r = 0 \text{ и}$$

$$-\gamma_r - k_r \gamma_r' + \beta_r a_0 = -a_{r+1},$$

где

$$\gamma_r = -\sum_{i=1}^{r} f_{r-i}^{(r)} a_i, \quad \gamma_r' = -\sum_{i=1}^{r} f_{i-1}^{(r)} a_i.$$

Следовательно, k_r и β_r надо полагать равными

$$k_r = \beta_r \text{ и } \beta_r = -\frac{(a_{r+1} - \gamma_r)}{(a_0 - \gamma_r')}.$$

Выписанные равенства позволяют продолжить решение r-го шага до решения $(r+1)$-го шага, так что тривиальное решение нулевого шага рекурсивно продолжается до решения n-го шага. Это завершает построение алгоритма Дурбина. Окончательная форма алгоритма представлена блок-схемой на рис. 11.2.

11.2. Алгоритм Тренча

В зависимости от наличия различных дополнительных свойств теплицеву систему уравнений

$$\begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \\ a_{-1} & a_0 & a_1 & \dots & a_{n-2} \\ a_{-2} & a_{-1} & a_0 & \dots & a_{n-3} \\ \vdots & & & & \\ a_{-n+1} & a_{-n+2} & a_{-n+3} & \dots & a_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{n-1} \end{bmatrix}$$

можно решать многими различными алгоритмами. В разд. 11.1 был рассмотрен случай, когда теплицева матрица симметрична. В этом случае применим алгоритм Левинсона или Дурбина. В настоящем параграфе рассматривается более общий случай, когда

теплицева матрица не является симметричной. В этом более общем случае алгоритмом решения является *алгоритм Тренча*. В своей наиболее полной форме алгоритм Тренча вычисляет обратную матрицу $\mathbf{A}^{-1}$ и вектор $\mathbf{f}$. Мы остановимся только на вычислении матрицы $\mathbf{A}^{-1}$.

В общем случае не всякая теплицева матрица симметрична, но всякая теплицева матрица персимметрична. Матрица называется *персимметричной*, если стоящие симметрично относительно побочной диагонали ее элементы равны. Если $(n \times n)$-матрица $\mathbf{A}$ персимметрична, то $a_{ij} = a_{n+1-j,\, n+1-i}$. Другими словами, матрица $\mathbf{A}$ персимметрична тогда и только тогда, когда $\mathbf{JAJ} = \mathbf{A}$, где $\mathbf{J}$ — обменная матрица той же размерности, что и $\mathbf{A}$. Матрица, обратная к теплицевой, в общем случае теплицевой не является, но остается персимметричной.

Теорема 11.2.1. *Обратная к персимметричной матрице* $\mathbf{A}$ *является персимметричной.*

Доказательство. Пусть $\mathbf{J}$ обозначает обменную матрицу той же самой размерности, что и матрица $\mathbf{A}$. Напомним, что $\mathbf{J}^2 = \mathbf{I}$, так что $\mathbf{J}^{-1} = \mathbf{J}$. Но $\mathbf{JAJ} = \mathbf{A}$. Следовательно, $\mathbf{JA}^{-1}\mathbf{J} = \mathbf{A}^{-1}$ и матрица $\mathbf{A}^{-1}$ является персимметричной. □

Другое свойство, которое сохраняется при обращении теплицевой матрицы, состоит в том, что она полностью определяется своей границей. Более точно, она полностью определяется заданием своих первой строки и первого столбца или своих последней строки и последнего столбца. В продолжение этого раздела дается доказательство этого свойства и затем выписывается рекурсивная процедура вычисления матрицы по ее нижней полуграницe, состоящей из последней строки и последнего столбца. Для вычисления матрицы $\mathbf{A}^{-1}$ в алгоритме Тренча используется полуграница матрицы $\mathbf{A}^{-1}$.

Определим вектор-столбцы $\mathbf{a}_+$ и $\mathbf{a}_-$ длины $r - 1$ каждый равенствами

$$\mathbf{a}_+ = (a_1,\ a_2,\ a_3,\ \ldots,\ a_{r-1})^T,$$
$$\mathbf{a}_- = (a_{-1},\ a_{-2},\ a_{-3},\ \ldots,\ a_{-(r-1)})^T.$$

Пусть $\tilde{\mathbf{a}}_+$ и $\tilde{\mathbf{a}}_-$ обозначают два вектора, компонентами которых являются компоненты векторов $\mathbf{a}_+$ и $\mathbf{a}_-$ соответственно, выписанные в обратном порядке. Иными словами, $\tilde{\mathbf{a}}_+ = \mathbf{J}\mathbf{a}_+$ и $\tilde{\mathbf{a}}_- = \mathbf{J}\mathbf{a}_-$, где $\mathbf{J}$ равно обменной матрице размерности $(r-1) \times (r-1)$. Матрицу $\mathbf{A}^{(r)}$ можно следующим образом разбить на блоки:

$$\mathbf{A}^{(r)} = \left[\begin{array}{c|c} \mathbf{A}^{(r-1)} & \tilde{\mathbf{a}}_- \\ \hline \tilde{\mathbf{a}}_-^T & a_0 \end{array}\right]$$

Аналогично можно разбить на блоки обратную матрицу $\mathbf{B}^{(r)} = (\mathbf{A}^{(r)})^{-1}$:

$$\mathbf{B}^{(r)} = \left[\begin{array}{c|c} \mathbf{M}^{(r)} & \tilde{\mathbf{b}}_+^{(r)} \\ \hline \tilde{\mathbf{b}}_-^{(r)T} & b_0^{(r)} \end{array}\right].$$

Нашей целью является описание блоков этого последнего разбиения.

Теорема 11.2.2. *Блоки обратной матрицы* $\mathbf{B}^{(r)}$ *удовлетворяют равенству*

$$\mathbf{M}^{(r)} = \mathbf{B}^{(r-1)} + \frac{1}{b_0^{(r)}}\tilde{\mathbf{b}}_+^{(r)}\tilde{\mathbf{b}}_-^{(r)T}.$$

Доказательство. Так как $\mathbf{A}^{(r)}\mathbf{B}^{(r)} = \mathbf{I}$, то имеет место равенство

$$\left[\begin{array}{c|c} \mathbf{A}^{(r-1)}\mathbf{M}^{(r)} + \tilde{\mathbf{a}}_-\tilde{\mathbf{b}}_-^{(r)T} & \mathbf{A}^{(r-1)}\tilde{\mathbf{b}}_+^{(r)} + \tilde{\mathbf{a}}_- b_0^{(r)} \\ \hline \tilde{\mathbf{a}}_-^T\mathbf{M}^{(r)} + a_0\tilde{\mathbf{b}}_-^{(r)T} & \tilde{\mathbf{a}}_-^T\tilde{\mathbf{b}}_+^{(r)} + a_0 b_0^{(r)} \end{array}\right] = \mathbf{I},$$

Следовательно,

$$\begin{aligned} \mathbf{A}^{(r-1)}\mathbf{M}^{(r)} + \tilde{\mathbf{a}}_-\tilde{\mathbf{b}}_-^{(r)T} &= \mathbf{I}, \\ \mathbf{A}^{(r-1)}\tilde{\mathbf{b}}_+^{(r)} + \tilde{\mathbf{a}}_- b_0^{(r)} &= \mathbf{0}, \\ \tilde{\mathbf{a}}_-^T\mathbf{M}^{(r)} + a_0\tilde{\mathbf{b}}_-^{(r)T} &= \mathbf{0}, \\ \tilde{\mathbf{a}}_-^T\tilde{\mathbf{b}}_+^{(r)} + a_0 b_0^{(r)} &= 1. \end{aligned}$$

Матрицу $\mathbf{M}^{(r)}$ можно исключить, выразив ее через обратную к $\mathbf{A}^{(r-1)}$ матрицу $\mathbf{B}^{(r-1)}$. Умножая первое уравнение на $\mathbf{B}^{(r-1)}$ слева, получаем

$$\mathbf{M}^{(r)} = \mathbf{B}^{(r-1)} - \mathbf{B}^{(r-1)}\tilde{\mathbf{a}}_+\tilde{\mathbf{b}}_-^{(r)T}.$$

Теперь утверждение теоремы следует непосредственно из второго из выписанных уравнений. □

До сих пор мы мало что предполагали о матрице $\mathbf{B}^{(r)}$, требовалось только существование матрицы $\mathbf{B}^{(r-1)}$. Теперь для построения рекурсивной процедуры вычисления матрицы $\mathbf{B}^{(r)}$ по ее полуграницe воспользуемся свойством персимметричности.

Теорема 11.2.3. *Пусть* $\mathbf{B}$ *обозначает матрицу, обратную к теплицевой матрице. Тогда* $\mathbf{B}$ *полностью определяется своей верхней полуграницей по восходящей рекурсии*

$$b_{i+1,\,j+1} = b_{ij} + \frac{1}{b_0}[\mathbf{b}_+\mathbf{b}_-^T - \tilde{\mathbf{b}}_+\tilde{\mathbf{b}}_-^T], \qquad \begin{array}{l} i = 1,\ \ldots,\ n-1, \\ j = 1,\ \ldots,\ n-1, \end{array}$$

и своей нижней полуграницей по нисходящей рекурсии

$$b_{i-1,\,j-1} = b_{ij} + \frac{1}{b_0}\,[\tilde{\mathbf{b}}_+\tilde{\mathbf{b}}_-^T - \mathbf{b}_+\mathbf{b}_-^T], \qquad \begin{matrix} i = n,\ \ldots,\ 2, \\ j = n,\ \ldots,\ 2, \end{matrix}$$

где первая и последняя строки матрицы равны соответственно $(b_0,\ \mathbf{b}_-)^T$ *и* $(\tilde{\mathbf{b}}_-,\ b_0)^T$, *а первый и последний столбцы матрицы равны соответственно* $(b_0,\ \mathbf{b}_+)$ *и* $(\tilde{\mathbf{b}}_+,\ b_0)$.

Доказательство. В теореме 11.2.2 мы уже доказали, что

$$\mathbf{B}^{(r)} = \left[\begin{array}{c|c} \mathbf{B}^{(r-1)} + \dfrac{1}{b_0^{(r)}}\tilde{\mathbf{b}}_-^{(r)}\tilde{\mathbf{b}}_-^{(r)T} & \tilde{\mathbf{b}}_-^{(r)} \\ \hline \tilde{\mathbf{b}}_-^{(r)T} & b_0^{(r)} \end{array}\right].$$

Так как матрица $\mathbf{B}^{(r)}$ является персимметричной, то можно также записать равенство

$$\mathbf{B}^{(r)} = \left[\begin{array}{c|c} b_0^{(r)} & \mathbf{b}_+^{(r)T} \\ \hline \mathbf{b}_+^{(r)} & \mathbf{B}^{(r-1)} + \dfrac{1}{b_0^{(r)}}\mathbf{b}_+^{(r)}\mathbf{b}_+^{(r)T} \end{array}\right].$$

Используя эти два разбиения матрицы $\mathbf{B}^{(r)}$, выразим ее элементы $b_{ij}^{(r)}$ двумя способами. Первый из них дает

$$b_{ij}^{(r)} = b_{ij}^{(r-1)} + \frac{1}{b_0^{(r)}}\,(\tilde{\mathbf{b}}_+^{(r)}\tilde{\mathbf{b}}_-^{(r)\,T})_{ij}, \qquad \begin{matrix} i = 1,\ \ldots,\ n-1, \\ j = 1,\ \ldots,\ n-1. \end{matrix}$$

Второй дает

$$b_{i+1,\,j+1}^{(r)} = b_{ij}^{(r-1)} + \frac{1}{b_0^{(r)}}\,(\mathbf{b}_+^{(r)}\mathbf{b}_-^{(r)\,T})_{ij}, \qquad \begin{matrix} i = 1,\ \ldots,\ n-1, \\ j = 1,\ \ldots,\ n-1. \end{matrix}$$

Исключая элементы $b_{ij}^{(r-1)}$, приходим к равенствам

$$b_{i+1,\,j+1}^{(r)} = b_{ij}^{(r)} + \frac{1}{b_0^{(r)}}\,(\mathbf{b}_+^{(r)}\mathbf{b}_-^{(r)\,T} - \tilde{\mathbf{b}}_+^{(r)}\tilde{\mathbf{b}}_-^{(r)\,T})_{ij}, \qquad \begin{matrix} i = 1,\ \ldots,\ n-1, \\ j = 1,\ \ldots,\ n-1, \end{matrix}$$

которые теперь содержат только помеченные индексом r величины. Эти равенства задают восходящую рекурсию. Аналогично строится нисходящая рекурсия, и это заканчивает доказательство. □

Доказательство содержит еще один информативный фрагмент, который будет ниже использован. Выражая стоящий в первой строке и первом столбце элемент из первого равенства, получаем уравнение

$$b_0^{(r)} = b_{11}^{(r-1)} + \frac{1}{b_0^{(r)}}\,(\tilde{\mathbf{b}}_+^{(r)}\tilde{\mathbf{b}}_-^{(r)\,T})_{11}.$$

Так как $\tilde{\mathbf{b}}_+\tilde{\mathbf{b}}_-^T = \mathbf{b}_+\mathbf{b}_-^T$ и согласно второму равенству $b_{11}^{(r-1)} = b_0^{(r-1)}$, то оно может быть переписано в виде

$$b_0^{(r)} = b_0^{(r-1)} + \frac{1}{b_0^{(r)}}\,(\mathbf{b}_+^{(r)}\mathbf{b}_-^{(r)\,T})_{11},$$

связывающем величины $b_0^{(r)}$ и $b_0^{(r-1)}$.

Теперь нам нужно построить алгоритм, вычисляющий полуграницу матрицы $\mathbf{B}^{(n)}$ по полуграница матрицы $\mathbf{A}^{(n)}$. Но у нас уже выписаны уравнения, связывающие границу матрицы $\mathbf{B}^{(r)}$ с границей матрицы $\mathbf{A}^{(r)}$, так что для построения этого алгоритма нам остается только исключить матрицы $\mathbf{A}^{(r-1)}$ и $\mathbf{M}$. Такое исключение можно выполнить, вводя обратно полуграницу предыдущего шага итераций.

Теорема 11.2.4. *Полуграница матрицы* $\mathbf{B}^{(r)}$ *удовлетворяет следующим рекуррентным уравнениям*:

$$\text{(i)}\ \mathbf{b}_-^{(r)} = \frac{b_0^{(r)}}{b_0^{(r-1)}}\left\{\begin{bmatrix}\mathbf{b}_-^{(r-1)} \\ 0\end{bmatrix} - (\tilde{\mathbf{b}}_-^{(r-1)T}\mathbf{a}_-^{(r-1)} + b_0^{(r-1)}a_{r-1})\begin{bmatrix}\tilde{\mathbf{b}}_-^{(r-1)} \\ b_0^{(r-1)}\end{bmatrix}\right\},$$

$$\text{(ii)}\ \mathbf{b}_+^{(r)} = \frac{b_0^{(r)}}{b_0^{(r-1)}}\left\{\begin{bmatrix}\mathbf{b}_+^{(r-1)} \\ 0\end{bmatrix} - (\tilde{\mathbf{b}}_+^{(r-1)T}\mathbf{a}_+^{(r-1)} + b_0^{(r-1)}a_{r+1})\begin{bmatrix}\tilde{\mathbf{b}}_+^{(r-1)} \\ b_0^{(r-1)}\end{bmatrix}\right\},$$

$$\text{(iii)}\ b_0^{(r)} = b_0^{(r-1)} + \frac{1}{b_0^{(r)}}\,(\mathbf{b}_-^{(r)}\mathbf{b}_+^{(r)})_{11}.$$

Доказательство. Третье уравнение уже было выведено как следствие из теоремы 11.2.3. Из соображений симметрии ясно, что если выполняется первое равенство, то выполняется и второе. Его можно получить тем же самым способом. Следовательно, надо доказать только первое равенство. Начнем со второго в множестве полученных в доказательстве теоремы 11.2.2 четырех равенств:

$$\mathbf{A}^{(r-1)}\tilde{\mathbf{b}}_+^{(r)} + \tilde{\mathbf{a}}_+^{(r)}b_0^{(r)} = \mathbf{0}.$$

Так как матрица $\mathbf{A}^{(r-1)}$ является персимметричной, то это равенство можно также переписать в виде

$$\mathbf{A}^{(r-1)\,T}\mathbf{b}_+^{(r)} + \mathbf{a}_+^{(r)}b_0^{(r)} = \mathbf{0},$$

откуда

$$\mathbf{b}_+^{(r)} = -\,b_0^{(r)}\mathbf{B}^{(r-1)\,T}\mathbf{a}_+^{(r)}.$$

Теперь воспользуемся снова уже применявшимся ранее разбиением матрицы $\mathbf{B}^{(r-1)}$:

$$\mathbf{b}_-^{(r)} = -\,b_0^{(r)}\left[\begin{array}{c|c} \mathbf{B}^{(r-2)} + \dfrac{1}{b_0^{(r-1)}}\tilde{\mathbf{b}}_-^{(r-1)}\tilde{\mathbf{b}}_-^{(r-1)T} & \tilde{\mathbf{b}}_-^{(r-1)} \\ \hline \tilde{\mathbf{b}}_-^{(r-1)T} & b_0^{(r-1)} \end{array}\right]^T \begin{bmatrix}\mathbf{a}_-^{(r-1)} \\ \hline a_{r-1}\end{bmatrix},$$

где $\mathbf{a}_+^{(r)}$ записан в виде вектора $\mathbf{a}_+^{(r-1)}$ с одной дополнительной компонентой. Это уравнение уже определяет искомую рекурсию,

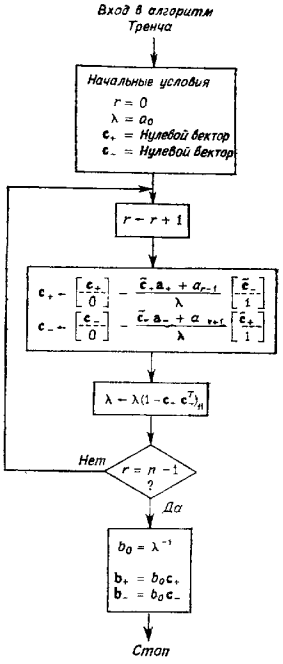


Рис. 11.3. Алгоритм Тренча.

но мы уточним его, воспользовавшись равенством $\mathbf{b}_+^{(r-1)} = -b_0^{(r-1)}\mathbf{B}^{(r-2)T}\mathbf{a}_+^{(r-1)}$. Тогда

$$\mathbf{b}_+^{(r)} = \frac{b_0^{(r)}}{b_0^{(r-1)}}\,\mathbf{b}_+^{(r-1)} - b_0^{(r)}\begin{bmatrix} \dfrac{1}{b_0^{(r-1)}}\,\tilde{\mathbf{b}}_-^{(r-1)}\tilde{\mathbf{b}}_+^{(r-1)T} & \tilde{\mathbf{b}}_-^{(r-1)} \\ \tilde{\mathbf{b}}_+^{(r-1)T} & b_0^{(r-1)} \end{bmatrix}\begin{bmatrix} \mathbf{a}_+^{(r-1)} \\ a_{r-1} \end{bmatrix},$$

что можно переписать в виде, входящем в формулировку теоремы. □

Теперь у нас уже есть все необходимое для формулировки алгоритма Тренча. Все, что осталось сделать, сводится к такому переопределению обозначений, при котором все величины, индексируемые символом r, стоят в левых частях рекуррентных уравнений, а величины, индексируемые символом $r-1$, стоят в правой части.

Алгоритм Тренча подытожен на рис. 11.3. Блок-схема основана на теореме 11.2.4, но величины $\mathbf{b}_+$, $\mathbf{b}_-$ и b_0 в итерациях заменены нормализованными величинами $\mathbf{c}_+$, $\mathbf{c}_-$ и λ в соответствии с равенствами

$$\mathbf{c}_+^{(r)} = \frac{1}{b_0^{(r)}}\mathbf{b}_+^{(r)}, \quad \mathbf{c}_-^{(r)} = \frac{1}{b_0^{(r)}}\mathbf{b}_-^{(r)} \text{ и } \lambda^{(r)} = \frac{1}{b_0^{(r)}}.$$

Соответственно рекурсия описывается уравнениями

$$\mathbf{c}_+^{(r)} = \begin{bmatrix} \mathbf{c}_+^{(r-1)} \\ 0 \end{bmatrix} - \frac{\tilde{\mathbf{c}}_-^{(r-1)T}\mathbf{a}_+^{(r-1)} + a_{r-1}}{\lambda^{(r-1)}}\begin{bmatrix} \tilde{\mathbf{c}}_-^{(r-1)} \\ 1 \end{bmatrix},$$

$$\mathbf{c}_-^{(r)} = \begin{bmatrix} \mathbf{c}_-^{(r-1)} \\ 0 \end{bmatrix} - \frac{\tilde{\mathbf{c}}_+^{(r-1)T}\mathbf{a}_-^{(r-1)} + a_{n+r-1}}{\lambda^{(r-1)}}\begin{bmatrix} \tilde{\mathbf{c}}_+^{(r-1)} \\ 1 \end{bmatrix},$$

$$\lambda^{(r)} = \lambda^{(r-1)}(1 - \mathbf{c}_-^{(r)}\mathbf{c}_+^{(r)})_{11}.$$

На рис. 11.3 использована эта форма уравнений.

11.3. Алгоритм Берлекэмпа—Месси

Алгоритм Берлекэмпа—Месси представляет собой алгоритм решения в произвольном поле F теплицевой системы уравнений вида [1])

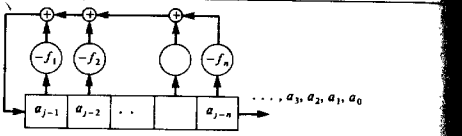
$$\begin{bmatrix} a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \\ a_n & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_{n+1} & a_n & a_{n-1} & \dots & a_2 \\ \vdots & & & & \vdots \\ a_{2n-2} & a_{2n-3} & a_{2n-4} & \dots & a_{n-1} \end{bmatrix}\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} -a_n \\ -a_{n+1} \\ \\ \vdots \\ -a_{2n-1} \end{bmatrix}.$$

Матрица системы не является симметрической, что требуется для применимости алгоритма Левинсона. С другой стороны, стоящий в правой части системы вектор не является произвольным, а составлен из компонент матрицы системы.

Лучшим подходом к алгоритму Берлекэмпа—Месси является интерпретация выписанного матричного уравнения как описания авторегрессионного фильтра. Предположим, что вектор $\mathbf{f}$ известен. Тогда первая строка рассматриваемого матричного уравнения определяет a_n через $a_0, a_1, \dots, a_{n-1}$, вторая строка определяет a_{n+1} через $a_1, \dots, a_n$, и т. д. Этот последовательный процесс описывается уравнением

$$a_j = -\sum_{i=1}^{n} f_i a_{j-i}, \quad j = n, \dots, 2n-1.$$

[1]) В настоящем разделе более удобно нумеровать компоненты вектора $\mathbf{f}$ числами от 1 до n и обозначать элементы матрицы $\mathbf{A}$ величинами $a_0, a_1, \dots, a_{2n-1}$.



Подставить начальные значения a_{n-1}, a_{n-2}, ..., a_0 в выражение

$$a_j = -\sum_{i=1}^{n} f_i a_{j-i}, \quad j = n, \ldots, 2n-1.$$

Рис. 11.4. Авторегрессионный фильтр.

При фиксированном $\mathbf{f}$ это уравнение является уравнением авторегрессионного фильтра, который может быть реализован в виде линейного регистра сдвига с обратной связью, в отводах которого стоят умножители на компоненты вектора $\mathbf{f}$.

С этой точки зрения задача решения данной теплицевой системы уравнений становится задачей построения показанного на рис. 11.4 авторегрессионного фильтра, на выходе которого формируется заданная последовательность символов. Если теплицева матрица обратима, то существует точно один авторегрессионный фильтр, удовлетворяющий данной системе уравнений. Если матрица необратима, то таких решений может существовать много или не существовать ни одного. Однако, алгоритм Берлекэмпа—Месси всегда в качестве решения дает кратчайший авторегрессионный фильтр, удовлетворяющий данным уравнениям. Отводы этого фильтра описываются вектором $\mathbf{f}$, содержащим не более n ненулевых компонент, если исходная теплицева система имеет по меньшей мере одно решение, и большее, чем n, число ненулевых компонент f_i, $i > n$, если исходная теплицева система не имеет решений.

Любая процедура построения авторегрессионного фильтра является также методом решения рассматриваемого матричного уравнения с заданным вектором $\mathbf{f}$. Мы сейчас опишем такую процедуру построения регистров сдвига. В процедуре не предполагается наличие каких бы то ни было специальных свойств последовательности $a_0, a_1, \ldots, a_{2n-1}$. Произвольный линейный регистр сдвига с обратной связью можно задать многочленом $f(x)$ обратных связей,

$$f(x) = f_n x^n + f_{n-1} x^{n-1} + \cdots + f_1 x + 1,$$

и длиной L регистра сдвига. Длина регистра сдвига может быть больше степени многочлена $f(x)$, так как самые правые ячейки регистра могут не включаться в обратную связь.

Для построения регистра сдвига надо определить две величины: длину L регистра и многочлен $f(x)$ обратных связей степени $\deg f(x) \leqslant L$. Обозначим эту пару через $(L, f(x))$. Надо найти

регистр сдвига с обратной связью, который при соответствующих начальных условиях порождает заданную последовательность $a_0, \ldots, a_{2n-1}$ и является при этом кратчайшим.

Рассматриваемая процедура построения является рекурсивной. Для каждого r, начиная с $r = 1$, мы будем строить регистр сдвига, порождающий последовательность $a_0, \ldots, a_r$. Регистр сдвига минимальной длины, порождающий последовательность $a_0, \ldots, a_r$, обозначим через $(L_r, f^{(r)}(x))$. Этот регистр не обязательно должен определяться однозначно; возможно существование нескольких таких регистров с одной и той же длиной. К началу r-го шага имеется список регистров сдвига

$$(L_1, f^{(1)}(x)), \quad (L_2, f^{(2)}(x)), \quad \ldots, \quad (L_{r-1}, f^{(r-1)}(x)).$$

Алгоритм Берлекэмпа—Месси вычисляет новый кратчайший регистр $(L_r, f^{(r)}(x))$, генерирующий последовательность $a_0, \ldots, a_r$. Для этого используется самый последний из вычисленных регистров, в котором по мере надобности модифицируются длина и весовые множители в отводах.

На r-м шаге вычисляется следующий элемент на выходе $(r-1)$-го регистра сдвига:

$$\hat{a}_r = -\sum_{j=1}^{L_{r-1}} f_j^{(r-1)} a_{r-j}.$$

Так как L_{r-1} может быть больше степени многочлена $f^{(r-1)}(x)$, то некоторые члены суммы могут быть равными нулю. Эту сумму следовало бы записать с пределами суммирования от 1 до $\deg f^{(r-1)}(x)$. Мы, однако, выбрали менее громоздкое обозначение.

Пусть Δ_r обозначает разность между требуемым элементом на выходе a_r и истинным элементом, полученным на выходе самого последнего регистра сдвига:

$$\Delta_r = a_r - \hat{a}_r = a_r + \sum_{j=1}^{L_{r-1}} f_j^{(r-1)} a_{r-j}.$$

Эквивалентно,

$$\Delta_r = \sum_{j=0}^{L_{r-1}} f_j^{(r-1)} a_{r-j}.$$

Если $\Delta_r = 0$, то полагаем $(L_r, f^{(r)}(x)) = (L_{r-1}, f^{(r-1)}(x))$ и завершаем этим r-ю итерацию. В противном случае изменим весовые множители в цепи обратной связи регистра сдвига по правилу:

$$f^{(r)}(x) = f^{(r-1)}(x) + A x^l f^{(m-1)}(x),$$

где A — элемент поля, l — целое число и $f^{(m-1)}(x)$ — один из многочленов обратной связи, встречавшийся ранее в списке регистров сдвига.

Используя этот новый многочлен, определим

$$\Delta_r' = \sum_{l=0}^{L_{r-1}} f_l^{(r)} a_{r-l} = \sum_{l=0}^{L_{r-1}} f_l^{(r-1)} a_{r-l} + A \sum_{l=0}^{L_{r-1}} f_l^{(m-1)} a_{r-l-l}.$$

Теперь все готово для определения величин m, l и A. Выбе рем m меньше r и такое, что $\Delta_m \neq 0$; выберем также $l = r - m$ и $A = -\Delta_m^{-1} \Delta_r$. Тогда

$$\Delta_r' = \Delta_r - \frac{\Delta_r}{\Delta_m} \Delta_m = 0,$$

так что новый регистр сдвига будет генерировать последователь ность $a_1, \ldots, a_{r-1}, a_r$. У нас остался произвол в выборе m, дл которого $\Delta_m \neq 0$. Выбрав в качестве m номер ближайшей итера ции, для которой выполнялось условие $L_m > L_{m-1}$, получим каждой итерации регистр сдвига минимальной длины, однако та кое усовершенствование требует дополнительных рассуждений и будет описано позже.

Практическая интерпретация того, что было изложено до на стоящего момента, представлена на рис. 11.5. Два регистра сдвига, отвечающие итерациям с номерами m и r, показаны на фрагменте(i) рисунка. В качестве m-й итерации выбирается та, при которой ре гистр сдвига $(L_{m-1}, f^{(m-1)}(x))$ не способен генерировать компо ненту a_m, и минимальная длина регистра сдвига, генерирующего требуемую компоненту при m-й итерации, больше L_{m-1}. Будем также считать, что регистр сдвига $(L_{r-1}, f^{(r-1)}(x))$ не способен ге нерировать компоненту a_r, так как в противном случае мы не дол жны ничего с ним делать.

На фрагменте (ii) рис. 11.5 регистр сдвига $(L_{m-1}, f^{(m-1)}(x))$ превращен во вспомогательный регистр путем увеличения его дли ны, позиционирования и такого изменения его выхода, которое позволяет скомпенсировать неспособность регистра $(L_{r-1}, f^{(r-1)}(x))$ генерировать компоненту a_r. Отметим, что вспомогательный ре гистр имеет дополнительный отвод с весовым множителем еди ница, соответствующим коэффициенту $f_0^{(m-1)}$. На протяжении пер вых $r-1$ итераций в остальных отводах цепи обратной связи формируется отрицательное значение соответствующей этому до полнительному отводу величины, и поэтому на выходе вспомога тельного регистра формируется последовательность нулей. При r-й итерации эти величины взаимно не уничтожаются, и выход вспомогательного регистра оказывается ненулевым. Коэффициент A выбирается таким образом, чтобы его добавление к r-му выход ному сигналу обратной связи главного регистра сдвига приво дило бы к получению нужной компоненты a_r.

На фрагменте (iii) рис. 11.5 показано, как два регистра сдвига, изображенные на фрагменте (ii), сливаются в один регистр, что в силу принципа суперпозиции не меняет общего поведения схемы.

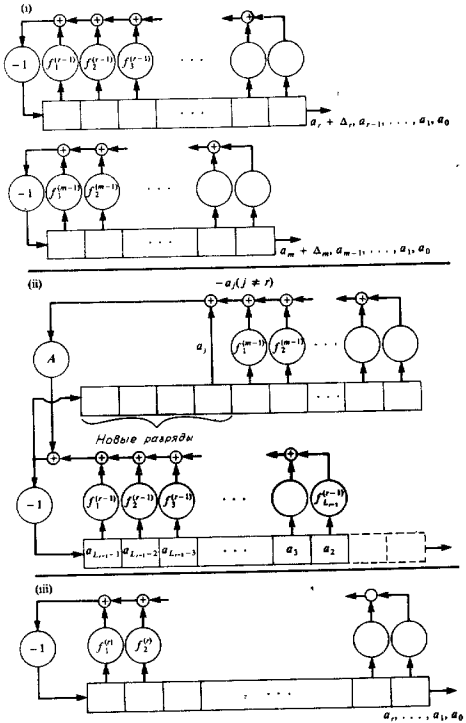


Рис. 11.5. Конструкция Берлекэмпа—Месси.

Это дает регистр $(L_r, f^{(r)}(x))$. Иногда $L_r = L_{r-1}$, иногда $L_r > L_{r-1}$. В последнем случае при проведении дальнейших итераций m заменяется на r.

Точное описание процедуры дается теоремой 11.3.2, которая утверждает, что эта процедура приводит к наименьшему регистру

сдвига с требуемым свойством. Прежде, чем доказывать эту теорему, получим границу для длины регистра сдвига.

Теорема 11.3.1. *Пусть* $(L_{r-1}, f^{(r-1)}(x))$ *— регистр сдвига с линейной обратной связью минимальной длины, генерирующий последовательность* $a_1, \ldots, a_{r-1}$, *а* $(L_r, f^{(r)}(x))$ *— регистр сдвига с линейной обратной связью минимальной длины, генерирующий последовательность* $a_1, \ldots, a_{r-1}, a_r$, *и* $f^{(r)}(x) \neq f^{(r-1)}(x)$. *Тогда* $L_1 \geqslant \max[L_{r-1}, r - L_{r-1}]$.

Доказательство. Неравенство, которое требуется доказать, разбивается на два неравенства

$$L_r \geqslant L_{r-1} \text{ и } L_r \geqslant r - L_{r-1}.$$

Первое неравенство очевидно, поскольку если регистр сдвига с линейной обратной связью генерирует некоторую последовательность, то он генерирует и любую меньшую последовательность, являющуюся началом исходной. Если $L_{r-1} \geqslant r$, то второе неравенство также становится очевидным. Поэтому предположим, что $L_{r-1} < r$ и второе неравенство не выполняется, и попытаемся прийти к противоречию. Из предположения следует, что $L_r \leqslant r - 1 - L_{r-1}$. Для сокращения записи введем временные обозначения: $c(x) = f^{(r-1)}(x)$, $b(x) = f^{(r)}(x)$, $L = L_{r-1}$ и $L' = L_r$. В новых обозначениях исходные предположения принимают вид: $L < r$ и $r \geqslant L + L' + 1$. Кроме того, из условий теоремы следует, что

$$a_r \neq -\sum_{i=1}^{L} c_i a_{r-i},$$

$$a_j = -\sum_{i=1}^{L} c_i a_{j-i}, \quad j = L+1, \ \ldots, \ r-1,$$

и

$$a_j = -\sum_{k=1}^{L'} b_k a_{j-k}, \quad j = L'+1, \ \ldots, \ r.$$

Теперь установим противоречие. С одной стороны,

$$a_r = -\sum_{k=1}^{L'} b_k a_{r-k} = \sum_{k=1}^{L'} b_k \sum_{i=1}^{L} c_i a_{r-k-i},$$

где справедливость разложения a_{r-k} следует из того, что $r - k$ пробегает целые значения от $r - 1$ до $r - L'$, содержащиеся среди чисел $L + 1, \ldots, r - 1$, в силу предположения $r \geqslant L + L' + 1$. С другой стороны,

$$a_r \neq -\sum_{i=1}^{L} c_i a_{r-i} = \sum_{i=1}^{L} c_i \sum_{k>1}^{L'} b_k a_{r-i-k},$$

где справедливость разложения a_{r-i} следует из того, что $r - i$ пробегает содержащиеся среди чисел $L' + 1, \ldots, r - 1$ целые значения от $r - 1$ до $r - L$, а это, в свою очередь, следует из предположения $r \geqslant L + L' + 1$. Изменение порядка суммирования в правой части последнего равенства приводит к выражению для a_r, полученному в правой части предыдущего равенства. Таким образом, получаем доказывающее теорему противоречие $a_r \neq a_r$. □

Если удастся построить регистр сдвига с длиной, удовлетворяющей соотношению из теоремы 11.3.1 с заменой знака нестрогого неравенства знаком равенства, то тем самым будет построен регистр сдвига минимальной длины. Доказательство теоремы 11.3.2 предъявляет конструкцию такого регистра сдвига.

Теорема 11.3.2. *Предположим, что* $(L_i, f^{(i)}(x))$, $i = 1, \ldots, r$, *является последовательностью регистров сдвига минимальной длины с линейной обратной связью, таких, что* $(L_i, f^{(i)}(x))$ *генерирует последовательность* $a_1, \ldots, a_i$. *Если* $f^{(r)}(x) \neq f^{(r-1)}(x)$, *то*

$$L_r = \max[L_{r-1}, \ r - L_{r-1}]$$

и любой регистр сдвига, генерирующий $a_1, \ldots, a_r$ *и имеющий длину, совпадающую с величиной правой части последнего равенства, является регистром сдвига минимальной длины.*

Доказательство. Согласно теореме 11.3.1 L_r не может быть меньше величины в правой части последнего равенства. Если удастся построить какой-либо регистр сдвига, который генерирует требуемую последовательность и длина которого совпадает с указанной величиной, то он будет регистром сдвига минимальной длины. Доказательство будет проводиться по индукции. Мы построим удовлетворяющий теореме регистр сдвига в предположении, что такие регистры последовательно построены для всех $k \leqslant r - 1$. Для каждого k, $k = 1, \ldots, r - 1$, обозначим через $(L_k, f^{(k)}(x))$ регистр сдвига минимальной длины, генерирующий $a_1, \ldots, a_k$. Примем за предположение индукции, что

$$L_k = \max[L_{k-1}, \ k - L_{k-1}], \quad k = 1, \ \ldots, \ r-1,$$

каждый раз, когда $f^{(k)}(x) \neq f^{(k-1)}(x)$. Это, очевидно, верно для $k = 0$, если a_1 отличен от нуля, поскольку $L_0 = 0$ и $L_1 = 1$. В более общем случае, если a_i — первый ненулевой элемент в заданной последовательности, то $L_{i-1} = 0$ и $L_i = i$. Тогда предположение индукции относится к индексу $k = i$.

Значение k в последней итерации, приведшей к изменению длины, будем обозначать через m. Последнее означает, что при завершении $(r - 1)$-й итерации m является целым числом, таким, что

$$L_{r-1} = L_m > L_{m-1}.$$

Теперь имеет место следующее равенство:

$$a_l + \sum_{i=1}^{L_{r-1}} f_i^{(r-1)} a_{l-i} = \sum_{i=0}^{L_{r-1}} f_i^{(r-1)} a_{l-i} = \begin{cases} 0, & j = L_{r-1}, \ldots, r-1, \\ \Delta_r, & j = r. \end{cases}$$

Если $\Delta_r = 0$, то регистр $(L_{r-1}, f^{(r-1)}(x))$ также генерирует первые r символов, и, таким образом,

$$L_r = L_{r-1} \text{ и } f^{(r)}(x) = f^{(r-1)}(x).$$

Если $\Delta_r \neq 0$, то необходимо построить новый регистр сдвига. Напомним, что изменение длины регистра сдвига произошло при $k = m$. Следовательно,

$$a_l + \sum_{i=1}^{L_{m-1}} f_i^{(m-1)} a_{l-i} = \begin{cases} 0, & j = L_{m-1}, \ldots, m-1, \\ \Delta_m \neq 0, & j = m, \end{cases}$$

и по предложению индукции

$$L_{r-1} = L_m = \max[L_{m-1}, m - L_{m-1}] = m - L_{m-1},$$

поскольку $L_m > L_{m-1}$. Теперь выберем новый многочлен

$$f^{(r)}(x) = f^{(r-1)}(x) - \Delta_r \Delta_m^{-1} x^{r-m} f^{(m-1)}(x)$$

и положим $L_r = \deg f^{(r)}(x)$. В этом случае поскольку $\deg f^{(r-1)}(x) \leqslant L_{r-1}$ и $\deg [x^{r-m} f^{(m-1)}(x)] \leqslant r - m + L_{m-1}$, то

$$L_r \leqslant \max[L_{r-1}, r - m + L_{m-1}] \leqslant \max[L_{r-1}, r - L_{r-1}].$$

Из теоремы 11.3.1 при условии, что $(L_r, f^{(r)}(x))$ генерирует $a_1, \ldots, a_r$, следует, что $L_r = \max[L_{r-1}, r - L_{r-1}]$. Осталось только доказать, что регистр сдвига $(L_r, f^{(r)}(x))$ генерирует требуемую последовательность. Докажем это непосредственным вычислением разности между a_j и сигналом на выходе обратной связи регистра сдвига:

$$a_l - \left(- \sum_{i=1}^{L_r} f_i^{(r)} a_{l-i} \right) = a_l + \sum_{i=1}^{L_{r-1}} f_i^{(r-1)} a_{l-i} -$$

$$- \Delta_r \Delta_m^{-1} \left[a_{l-r+m} + \sum_{i=1}^{L_{m-1}} f_i^{(m-1)} a_{l-r+m-i} \right] =$$

$$= \begin{cases} 0 & j = L_r, L_r+1, \ldots, r-1, \\ \Delta_r - \Delta_r \Delta_m^{-1} \Delta_m = 0, & j = r. \end{cases}$$

Итак, регистр $(L_r, f^{(r)}(x))$ генерирует $a_1, \ldots, a_r$. В частности, $(L_n, f^{(n)}(x))$ генерирует $a_1, \ldots, a_n$ и теорема доказана. □

Теорема 11.3.3 (Алгоритм Берлекэмпа — Месси). *Пусть заданы $a_1, \ldots, a_n$ из некоторого поля, и пусть при начальных условиях $f^{(0)}(x) = 1$, $t^{(0)}(x) = 1$ и $L_0 = 0$ выполняются следующие*

рекуррентные равенства, используемые для вычисления $f^{(n)}(x)$:

$$\Delta_r = \sum_{j=0}^{n-1} f_j^{(r-1)} a_{r-j},$$

$$L_r = \delta_r (r - L_{r-1}) + (1 - \delta_r) L_r,$$

$$\begin{bmatrix} f^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1} \delta_r & (1 - \delta_r) x \end{bmatrix} \begin{bmatrix} f^{(r-1)}(x) \\ t^{(r-1)}(x) \end{bmatrix},$$

$r = 1, \ldots, 2n$, где $\delta_r = 1$, если одновременно $\Delta_r \neq 0$ и $2L_{r-1} \leqslant r - 1$, и $\delta_r = 0$ в противном случае. Тогда $f^{(2r)}(x)$ является многочленом наименьшей степени, коэффициенты которого удовлетворяют равенствам $f_0^{(2r)} = 1$ и

$$a_r + \sum_{j=1}^{n-1} f_j^{(2r)} a_{r-j} = 0,$$

$$r = L_{2r} + 1, \ldots, 2r.$$

Доказательство. Следует из теоремы 11.3.2. □

В этой теореме Δ_r может обращаться в нуль, но только в том случае, когда $\delta_r = 0$. Положим тогда по определению $\Delta_r^{-1} \delta_r = 0$.

Блок-схема алгоритма Берлекэмпа—Месси приведена на рис. 11.6. На r-м шаге указанный алгоритм содержит число умножений, равное примерно удвоенной степени многочлена $f^{(r)}(x)$. Степень многочлена $f^{(r)}(x)$ равна примерно $r/2$, и всего имеется $2n$ итераций, так что всего алгоритм содержит примерно $2n^2 = \sum_{r=0}^{2n} r$ умножений и примерно такое же число сложений. Короче можно сказать, что порядок числа умножений в алгоритме Берлекэмпа—Месси равен n^2, или, формально, $O(n^2)$.

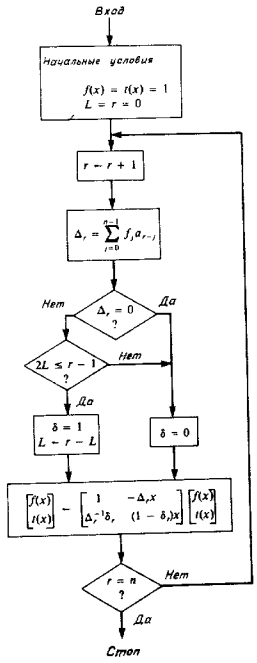


Рис. 11.6. Алгоритм Берлекэмпа—Месси.

11.4. Рекурсивный алгоритм Берлекэмпа—Месси

Все рассмотренные нами до сих пор методы решения теплицевых систем уравнений содержат число умножений, пропорциональное n^2. В настоящем параграфе мы воспользуемся стратегией дублирования для того, чтобы снизить вычислительную сложность алгоритма Берлекэмпа — Месси при больших n.

Число используемых на r-м шаге итераций умножений примерно равно удвоенной степени многочлена $f^{(r)}(x)$. На первых шагах итераций эта степень мала, и итерации выполняются легко, но с ростом r растет и сложность вычислений. В ускоренном алгоритме за счет одновременного выполнения нескольких итераций используется преимущество, которое дается простотой вычислений на первых шагах. После выполнения каждой такой группы итераций исходная задача модифицируется так, чтобы учесть это полученное решение. Затем сызнова начинается выполнение новой группы итераций алгоритма Берлекэмпа—Месси, но уже для модифицированной задачи и модифицированного начального значения многочлена $f(x)$.

Рассматриваемое ниже построение начинается с более компактной организации алгоритма Берлекэмпа—Месси. Заменим многочлены $f^{(r)}(x)$ и $t^{(r)}(x)$ полиномиальной матрицей размера 2×2:

$$\mathbf{F}^{(r)}(x) = \begin{bmatrix} F_{11}^{(r)}(x) & F_{12}^{(r)}(x) \\ F_{21}^{(r)}(x) & F_{22}^{(r)}(x) \end{bmatrix}.$$

Коэффициенты многочлена $F_{ij}^{(r)}(x)$ будем обозначать через $F_{ij,k}^{(r)}$. Матрица $\mathbf{F}^{(r)}(x)$ определяется так, чтобы многочлены $f^{(r)}(x)$ и $t^{(r)}(x)$ можно было вычислить, используя уравнения

$$\begin{bmatrix} f^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \begin{bmatrix} F_{11}^{(r)}(x) + F_{12}^{(r)}(x) \\ F_{21}^{(r)}(x) + F_{22}^{(r)}(x) \end{bmatrix} = \mathbf{F}^{(r)}(x) \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Напомним, что сначала выполняемые в алгоритме Берлекэмпа—Месси вычисления записываются в виде двух уравнений:

$$\Delta_r = \sum_{j=0}^{n-1} f_j^{(r-1)} a_{r-j},$$

$$\begin{bmatrix} f^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \begin{bmatrix} f^{(r-1)}(x) \\ t^{(r-1)}(x) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \cdots \begin{bmatrix} 1 & -\Delta_1 x \\ \Delta_1^{-1}\delta_1 & (1-\delta_1)x \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Следовательно,

$$\mathbf{F}^{(r)}(x) = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \cdots \begin{bmatrix} 1 & -\Delta_1 x \\ \Delta_1^{-1}\delta_1 & (1-\delta_1)x \end{bmatrix}.$$

Алгоритм модифицирует как матрицу $\mathbf{F}^{(r)}(x)$, так и многочлены $f^{(r)}(x)$ и $t^{(r)}(x)$. Прямой метод перевычисления $\mathbf{F}^{(r)}(x)$ требует примерно в два раза больше умножений, поскольку матрица содержит не два элемента, а четыре. Хотя в такой форме вычисления и допускают использование метода дублирования, но мы получаем большее, чем хотим, число умножений. Надо соответствующим образом реорганизовать вычисления.

Рекурсивная форма алгоритма Берлекэмпа—Месси строится вокруг следующих уравнений, эквивалентных выписанным ранее:

$$\Delta_r = \sum_{j=0}^{n-1} F_{11,j}^{(r-1)} a_{r-j} + \sum_{j=0}^{n-1} F_{12,j}^{(r-1)} a_{r-j},$$

$$\mathbf{F}^{(r)}(x) = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \mathbf{F}^{(r-1)}(x).$$

Предполагая n четным, разобьем алгоритм пополам и положим

$$\mathbf{F}^{(n)}(x) = \mathbf{F}'^{(n)}(x)\,\mathbf{F}''^{(n/2)}(x),$$

где

$$\mathbf{F}'^{(n)}(x) = \prod_{r=n}^{n/2+1} \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix},$$

$$\mathbf{F}''^{(n/2)}(x) = \prod_{r=n/2}^{1} \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix}.$$

Мы будем вычислять каждую из половин отдельно, а затем их перемножать. Сложность при этом получается меньше, чем в исходном способе решения. Необходимо также реорганизовать уравнение, определяющее Δ_r. Будем величину Δ_r вычислять как r-й коэффициент первой компоненты двумерного вектора многочленов

$$\begin{bmatrix} \Delta(x) \\ \Delta'(x) \end{bmatrix} = \begin{bmatrix} F_{11}^{(r-1)}(x) & F_{12}^{(r-1)}(x) \\ F_{21}^{(r-1)}(x) & F_{22}^{(r-1)}(x) \end{bmatrix} \begin{bmatrix} a(x) \\ a(x) \end{bmatrix}.$$

Таким образом, для r, больших чем $n/2$,

$$\begin{bmatrix} \Delta(x) \\ \Delta'(x) \end{bmatrix} = \mathbf{F}'^{(r-1)}(x)\,\mathbf{F}''^{(n/2)}(x) \begin{bmatrix} a(x) \\ a(x) \end{bmatrix} = \mathbf{F}'^{(r-1)}(x)\,\mathbf{a}^{(n/2)}(x),$$

где

$$\mathbf{a}^{(n/2)}(x) = \mathbf{F}''^{(n/2)}(x) \begin{bmatrix} a(x) \\ a(x) \end{bmatrix}.$$

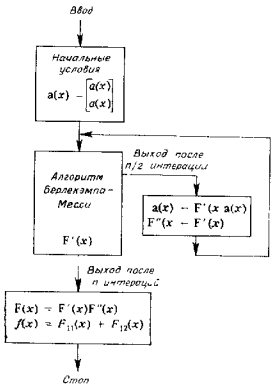


Рис. 11.7. Разложение алгоритма Берлекэмпа—Месси.

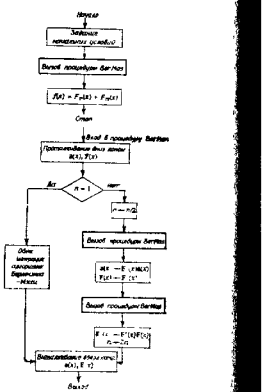


Рис. 11.8. Рекурсивный алгоритм Берлекэмпа—Месси.

Это завершает описание разбиения алгоритма Берлекэмпа—Месси. Основная форма алгоритма теперь записывается в виде

$$\Delta_r = \sum_{j=0}^{n-1} F_{11,\,j}^{(r-1)} a_{1,\,r-j} + \sum_{j=0}^{n-1} F_{12,\,j}^{(r-1)} a_{2,\,r-j},$$

$$\mathbf{F}^{(r)}(x) = \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1}\delta_r & (1-\delta_r)x \end{bmatrix} \mathbf{F}^{(r-1)}(x),$$

где вместо $\mathbf{F}^{(r)}(x)$ подставляется $\mathbf{F}'^{(r)}(x)$ или $\mathbf{F}''^{(r)}(x)$, а вектор $(a_1(x), a_2(x))$ в первой половине вычислений равен $(a(x), a(x))$, а во второй половине вычислений модифицируется в вектор $\mathbf{a}^{n/2}(x)$. После того, как вычислены обе половины задачи, матрица $\mathbf{F}^{(n)}(x)$ получается перемножением своих двух половин.

Разбиение алгоритма Берлекэмпа—Месси показано на рис. 11.7. Заметим, что каждая из половин алгоритма сама является алгоритмом Берлекэмпа—Месси. Следовательно, если $n/2$ четно, то каждая из половин может быть разбита в свою очередь; если n равно степени двух, то такое разбиение можно продолжить до тех пор, пока не будут получены фрагменты, состоящие только из одной итерации. В этих фрагментах выполняются уже все вычисления, но они достаточно тривиальны.

На рис. 11.8 алгоритма приведена рекурсивная форма Берлекэмпа—Месси. Вся вычислительная работа сводится к вычислению полиномиальных произведений, сочетающих половинные результаты. Эти вычисления представляют собой свертки многочленов и могут быть выполнены с помощью любого алгоритма линейной свертки.

11.5. Методы, основанные на алгоритме Евклида

В некоторых методах решения теплицевых систем уравнений используется алгоритм Евклида. Наиболее подходящей для этого является описанная в разд. 10.7 рекурсивная форма алгоритма Евклида.

Начнем со следующей теплицевой системы уравнений:

$$\begin{bmatrix} a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0 \\ a_n & a_{n-1} & a_{n-2} & \cdots & a_1 \\ a_{n+1} & a_n & a_{n-1} & \cdots & a_2 \\ \vdots & & & & \\ a_{2n-2} & a_{2n-3} & a_{2n-4} & \cdots & a_{n-1} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} -a_n \\ -a_{n+1} \\ \vdots \\ -a_{2n-1} \end{bmatrix}.$$

Это та же самая теплицева система уравнений, которую в § 11.3 мы решали с помощью алгоритма Берлекэмпа—Месси.

Пусть

$$a(x) = \sum_{i=0}^{2n-1} a_i x^i, \quad f(x) = 1 + \sum_{i=1}^{n} f_i x^i,$$

и рассмотрим произведение этих многочленов $g(x) = f(x)\,a(x)$. Из выписанного матричного уравнения видно, что

$$g_i = 0, \quad i = n, \ldots, 2n-1,$$

но при значениях индекса i, больших чем $2n-1$, коэффициенты g_i могут быть и ненулевыми. Рассмотрим задачу вычисления таких $f(x)$ и $g(x)$, которые удовлетворяют условиям $\deg f(x) \leqslant n$, $\deg g(x) \leqslant n-1$ и

$$g(x) = f(x)\,a(x) \pmod{x^{2n}}.$$

Одним из путей решения этой задачи, конечно, является решение исходного матричного уравнения относительно вектора $\mathbf{f}$ и дальнейшего вычисления многочлена $g(x)$. Другой метод основан на использовании алгоритма Евклида для многочленов.

Для того, чтобы увидеть, как можно решать это уравнение относительно $f(x)$ и $g(x)$, вернемся к доказательству алгоритма

Евклида. Из этого доказательства легко увидеть, что

$$\begin{bmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \begin{bmatrix} A_{11}^{(r)}(x) & A_{12}^{(r)}(x) \\ A_{21}^{(r)}(x) & A_{22}^{(r)}(x) \end{bmatrix} \begin{bmatrix} s(x) \\ t(x) \end{bmatrix},$$

так что

$$t^{(r)}(x) = A_{22}^{(r)}(x)\, t(x) \pmod{s(x)},$$

что при $t(x) = a(x)$ и $s(x) = x^{2n}$ является одной из форм решаемого нами уравнения. Выписанное уравнение справедливо при всех r. Чтобы решить задачу, надо найти такое r, при котором $\deg A_{22}^{(r)}(x) \leqslant n$ и $\deg t^{(r)}(x) \leqslant n-1$. Если такое r существует, то многочлены $A_{22}^{(r)}(x)$ и $t^{(r)}(x)$ должны равняться искомым многочленам $f(x)$ и $g(x)$. Выберем значение r, при котором удовлетворяются неравенства

$$\deg t^{(r-1)}(x) \geqslant n \text{ и } \deg t^{(r)}(x) \leqslant n-1.$$

Так как $\deg t^{(0)}(x) = 2n$ и с ростом r степень многочленов $t^{(r)}(x)$ строго убывает, то это определяет r однозначно. При таком определении r требование $\deg t^{(r)}(x) \leqslant n-1$ удовлетворено. С ростом r степень многочлена $A_{22}^{(r)}(x)$ растет, и надо только показать, что $\deg A_{22}^{(r)}(x) \leqslant n$. Для того, чтобы это доказать, воспользуемся матрицей, обратной к матрице $\mathbf{A}^{(r)}(x)$. Сначала напомним, что

$$\mathbf{A}^{(r)}(x) = \prod_{l=1}^{r} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(l)}(x) \end{bmatrix}.$$

Отсюда ясно, что $\deg A_{22}^{(r)}(x) > \deg A_{22}^{(r)}(x)$. Напомним также, что $\deg s^{(r)}(x) > \deg t^{(r)}(x)$. Из этих неравенств и матричного уравнения

$$\begin{bmatrix} s(x) \\ t(x) \end{bmatrix} = (-1)^r \begin{bmatrix} A_{22}^{(r)}(x) & -A_{12}^{(r)}(x) \\ -A_{21}^{(r)}(x) & A_{11}^{(r)}(x) \end{bmatrix} \begin{bmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix}$$

следует, что $\deg s(x) = \deg A_{22}^{(r)}(x) + \deg s^{(r)}(x)$, и так как $s^{(r)}(x) = t^{(r-1)}(x)$, то это преобразуется к соотношению

$$\deg A_{22}^{(r)}(x) = \deg s(x) - \deg t^{(n-1)}(x) \leqslant 2n - n = n,$$

где неравенство вытекает из определения параметра r.

Мы получили почти полное доказательство следующей теоремы.

Теорема 11.5.1. *При заданных $s^{(0)}(x) = x^{2n}$ и $t^{(0)}(x) = a(x)$ пусть*

$$\mathbf{A}^{(0)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

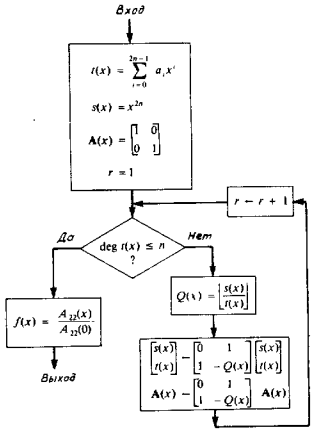


Рис. 11.9. Решение теплицевой системы с помощью алгоритма Евклида.

Продолжая рекурсию до тех пор, пока $\deg t^{(r)}(x) < n-1$, решим следующую систему рекуррентных уравнений:

$$Q^{(r)}(x) = \left\lfloor \frac{s^{(r-1)}(x)}{t^{(r-1)}(x)} \right\rfloor,$$

$$\mathbf{A}^{(r)}(x) = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} \mathbf{A}^{(r-1)}(x),$$

$$\begin{bmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} \begin{bmatrix} s^{(r-1)}(x) \\ t^{(r-1)}(x) \end{bmatrix},$$

и положим $g(x) = \Delta^{-1} t^{(r)}(x)$ и $f(x) = \Delta^{-1} A_{22}^{(r)}(x)$, где $\Delta = A_{22}^{(r)}(0)$. При условии, что Δ отлично от нуля, эти многочлены удовлетворяют уравнению

$$g(x) = f(x)\, a(x) \pmod{x^{2n}},$$

причем $\deg f(x) \leqslant n$, $\deg g(x) \leqslant n-1$ и $f_0 = 1$.

Доказательство. Деление на Δ обеспечивает равенство $f_0 = 1$. Справедливость остальных утверждений была установлена ранее. □

Таким образом, мы получили другой способ решения той же самой теплицевой системы уравнений (в случае, когда эта система обратима), которую мы уже решали раньше, используя алгоритм Берлекэмпа—Месси; блок-схема этого метода показана на рис. 11.9. Если теплицева система не обратима, то описанное применение алгоритма Евклида также приводит к многочлену $A^{(r)}_{22}(x)$ наименьшей степени, который удовлетворяет определяющему его равенству, но в этом случае $\Delta = 0$ и поэтому многочлен $f(x)$ не определен. Если $\Delta = 0$, то описанный алгоритм приводит к решению системы

$$\begin{bmatrix} a_n & a_{n-1} & \dots & a_0 \\ a_{n+1} & a_n & \dots & a_1 \\ \vdots & & & \\ a_{2n-2} & a_{2n-3} & \dots & a_{n-1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Задачи

11.1. Используя алгоритм Левинсона, решить в поле $GF(7)$ уравнение

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 1 \\ 3 \end{bmatrix}.$$

11.2. Реконструировать алгоритм Левинсона для случая, когда основное поле является комплексным, а матрица **A** эрмитовой.

11.3. Предположим, что входящая в алгоритм Берлекэмпа последовательность **a** является периодической, период N которой намного больше, чем $2n$.
а. Показать, как можно полностью вычислить последовательность по ее первым $2n$ компонентам.
б. Предположим, что вместо вектора **a** задано его преобразование Фурье, вектор **A**. Показать, как надо модифицировать алгоритм Берлекэмпа — Месси для того, чтобы использовать непосредственно вектор **A**, не вычисляя обратного преобразования Фурье.

11.4. Используя алгоритм Берлекэмпа — Месси для входной последовательности $(a_0, a_1, \dots, a_7) = (0, 0, 0, 0, 0, 0, 0, 1)$, найти решение **f**, длина которого больше 4. Эта задача является примером того, как алгоритм Берлекэмпа — Месси приводит к авторегрессионному фильтру даже в тех случаях, когда теплицева матрица вырождена.

11.5. Используя алгоритм Евклида, решить уравнение

$$\begin{bmatrix} 3 & 2 & 1 \\ 3 & 3 & 2 \\ 2 & 3 & 3 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = - \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}.$$

11.6. Доказать, что все шаги рекурсии алгоритма Левинсона можно провести в том и только в том случае, когда все вложенные главные подматрицы невырождены.

11.7. Сформулировать условия невырожденности алгоритма Дурбина.

Замечания

Первый быстрый алгоритм решения теплицевых систем уравнений был разработан Левинсоном (1947). Он был сразу же отнесен скорее к области приложений к задачам винеровской фильтрации, чем к задачам построения быстрых алгоритмов как таковых. Другие быстрые алгоритмы решения некоторых теплицевых систем уравнений были построены Дурбиным (1960), Тренчем (1964) и Берлекэмпом (1968). Упрощение алгоритма Тренча предложил Зонар (1974). Месси (1969) упростил алгоритм Берлекэмпа, интерпретируя его как метод построения регистра сдвига с линейной обратной связью. Велч и Шольц (1979) списали этот же алгоритм с точки зрения алгоритма построения непрерывной дроби. Рекурсивную форму алгоритма Берлекэмпа — Месси описал Блейхут (1983). Использование алгоритма Евклида для решения теплицевых систем уравнений предложено Сугиямой, Касахарой, Хирасавой и Намекавой (1975). Ускорение алгоритма Евклида с помощью дублирования в его приложении к решению теплицевых систем уравнений разработали Брент, Густавсон и Юн (1980).

Неудивительно, что быстрые алгоритмы решения теплицевых систем уравнений связаны со многими другими задачами. Если выписать шаги алгоритма Левинсона в обратном порядке, то он преобразуется в алгоритм, известный как критерий Шура — Коена для тестирования стабильности авторегрессионного фильтра, задаваемого многочленом $a(x)$. Эту связь отметили Вийера и Кайлат (1977).

Объем книги немедленно бы удвоился, если бы мы стали рассматривать такие обобщения и специализации теплицевых матриц, как блочно-теплицевы матрицы или почти теплицевы матрицы. К таким работам относятся статьи Вингинса и Робинсона (1965), Дикинсона (1979), Дикинсона, Морфа и Кайлата (1974), Фридлендера, Морфа, Кайлата и Льюнга (1979), Морфа, Дикинсона, Кайлата и Вайеры (1977), Мондена и Аримото (1980).

Глава 12

БЫСТРЫЕ АЛГОРИТМЫ ПОИСКА ПО РЕШЕТКЕ И ПО ДЕРЕВУ

Машина с конечным числом состояний, производящая в каждый момент времени один или более элементов некоторого поля F, генерирует последовательность элементов этого поля. Множество всех возможных таких последовательностей на выходе машины с конечным числом состояний описываются графом определенного вида, который называется решеткой или, в случае очень большого числа состояний, деревом. Имеется много приложений, в которых по результатам наблюдений в шуме одной из таких последовательностей требуется оценить либо саму эту последовательность, либо историю машины с конечным числом состояний. Это задача поиска по решетке или по дереву такого пути, который лучше всего отвечает данной последовательности. Такое направление в обработке дискретных сигналов сильно отличается от других направлений, и алгоритмы, рассматриваемые в настоящей главе, сильно отличаются от уже изученных нами алгоритмов.

Алгоритмы поиска по решетке и по дереву находят приложения в таких областях, как декодирование сверточных кодов, демодуляция сигналов при наличии межсимвольной интерференции, демодуляция сигнала по частичному отклику или фазово-разностно-модулированных сигналов, распознавание речи и текстовых символов.

12.1. Поиск по решетке и по дереву

Решетки и деревья, по которым реализуется поиск, возникают как описания машин с конечным числом состояний. Машина с конечным числом состояний задается множеством состояний, в которых она может находиться, множеством переходов между этими состояниями и выходными символами, соответствующими каждому такому переходу. Простейший способ построения машины с конечным числом состояний с помеченными переходами состоит в использовании регистра сдвига так, как показано на примере на рис. 12.1. Состояния машины в этом примере описываются двумя содержащимися в ячейках регистра сдвига битами; всего имеется четыре так определяемых состояния. В каждый момент времени на вход поступают два бита, так что из каждого состояния возможны

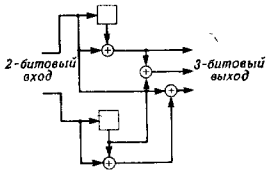


Рис. 12.1 Пример машины с конечным числом состояний.
+ обозначает сложение по модулю 2 (исключающее ИЛИ)

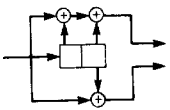


Рис. 12.2. Другой пример машины с конечным числом состояний.
+ обозначает сложение по модулю 2 (исключающее ИЛИ)

четыре перехода. В данном частном примере из каждого состояния возможен прямой переход в любое другое состояние. Изображенная на рис. 12.1 схема имеет три выходных бита; каждый переход порождает трехбитовую последовательность.

На рис. 12.2 показан другой пример, в котором уже нельзя из произвольного состояния перейти в любое другое состояние. Для достижения некоторых состояний требуется совершить два перехода. Схема на рис. 12.2 имеет двухбитовый выход.

Диаграмма переходов для машины с конечным числом состояний показана на рис. 12.3. Переходы помечены векторами длины два. В каждый момент времени машина делает переход между состояниями вдоль пути и метит путь переходов. Приведенная на рис. 12.3 диаграмма соответствует схеме на рис. 12.2.

Как правило, вид графа, который называется *решеткой*, можно использовать для описания выходных последовательностей произвольной машины с конечным числом состояний. Типичная решетка, из каждой вершины которой выходят два ребра, показана на рис. 12.4. Вершины любого столбца решетки обозначают состояния, в которых может находиться машина. Каждый последовательный столбец соответствует последовательным моментам времени и содержит одно и то же множество состояний. Ребра, связывающие вершины двух столбцов, представляют собой все возможные переходы в течение данного временного интервала, именуемого *кадром*. В общем случае решетка представляет собой полубесконечный вправо прямоугольный граф, число вершин в каждом столбце которого конечно. Расположение ребер, связывающих вершины данного столбца с вершинами следующего справа столбца, одно и то же для каждого столбца. Диаграмма начинается в верхней левой вершине; так как движение осуществляется только вправо, то недостижимые левые узлы на диаграмме не указываются. *Длиной кодового ограничения* решетки называется число битов, необходимых для определения состояния на решетке. Длина кодового ограничения решетки, показанной на рис. 12.4, равна двум.

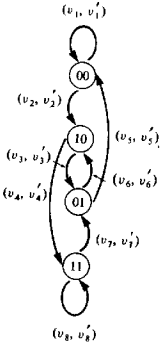


Рис. 12.3. Диаграмма состояний.

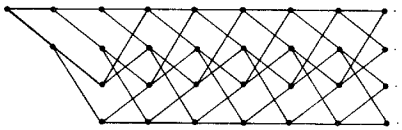


Рис. 12.4. Простая решетка.

Каждый кадр машины с конечным числом состояний приводит к изменению состояния. На решетке это изменение показано ребром, входящим в следующую вершину. Каждое ребро на рис. 12.5 помечено. В общем случае ребро маркируется некоторым фиксированным числом n_0 элементов основного поля F. При переходе от кадра к кадру множество меток на ребре может быть как фиксированным, так и переменным. Машина с конечным числом состояний может двигаться по решетке слева направо по многим путям решетки. Идя по каждому из этих путей, она вычисляет некоторую последовательность элементов поля F, которая и является ее выходной последовательностью.

Пусть $\mathbf{c} = \{c_i,\ i = 0, \ldots\}$ обозначает последовательность символов источника над полем F, генерируемых машиной с конечным числом состояний, и пусть $\mathbf{v} = \{v_i,\ i = 0, \ldots\}$ обозначает последовательность символов данных над F. Во многих приложениях $v_i = c_i + e_i$, где e_i — составляющая ошибки. О такой ситуации говорят, что последовательность на выходе источника наблюдается в аддитивном шуме.

Задача поиска по решетке состоит в вычислении такого пути на решетке, для которого последовательность $\mathbf{c}$ согласуется с заданной последовательностью $\mathbf{v}$ данных наилучшим образом. Для измерения меры согласованности необходимо ввести функцию расстояния.

Расстояние на множестве S определяется как вещественная функция $d(\alpha, \beta)$ на паре элементов из S, удовлетворяющая условиям

1. $d(\alpha, \beta) \geqslant 0$ и $d(\alpha, \alpha) = 0$,
2. $d(\alpha, \beta) = d(\beta, \alpha)$.

Если эта функция удовлетворяет также условию

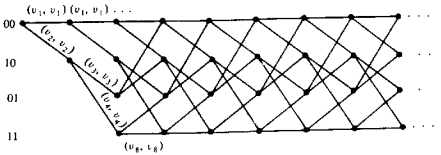


Рис. 12.5. Маркированная решетка.

3. $d(\alpha, \beta) \leqslant d(\alpha, \gamma) + d(\gamma, \beta)$ для всех γ из S,

то она называется *метрикой*. *Евклидово расстояние*, определяемое на n-мерном векторном пространстве формулой

$$d(\mathbf{v}, \mathbf{w}) = \sum_{i=0}^{n-1} (v_i - w_i)^2,$$

является метрикой. Каждый состоящий из l кадров путь на решетке определяет вектор длины ln_0 над полем F. Если из каждой вершины решетки выходит q путей, то на длине путей в l кадров мы получаем q таких векторов длины ln_0. Задача поиска по решетке состоит в выделении того из этих векторов, который ближе всего в данном расстоянии подходит к данному вектору $\mathbf{v}$ длины ln_0.

Наиболее простой для понимания метод поиска по решетке состоит в вычислении всех расстояний данного вектора $\mathbf{v}$ от q^l последовательностей, производимых данной решеткой, и нахождении в полученном списке наименьшего расстояния. Это можно проделать, мысленно накладывая заданную последовательность $\mathbf{v}$ на каждый из возможных путей на решетке. Сложность этой процедуры растет экспоненциально по l, так что при больших l она практически неприемлема. Обычно l столь велико, что его можно полагать бесконечным. Практически алгоритм не может рассматривать всю входную последовательность целиком при больших l. Он начинает с самого начала и, двигаясь вдоль последовательности, принимает окончательные решения. Эффективный поиск по решетке можно осуществлять по алгоритму Витерби, который будет описан в следующем разделе.

В некоторых случаях число состояний машины с конечным числом состояний велико. Тогда число узлов в каждом кадре решетки также велико, иногда настолько, что его можно полагать неограниченным. В этом случае исходная часть решетки тоже растет неограниченно, и в графе мы пренебрегаем тем, что на самом деле при возврате машины с конечным числом состояний в некоторое состояние в конечном итоге пути рекомбинируют. Ведь в каждый момент времени приходится рассматривать лишь малые фрагменты решетки. Хорошим изображением того, что происходит,

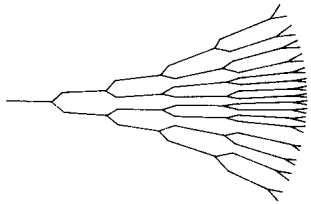


Рис. 12.6. Дерево с двумя ребрами в каждом узле.

является представленное на рис. 12.6 *дерево*. Число узлов дерева растет неограниченно.

Так же как на решетке, l кадров маркированного дерева определяют q^l векторов. Задача поиска по дереву состоит в нахождении того из этих векторов, который ближе всего к заданному вектору $\mathbf{v}$ данных. Мы можем мысленно наложить данную последовательность на каждый из возможных путей дерева длиной в l кадров и определить наилучшее совпадение. Опять сложность этой процедуры растет экспоненциально по l, так что практически она неприемлема. Эффективные процедуры поиска по дереву даются последовательными алгоритмами, рассматриваемыми в следующих параграфах.

12.2. Алгоритм Витерби

Принцип оптимальности динамического программирования утверждает, что если в некотором смысле оптимальный путь от точки A к точке C проходит через точку B, то отрезок пути из точки B в точку C совпадает с оптимальным путем из точки B в точку C. Этот принцип применим к задаче отыскания лучшего пути по решетке. Основанная на этом принципе итеративная процедура построения переменного множества претендентов на лучший путь по решетке известна под названием алгоритма Витерби.

Алгоритм Витерби для решетки с q ребрами в каждой вершине и длиной кодового ограничения ν оперирует q^ν претендентами, так что его сложность пропорциональна q^ν. Практически алгоритм пригоден только при малых ν. Если длина кодового ограничения равна 10 и из каждой вершины решетки выходят два ребра, то алгоритм Витерби оперирует 1024 претендентами. Практически это допустимо. Но при длине кодового ограничения 20 число претендентов уже больше миллиона, так что алгоритм практически неприемлем.

Двигаясь вдоль решетки, алгоритм Витерби итеративно обрабатывает кадр за кадром. Находясь на некотором кадре в момент l,

алгоритм не знает, какая из вершин является ближайшей, и даже не пытается вычислить эту вершину. Вместо этого алгоритм определяет лучший из путей от начальной вершины до каждой из вершин l-го кадра. Он вычисляет расстояния между данной последовательностью и всеми претендентами для каждой вершины l-го кадра. Для каждого из путей-претендентов это расстояние называется его *расходимостью*. Если все пути-претенденты проходят в первом кадре через одну и ту же вершину, алгоритм находит первый кадр ближайшего пути, но не принимает при этом никакого решения относительно l-го кадра.

Далее алгоритм вычисляет пути-претенденты в каждую новую вершину $(l + 1)$-го кадра. Но чтобы попасть в любую новую вершину $(l + 1)$-го кадра, путь должен пройти через одну из старых вершин l-го кадра. Таким образом, пути-претенденты в новые вершины можно получить продолжением старых претендентов, которые допускают такое продолжение. Ближайший путь вычисляется прибавлением приращений расходимости каждого пути к расходимости лучшего пути в старую вершину. В каждую новую вершину ведут q^ν путей, и путь к новой вершине с минимальным приращением является ближайшим. Этот процесс повторяется для каждой из новых вершин. В конце вычислений, связанных с новым кадром, алгоритм вычисляет ближайший путь в каждую из вершин этого кадра $l + 1$. Если опять все эти пути проходят через одну и ту же вершину второго кадра, то алгоритм Витерби успешно вычисляет второй кадр.

Этот процесс повторяется во всех успешных кадрах. Если в некоторый момент возникает несколько ближайших путей в данную вершину, то алгоритм должен разрешить этот конфликт, используя некоторое правило. Альтернатива состоит в сохранении всех решений, так что, например, находятся два пути в данную вершину. Для лучшего понимания этого момента следует обратиться к рассматриваемому ниже примеру.

Для реализации алгоритма Витерби надо выбрать окно ширины b, которое, как правило, в несколько раз превышает ширину кодового ограничения ν и диктуется возможностями используемого компьютера. В момент n алгоритм просматривает все выжившие пути. Если в первом кадре они совпадают, то этот кадр выбирается в качестве первого кадра правильного пути; его можно передать пользователю.

Далее алгоритм отбрасывает первое ребро и переходит к вычислению следующей итерации, беря для этого новый кадр. Если опять все выжившие пути проходят через одну и ту же вершину наиболее старого кадра, то этот кадр принимается. Этот процесс повторяется бесконечно, вычисляя кадр за кадром.

Если b выбрано достаточно большим, то в данном кадре почти всегда можно принять однозначное решение. Редко, но встре-

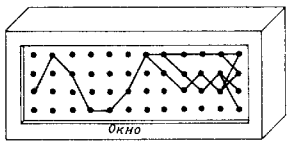


Рис. 12.7. Принципиальный вид алгоритма Витерби.

чаются ситуации, когда принять однозначное решение и отбросить кадр не удается либо потому, что имеется несколько ближайших путей, либо потому, что ширина окна b слишком мала для принятия решения. Тогда используется некоторое правило разрешения этой неопределенности. Такое событие случается пренебрежимо редко.

Вместе с ростом числа кадров растет расходимость каждого пути. Чтобы избежать переполнения памяти, приходится время от времени редуцировать расходимость. Простое правило такой редукции дается вычитанием наименьшей расходимости из всех расходимостей. Это не влияет на выбор минимальной расходимости.

Полезно представлять себе декодер Витерби как окно, через которое можно наблюдать часть решетки. Это иллюстрируется рис. 12.7. На рисунке можно видеть только часть решетки конечной длины, на которой отмечены выжившие пути и их расходимости. С течением времени решетка скользит влево. При появлении справа новых вершин некоторые пути продолжаются в них, а другие исчезают; самый старый столбец выходит из-под наблюдения налево. Со временем левый столбец вершин исчезает, и лишь для одной из его вершин будет существовать проходящий через нее путь.

На рис. 12.8 приведен пример. Для простоты и метки на решетке, и последовательность данных в примере выбраны двоичными. Евклидово расстояние заменяется просто числом позиций, в котором различаются две последовательности. Выберем ширину окна b равной 15. Предположим, что заданная последовательность равна

$$\mathbf{v} = 10100000100000000000000 \ldots,$$

и требуется найти путь на решетке, ближайший к заданной последовательности.

Диаграмма состояний декодера показана на рис. 12.8. На третьей итерации алгоритм находит ближайший путь к каждой вершине третьего кадра. Затем, продолжая пути, ведущие к вершинам (r — 1)-го кадра, и сохраняя ближайший путь к каждой вершине r-го кадра, алгоритм на r-й итерации находит ближайшие пути в вершины r-го кадра. Иногда может возникнуть неопределенность. В данном примере имеются две неопределенности в пятой итерации.

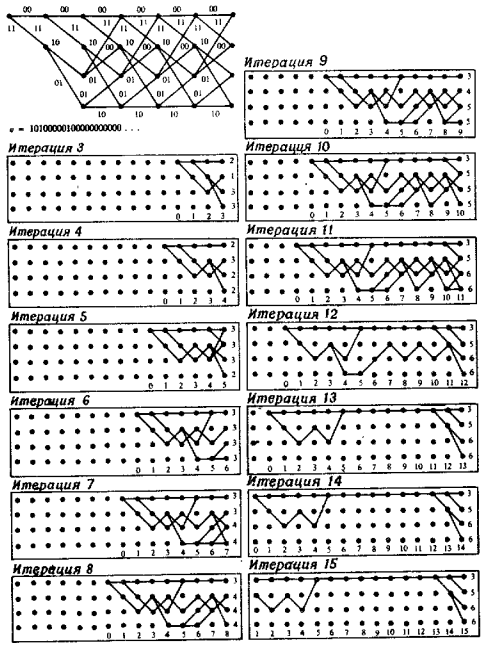


Рис. 12.8. Пример алгоритма Витерби.

Алгоритм может разрешить неопределенность случайным образом или сохранить оба пути, относительно которых существует неопределенность. В данном примере неопределенность сохраняется до тех пор, пока не находится лучший путь или сомнительный участок не выводится за пределы буфера. Если имеется только один путь, проходящий через вершину наиболее старого кадра, то принимается однозначное решение.

В этом примере ближайшей к данной последовательности будет либо нулевая последовательность, либо последовательность с началом 1110001011000... .

Истинная реализация рассматриваемого алгоритма может сильно отличаться от приведенного на рис. 12.8 символического описания. Например, выживающие пути на решетке можно задавать в виде таблицы 15-битовых чисел. При каждой итерации некоторые из этих 15-битовых чисел сдвигаются влево, выталкивая наиболее левый бит и добавляя справа новый бит. Другие из 15-битовых чисел этого списка не видоизменяются, а выбрасываются из списка.

12.3. Стек-алгоритм

Для больших решеток алгоритм Витерби быстро становится непрактичным; при длине кодового ограничения 10 алгоритм требует запоминания 1024 путей-претендентов. Для того чтобы ослабить влияние больших длин кодового ограничения, была разработана стратегия, игнорирующая маловероятные пути по решетке, как только они становятся маловероятными. Стратегия поиска на решетке только наиболее вероятных путей известна под общим названием последовательных алгоритмов. Как правило, в последовательных алгоритмах не принимается окончательных решений о непрерывном отбрасывании путей. Время от времени последовательный алгоритм принимает решение вернуться обратно и продолжить оставленный ранее путь. Последовательные алгоритмы в равной мере пригодны и для поиска по дереву, и для поиска по решетке.

Представим себе решетку с большой длиной кодового ограничения, скажем, равного 40. Для принятия первого решения в такой решетке надо обработать несколько сотен кадров. Каждый кадр содержит и вовсе огромное число вершин, а именно 2^{40}, или, примерно, 10^{12} вершин. Последовательный алгоритм должен найти наиболее близкий к заданной последовательности путь на этой решетке; он должен сделать это, по крайней мере, с очень высокой вероятностью, хотя время от времени ему дозволено отказаться от декодирования. Этот отказ связан не с недостаточностью данных, а с неполнотой алгоритма. В алгоритме Витерби мы не встречались с ошибками подобного сорта.

Последовательный алгоритм просматривает только первый кадр, принимает решение и переходит в вершину решетки первого уровня. Затем он повторяет эту процедуру. На каждом уровне, находясь только в одной вершине, он просматривает следующий кадр, выбирая ребро, ближайшее к принятому кадру, и переходит в вершину следующего уровня, прокладывая таким образом

путь на дереве. Если заданная последовательность точно совпадает с некоторым путем на дереве, то эта процедура работает хорошо. Однако при наличии несовпадений последовательный алгоритм выбирает иногда неправильный путь. Если алгоритм продолжает следовать по ложному пути, он внезапно обнаружит, что в каждом кадре расходимость слишком велика и, по-видимому, он на неправильном пути. Последовательный алгоритм вернется назад на несколько кадров и начнет исследовать альтернативные пути до тех пор, пока не найдется правдоподобный путь. Затем он будет двигаться вдоль этого альтернативного пути. Ниже будут кратко описаны правила, которыми он руководствуется при этом поиске.

Характеристики алгоритма зависят от ширины b окна. Если алгоритм нашел путь, проходящий через b кадров дерева, он принимает окончательное решение относительно самого старого кадра, выводит этот кадр и вдвигает в окно новый кадр.

Число вычислений, необходимое последовательному алгоритму для продвижения по дереву на одну вершину вглубь, является случайной величиной. Эта величина служит основной характеристикой последовательного алгоритма. Она представляет собой основной параметр, определяющий сложность алгоритма, необходимую для обеспечения заданного уровня его характеристик. Если помехи малы, то алгоритм может двигаться по правильному пути, используя для продвижения вглубь по дереву на одну вершину только по одному вычислению в каждом кадре. Но если имеются сбивающие помехи, то алгоритм может пойти по неправильному пути, что приводит к задержке и большому числу вычислений, предшествующих выходу на правильный путь. Переменность числа вычислений влечет за собой необходимость большого объема памяти для входных данных. Любой буфер конечного объема, независимо от его величины, при использовании в последовательном алгоритме имеет ненулевую вероятность переполнения. Такое поведение алгоритма следует рассматривать как одну из его характеристик. Переполнение буферной памяти является существенным фактором любого алгоритма, который осуществляет по меньшей мере одно вычисление в каждой посещаемой вершине и который последовательно просматривает ребра так, что записанные на более глубоких ребрах дерева данные не используются. Второе условие оказывается критическим и приводит к особому поведению алгоритма.

Используемые последовательные алгоритмы разбиваются на два класса: стек-алгоритм, рассматриваемый в настоящем разделе, и алгоритм Фано, который будет рассмотрен в следующем разделе. Структура их совершенно различна. Используемые в этих двух подходах метрики являются предметом непрерывных обсуждений. Необходимое число вычислений в стек-алгоритме меньше,

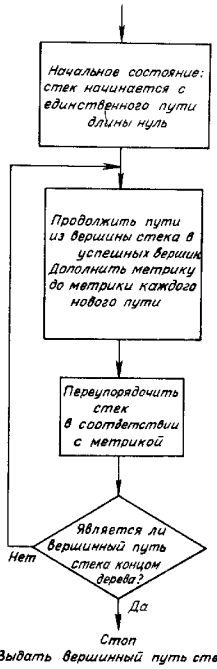


Рис. 12.9. Упрощенный стек-алгоритм.

чем в алгоритме Фано. Стек-алгоритм запоминает уже вычисленный набор путей. Это контрастирует с поведением алгоритма Фано, который, найдя свой путь в некоторую вершину, может вернуться на какое-то расстояние обратно только для того, чтобы повторить во всех деталях путь в ту же вершину. Стек-алгоритм предпочитает запоминать проделанную работу, а не повторять ее. С другой стороны, для стек-алгоритма требуется существенно больший объем памяти и бо́льшая вычислительная работа на каждой итерации.

Стек-алгоритм легок для понимания; его упрощенная блок-схема приведена на рис. 12.9. Алгоритм организуется вокруг стека уже вычисленных на предыдущих итерациях путей различной длины. Каждый вход в стек представляет собой путь, описываемый тремя величинами: длиной пути; последовательностью переменной длины, содержащей определившие данный путь переходы между состояниями; и расходимостью данного пути. В начальный момент стек содержит только тривиальный путь нулевой длины.

Расходимость пути определяется как расстояние между сегментом пути и начальным сегментом данных той же самой длины. Используемые в последовательных алгоритмах расстояния измеряются многими разными способами. Во многих важных задачах последовательность данных представляет собой последовательность записанных на ребрах символов, искаженных аддитивным шумом. В этом случае мера расстояния известна как *метрика Фано*, которая сейчас будет определена.

Расходимость пути определяется как логарифмическая функция правдоподобия этого пути и вычисляется следующим образом. Первые $N+1$ кадров заданного слова можно записать в виде

$$\mathbf{v}^{(N)} = (v_0^1, \ldots, v_0^{n_0}, v_1^1, \ldots, v_1^{n_0}, \ldots, v_N^1, \ldots, v_N^{n_0}).$$

Первые $m+1$ кадров символов вдоль произвольного пути можно записать в виде

$$\mathbf{c}^{(m)} = (c_0^1, \ldots, c_0^{n_0}, c_1^1, \ldots, c_1^{n_0}, \ldots, c_m^1, \ldots, c_m^{n_0}).$$

Мы хотим отыскать отрезок пути на дереве, который максимизирует $\Pr(\mathbf{c}^{(m)} \mid \mathbf{v}^{(N)})$. Последовательный алгоритм выбирает начальный отрезок пути по дереву, не заглядывая в глубь дерева. По правилу Байеса,

$$\Pr(\mathbf{c}^{(m)} \mid \mathbf{v}^{(N)}) = \frac{\Pr(\mathbf{v}^{(N)} \mid \mathbf{c}^{(m)}) \Pr(\mathbf{c}^{(m)})}{\Pr(\mathbf{v}^{(N)})}.$$

Член $\Pr(\mathbf{v}^{(N)} \mid \mathbf{c}^{(m)})$ можно разбить на два множителя

$$\Pr(\mathbf{v}^{(N)} \mid \mathbf{c}^{(m)}) = \left[\prod_{i=0}^{m}\prod_{j=1}^{n_0} \Pr(v_i^j \mid c_i^j)\right]\left[\prod_{i=m+1}^{N}\prod_{j=1}^{n_0} \Pr(v_i^j)\right].$$

Первый член равен произведению условных вероятностей для $m+1$ кадров истинного пути; второй множитель равен произведению безусловных вероятностей по тем кадрам, где путь не определен. Это разложение является следствием определения последовательного алгоритма. Алгоритм более общего вида не обязательно допускает такое разбиение.

Теперь будем максимизировать вдоль путей величину

$$\frac{\Pr(\mathbf{v}^{(N)} \mid \mathbf{c}^{(m)}) \Pr(\mathbf{c}^{(m)})}{\Pr(\mathbf{v}^{(N)})} = \left[\frac{\prod_{i=0}^{m}\prod_{j=1}^{n_0} \Pr(v_i^j \mid c_i^j)}{\prod_{i=0}^{m}\prod_{j=1}^{n_0} \Pr(v_i^j)}\right] \Pr(\mathbf{c}^{(m)}).$$

Так как все пути предполагаются равновероятными, то член $\Pr(\mathbf{c}^{(m)})$ является константой. Его можно отбросить, и это не влияет на выбор максимума. Используемая в рассматриваемом последовательном алгоритме метрика, именуемая метрикой Фано, задается логарифмической функцией

$$\mu(\mathbf{c}^{(m)}) = \log_q\left[\frac{\Pr(\mathbf{v}^{(N)} \mid \mathbf{c}^{(m)}) \Pr(\mathbf{c}^{(m)})}{\Pr(\mathbf{v}^{(m)})}\right] =$$

$$= \sum_{i=0}^{m}\left[\sum_{j=1}^{n_0} \log_q \frac{p(v_i^j \mid c_i^j)}{p(v_i^j)}\right].$$

Здесь i-я скобка представляет собой вклад метрики Фано на i-м кадре

$$\mu_i = \sum_{j=1}^{n_0} \log_q \frac{p(v_i^j \mid c_i^j)}{p(v_i^j)}.$$

Для каждого рассматриваемого пути в i-м кадре требуется вычислять только этот член. Метрика $\mu(c^{(m)})$ получается прибавлением нового приращения к метрике Фано пути, который записан на предыдущей итерации в вершине стека.

12.4. Алгоритм Фано

В алгоритме Фано требуется знать среднюю расходимость на кадр правильной последовательности $\bar{d}$ или хотя бы верхнюю границу для $\bar{d}$. Это предполагает наличие некоторой меры статистической регулярности расходимости, так что средняя расходимость может быть использована как типичная. Пока алгоритм Фано следует по правильному пути, можно ожидать, что в пределах первых l кадров расходимость равна примерно $\bar{d}l$. Алгоритм допускает несколько бо́льшую величину расходимости, но если она намного больше, то алгоритм сделает вывод, что он на неправильном пути. Выберем (быть может, моделированием) некоторый параметр $\bar{d}'$, больший, чем $\bar{d}$, и определим перекошенное расстояние формулой [1])

$$t(l) = \bar{d}'l - d(l),$$

где $d(l)$ равно расходимости текущего пути-претендента на дереве. Для правильного пути $d(l)$ приблизительно равно $\bar{d}l$, а $t(l)$ положительно и возрастает. Алгоритм следует по пути на дереве до тех пор, пока $t(l)$ возрастает. Если вдруг $t(l)$ станет уменьшаться, алгоритм заключает, что в некоторой вершине он выбрал ошибочное ребро и возвращается по дереву обратно, проверяя другие пути. Он может найти лучший путь и следовать по нему или может возвратиться к той же самой вершине, но уже более уверенно продолжать путь через нее. Для того чтобы решить, когда $t(l)$ начинает уменьшаться, в алгоритме Фано используется переменный порог T, который всегда кратен приращению Δ порога. Пока алгоритм движется вперед, порог, оставаясь не большим $t(l)$ и кратным приращению Δ, принимает максимально возможное при этих ограничениях значение. Благодаря тому, что T квантуется с шагом Δ, допускается небольшое убывание $t(l)$ без пересечения порога.

[1]) В этом выражении знак выбран так, чтобы правильной работе алгоритма соответствовало положительное значение расстояния.

В алгоритме Фано требуется, чтобы q ребер, выходящих из каждой вершины, были перенумерованы согласно некоторому правилу упорядочивания. Это правило ставит в соответствие каждому ребру индекс j, $j = 0, \ldots, q-1$. Нет необходимости запоминать эти индексы в каждом ребре; достаточно знать правило, по которому при возвращении алгоритма в вершину по ребру с известным индексом j он может переупорядочить ребра, найти ребро j и затем найти ребро с индексом $j+1$. Наиболее общим правилом является правило минимального расстояния. Ребра упорядочиваются в соответствии с их расстояниями до соответствующего кадра заданного слова, а неопределенность разрешается по любому удобному подправилу. Однако алгоритм будет правильно работать при любом фиксированном порядке ребер.

Фиксированное правило упорядочивания может привести к более длительному поиску назад-вперед, но освобождает от необходимости вычисления и перевычисления при достижении каждой вершины. Какое правило упорядочивания более просто в реализации, зависит от деталей конструкции. Для простоты понимания структуры алгоритма правило упорядочивания лучше оставить несколько неопределенным; предположим только, что в каждой вершине ребро, ближайшее к ребру данных, занумеровано первым. Алгоритм будет отыскивать ребра из каждой вершины в соответствии с этим правилом.

Основанная на регистрах сдвига реализация алгоритма Фано показана на рис. 12.10. Основой реализации является копия машины с конечным числом состояний, которая на рисунке представлена регистрами сдвига; к ним добавлено несколько вспомогательных запоминающих регистров. Алгоритм пытается подать символы в копию машины с конечным числом состояний таким образом, чтобы на ее выходе сформировалась последовательность, достаточно близкая к заданной последовательности. На каждой итерации алгоритм имеет доступ к самому последнему кадру, введенному в копию машины с конечным числом состояний. Он может изменить символ в этом кадре, вернуться к более раннему кадру или ввести новый кадр. Что именно делать, алгоритм решает на основе сравнения величины перекошенного расстояния $t(l)$ и текущего значения порога T.

В упрощенной форме алгоритм Фано показан на рис. 12.11. Практические детали, связанные с конечностью размеров буфера, временно игнорируются.

Если последовательность данных близка к генерируемой последовательности, то алгоритм будет циркулировать по правой петле на рис. 12.11, и при каждом цикле все регистры на рис. 12.10 сдвигаются на один кадр вправо. До тех пор пока $t(l)$ остается выше порога, алгоритм продолжает сдвигаться вправо и повышать

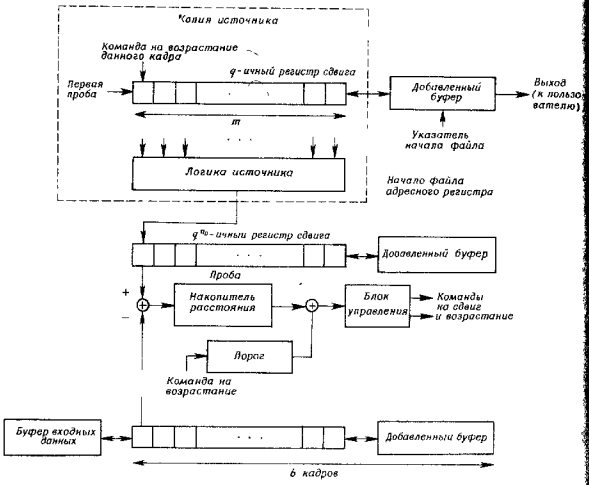


Рис. 12.10. Реализация алгоритма Фано на регистрах сдвига.

порог так, чтобы тот оставался близким к $t(l)$. Если $t(l)$ спускается ниже порога, то алгоритм Фано проверяет альтернативные ребра этого кадра, пытаясь найти то ребро, которое делает $t(l)$ выше порога. Если ему не удается сделать этого, то он возвращается назад. Как мы увидим в дальнейшем, если алгоритм начал возвращаться обратно, то логика заставит двигаться его назад до тех пор, пока не будет найден альтернативный путь, который находится над текущим значением порога, или вершина, в которой был установлен текущий порог. Затем алгоритм снова движется вперед с уже пониженным значением порога; но теперь, как мы убедимся позднее, порог не повышается до тех пор, пока алгоритм не придет в новую, ранее не рассматривавшуюся вершину. Каждый раз, когда алгоритм, двигаясь вперед, посещает ранее исследовавшуюся вершину, он имеет меньший порог. Алгоритм никогда не посетит одну и ту же вершину дважды с одинаковым значением порога. Следовательно, он может посещать любую вершину только конечное число раз. Это поведение гарантирует нас от зацикливания алгоритма; он должен продолжать движение вперед по последовательности данных.

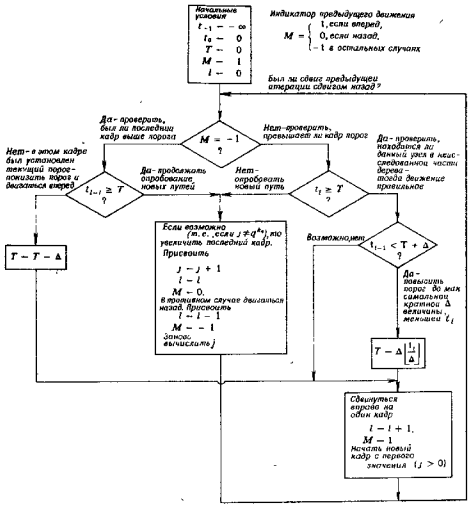


Рис. 12.11. Аннотированная блок-схема алгоритма Фано.

Замечания: Кадры индексированы указателем кадров l.
Итерации не индексированы.
Ответвления от текущего узла индексированы с помощью j.

Теперь необходимо доказать два сделанных ранее утверждения: 1) если алгоритм Фано не может найти альтернативный путь, то он движется назад к вершине, в которой было установлено текущее значение порога, и понижает его; 2) алгоритм не будет повышать порог до тех пор, пока не достигнет ранее не исследовавшегося узла. Что касается первого утверждения, очевидно, что если алгоритм Фано не может найти ребро, от которого он должен двигаться вперед, то он в конце концов должен вернуться в указанную вершину. Но если в предшествующем некоторой вершине кадре перекошенное расстояние $t(l-1)$ меньше текущего значения порога T, то порог был увеличен в l-м кадре. В блок-схему на рис. 12.11 включен этот тест для нахождения вершины, в которой был установлен текущий порог; теперь в этой вершине порог уменьшается.

Для доказательства второго утверждения заметим, что после того, как порог понижен на Δ, алгоритм Фано ищет ребра в том же порядке, что и до этого, до тех пор, пока не найдет место, в котором перекошенное расстояние, ранее бывшее меньше порога, становится больше него. До этого места порог T измениться не может. Это происходит из-за того, что после понижения порога на Δ перекошенное расстояние в тех вершинах, где оно превышало первоначальный порог, никогда не будет меньше $T + \Delta$. Когда алгоритм продвигается в новую часть дерева, он в конце концов достигает состояния, в котором $t(l-1) < T + \Delta$ и $t(l) \geqslant T$. В этой точке порог повышается. Это и является тестом для определения того, что алгоритм посетил новую вершину и нет необходимости помнить точно все посещенные ранее вершины. Этот тест включен в схему на рис. 12.11.

Алгоритм Фано зависит от двух параметров p' и Δ, которые можно выбрать моделированием на ЭВМ. Практически необходимо время от времени уменьшать $t(l)$ и T так, чтобы эти числа не становились слишком большими. Вычитание из обеих величин некоторого числа, кратного Δ, не влияет на последующие вычисления.

В практической реализации существенен также выбор ширины окна b. На рис. 12.12 приведена более полная блок-схема алгоритма Фано, отображающая важную роль параметра b. Каждый раз, когда самый старый кадр достигает конца буфера, что отмечается указателем кадра, он выходит из окна, а указатель кадра изменяется так, чтобы всегда указывать на самый старый имеющийся кадр. Иногда алгоритм может попытаться вернуться назад так далеко, что отыскиваемый им кадр оказывается уже выведенным. Такое явление называется переполнением буфера и случается, когда указатель кадров принимает отрицательные значения. Переполнение буфера является существенным ограничением алгоритма Фано. Во многих приложениях вероятность переполнения буфера так медленно убывает с ростом размера буфера, что вне зависимости от того, насколько большим выбран буфер, эта проблема полностью не снимается.

Существует два способа управления переполнением буфера. Наиболее надежным является периодическая подача на вход машины с конечным числом состояний известной последовательности переходов, длина которой равна длине кодового ограничения. Если буфер переполнится, то алгоритм декларирует отказ и ждет начала следующей известной ему последовательности, при котором он снова начинает работу. Все данные, поступившие между моментом переполнения буфера и следующим включением, теряются.

В случае, когда длина кодового ограничения не слишком велика, альтернативный путь состоит в движении указателя кадров вперед. Если алгоритму удастся найти правильную вершину, то он сможет перенастроиться. Это возможно, если заданная последовательность содержит достаточно длинный сегмент, в котором генерируемая последовательность с ней в точности совпадает.

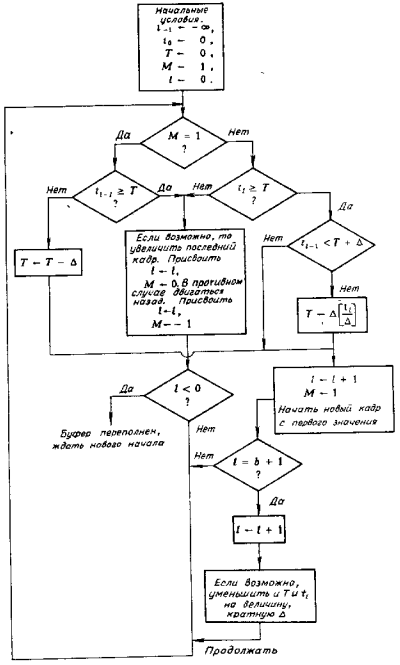


Рис. 12.12. Алгоритм Фано.

Задачи

12.1. Дать простую формулу метрики Фано для случая, когда разность между данной и полученной последовательностями задается гауссовским шумом с независимыми и одинаково распределенными отсчетами.

12.2. Повторить приведенный на рис. 12.7 пример для входной последовательности $\mathbf{v} = 1000100000100000\ldots$.

13.3. Алгоритм Витерби может быть запрограммирован в виде, в котором работоспособности путей на i-й итерации запоминаются в том же массиве, что и работоспособности путей на $(i-1)$-й итерации. Выписать последовательность адресов и вычислений, позволяющих сделать такую программу.

2.4. 2^v выживающих в алгоритме Витерби (при $q = 2$) путей можно записать в виде 2^v b-битовых двоичных слов. В типичных случаях $b = 40$ бит. Построить (используя такие средства, как указатели и битовые срезы) схему управления памятью для алгоритма Витерби, которая исключает необходимость считывания на каждой итерации 2^v b-битовых слов.

Замечания

Алгоритмы поиска по решетке появились сначала в связи с кодами, контролирующими ошибки, для которых задачу декодирования сверточных кодов можно интерпретировать как задачу поиска по решетке (или по дереву). Большинство публикаций по данному вопросу выполнено на языке кодов, контролирующих ошибки. Однако, в настоящее время к этим алгоритмам проявляется более широкий интерес, и в данной главе сделана попытка их изложения вне зависимости от какого бы то ни было приложения.

Хотя ретроспективно наиболее подходящим в качестве начала изложения данного круга вопросов представляется алгоритм Витерби, на самом деле первыми были разработаны последовательные алгоритмы. Последовательные алгоритмы были введены Возенкрафтом (1957) и подробно описаны Возенкрафтом и Рейффеном (1961). Дальнейшее развитие они получили у Фано (1963), Зигангирова (1966) и Джелинека (1969). Разработки Джелинека (1968) и Форни (1974), а также работы Шевилла и Костелло (1978) и Хаккоуна и Фергюсона (1975) также продвинули исследование данного вопроса. Наше изложение алгоритма Фано тесно связано с данным Галлагером (1968) изложением этого алгоритма. Первоначально Витерби (1967) опубликовал свой алгоритм скорее в учебных целях, чем в качестве серьезного алгоритма. Программные реализации алгоритма Витерби с эффективным использованием памяти описаны Рейдером (1981).

Нижние границы для распределения числа операций получили Джекобсон и Берлекэмп (1967), а верхние — Севедж (1966).

Приложение А

НАБОР АЛГОРИТМОВ ЦИКЛИЧЕСКИХ СВЕРТОК

В данном приложении приводится набор алгоритмов циклических сверток над полем вещественных чисел для длин $n = 2, 3, 4, 5, 7, 8$ и 9. Алгоритмы имеют вид равенств

$$\mathbf{s} = \mathbf{CGAd}.$$

Матрица $\mathbf{G}$ является диагональной и ее диагональные элементы выписываются в виде компонент вектора $\mathbf{G} = \mathbf{Bg}$. Матрицы $\mathbf{A}$ и $\mathbf{C}$ выписываются полностью. Кроме того, используя обозначения

$$\mathbf{D} = \mathbf{Ad}, \quad \mathbf{S} = \mathbf{GD}, \quad \mathbf{s} = \mathbf{CS},$$

мы приводим последовательности сложений, которыми можно заменить умножения на матрицы $\mathbf{A}$ и $\mathbf{C}$.

2-точечная циклическая свертка; 2 вещественных умножения, 4 вещественных сложения

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} G_0 \\ G_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix}$$

$$D_0 = d_0 + d_1 \qquad s_0 = S_0 + S_1$$
$$D_1 = d_0 - d_1 \qquad s_1 = S_0 - S_1$$

3-точечная циклическая свертка; 4 вещественных умножения, 11 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & -2 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 2 \\ 1 & 0 & 1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 3 & 0 & -3 \\ 0 & 3 & -3 \\ 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix}$$

$$t_0 = d_0 + d_1 \qquad T_0 = S_1 - S_3$$
$$D_0 = t_0 + d_2 \qquad T_1 = S_2 - S_3$$
$$D_1 = d_0 - d_2 \qquad s_0 = S_0 + T_0$$
$$D_2 = d_1 - d_2 \qquad s_1 = S_0 - T_0 - T_1$$
$$D_3 = d_1 + d_2 \qquad s_2 = S_0 + T_1$$

4-точечная вещественная свертка; 5 вещественных умножений, 15 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 2 & 0 & -2 & 0 \\ -2 & 2 & 2 & -2 \\ 2 & 2 & -2 & -2 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}.$$

$$\begin{aligned} t_0 &= d_0 + d_2 \\ t_1 &= d_1 + d_3 \\ D_0 &= t_0 + t_1 \\ D_1 &= t_0 - t_1 \\ D_2 &= d_0 - d_2 \\ D_3 &= d_1 - d_3 \\ D_4 &= D_2 + D_3 \end{aligned} \qquad \begin{aligned} T_0 &= S_0 + S_1 \\ T_1 &= S_0 - S_1 \\ T_2 &= S_2 - S_4 \\ T_3 &= S_2 + S_3 \\ s_0 &= T_0 + T_2 \\ s_1 &= T_1 + T_3 \\ s_2 &= T_0 - T_2 \\ s_3 &= T_1 - T_3 \end{aligned}$$

5-точечная циклическая свертка; 10 вещественных умножений, 31 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & -2 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 & -2 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 1 & 1 & -1 & -1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 1 \\ -1 & -1 & -2 & -1 & -1 & -2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \\ G_6 \\ G_7 \\ G_8 \\ G_9 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 5 & 0 & -5 & 5 & -5 \\ 0 & 5 & -5 & 5 & -5 \\ -2 & -2 & 3 & -2 & 3 \\ -5 & 5 & -5 & 5 & 0 \\ -5 & 5 & -5 & 0 & 5 \\ 3 & -2 & 3 & -2 & -2 \\ 0 & 0 & -5 & 5 & 0 \\ 0 & 5 & -5 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}$$

$$\begin{aligned} D_0 &= d_0 - d_4 \\ D_1 &= d_1 - d_4 \\ D_2 &= D_0 + D_1 \\ D_3 &= d_2 - d_4 \\ D_4 &= d_3 - d_4 \\ D_5 &= D_3 + D_4 \\ D_6 &= D_0 - D_1 \\ D_7 &= D_1 - D_4 \\ D_8 &= D_2 - D_5 \\ D_9 &= d_0 + d_1 + d_2 + d_3 + d_4 \end{aligned} \qquad \begin{aligned} T_0 &= S_0 + S_2 \\ T_1 &= S_1 + S_2 \\ T_2 &= S_3 + S_5 \\ T_3 &= S_4 + S_5 \\ T_4 &= S_6 + S_8 \\ T_5 &= S_7 + S_8 \\ s_0 &= T_0 - T_4 + S_9 \\ s_1 &= -T_0 - T_1 - T_2 - T_3 + S_9 \\ s_2 &= T_3 + T_5 + S_9 \\ s_3 &= T_2 + T_4 + S_9 \\ s_4 &= T_1 - T_5 + S_9 \end{aligned}$$

7-точечная циклическая свертка; 16 вещественных умножений, 70 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 1 & 0 & 0 & 0 & -3 \\ 1 & -1 & 1 & 0 & 0 & 0 & -1 \\ 1 & 2 & 4 & 0 & 0 & 0 & -7 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 1 & 1 & -3 \\ 0 & 0 & 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 1 & 2 & 4 & -7 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 1 & 1 & 1 & 0 \\ -1 & 1 & -1 & 1 & -1 & 1 & 0 \\ -1 & -2 & -4 & 1 & 2 & 4 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ -1 & -3 & -1 & -7 & -1 & -1 & -3 & 1 & -7 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 4 & 1 & 0 & -1 & -1 & -4 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 2 & 0 & 0 & -1 & 1 & -2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & -1 & -1 & -1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 4 & 1 & 1 \\ 0 & 1 & -1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 2 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2G_0 \\ 14G_1 \\ 6G_2 \\ 6G_3 \\ G_4 \\ G_5 \\ 14G_6 \\ 6G_7 \\ 6G_8 \\ G_9 \\ 2G_{10} \\ 14G_{11} \\ 6G_{12} \\ 6G_{13} \\ G_{14} \\ 7G_{15} \end{bmatrix} = \begin{bmatrix} 2 & 1 & -2 & -1 & 3 & -2 & -1 \\ -4 & -11 & 3 & 10 & -11 & 3 & 10 \\ 0 & -1 & 3 & -2 & -1 & 3 & -2 \\ 0 & 1 & 0 & -1 & 1 & 0 & -1 \\ 1 & -2 & -1 & 3 & -2 & -1 & 2 \\ -1 & 3 & -2 & -1 & 2 & 1 & -2 \\ 10 & -11 & 3 & 10 & -4 & -11 & 3 \\ -2 & -1 & 3 & -2 & 0 & -1 & 3 \\ -1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 3 & -2 & -1 & 2 & 1 & -2 & -1 \\ 0 & 1 & -2 & -1 & 2 & 0 & 0 \\ -2 & -9 & 5 & 12 & -2 & -2 & -2 \\ 0 & -1 & 3 & -2 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & -2 & -1 & 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \end{bmatrix}$$

$$\begin{aligned}
t_0 &= d_1 - d_6 & T_0 &= S_0 + S_{10} \\
t_1 &= d_2 - d_6 & T_1 &= S_1 + S_{11} \\
t_2 &= d_4 - d_6 & T_2 &= S_2 + S_{12} \\
t_3 &= d_5 - d_6 & T_3 &= S_3 + S_{13} \\
t_4 &= t_0 + t_1 & T_4 &= S_4 + S_{14} \\
t_5 &= -t_0 + t_1 & T_5 &= S_5 - S_{10} \\
t_6 &= t_2 + t_3 & T_6 &= S_6 - S_{11} \\
t_7 &= -d_4 + d_5 & T_7 &= S_7 - S_{12} \\
D_0 &= d_0 - d_6 & T_8 &= S_8 - S_{13} \\
D_1 &= D_0 + t_4 & T_9 &= S_9 - S_{14} \\
D_2 &= D_0 + t_5 & T_{10} &= T_1 + T_3 \\
D_3 &= D_1 + t_4 + t_4 + t_5 & T_{11} &= T_{10} + T_2 \\
D_4 &= t_1 & T_{12} &= T_0 + T_{11} \\
D_5 &= d_3 - d_6 & T_{13} &= T_{10} + T_3 \\
D_6 &= D_5 + t_6 & T_{14} &= T_{13} - T_2 \\
D_7 &= D_5 + t_7 & T_{15} &= (T_{13} + T_3 + T_3 + T_4) + T_2 \\
D_8 &= D_6 + t_6 + t_6 + t_7 & T_{16} &= -T_{12} - T_{13} - (T_{13} + T_3 + T_3 + T_4) \\
D_9 &= t_3 & T_{17} &= T_6 + T_8 \\
D_{10} &= D_5 - D_0 & T_{18} &= T_{17} + T_7 \\
D_{11} &= D_6 - D_1 & T_{19} &= T_5 + T_{18} \\
D_{12} &= D_7 - D_2 & T_{20} &= T_{17} + T_8 \\
D_{13} &= D_8 - D_3 & T_{21} &= T_{20} - T_7 \\
D_{14} &= D_9 - D_4 & T_{22} &= (T_{20} + T_8 + T_8 + T_9) + T_7 \\
D_{15} &= D_{11} + 2(d_2 + d_1 + d_0) + d_6 & T_{23} &= -T_{19} - T_{20} - (T_{20} + T_8 + T_8 + T_9) \\
& & s_0 &= T_{12} + S_{15} \\
& & s_1 &= T_{16} + T_{23} + S_{15} \\
& & s_2 &= T_{22} + S_{15} \\
& & s_3 &= T_{21} + S_{13} \\
& & s_4 &= T_{19} + S_{15} \\
& & s_5 &= T_{15} + S_{15} \\
& & s_6 &= T_{14} + S_{15}
\end{aligned}$$

8-точечная циклическая свертка; 14 вещественных умножений, 46 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 \\ 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & -1 & 1 & 1 & 1 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 & 0 & 1 & -1 & 0 & 1 & -1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & -1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & 0 & -1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & -1 & 0 & -1 & 1 & -1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 1 & 1 & 1 & -1 & 0 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 & 0 & 1 & -1 & 0 & -1 & 1 & 0 & 1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 2G_0 \\ 2G_1 \\ 2G_2 \\ 2G_3 \\ 4G_4 \\ 4G_5 \\ 8G_6 \\ 8G_7 \\ 2G_8 \\ 2G_9 \\ 2G_{10} \\ 2G_{11} \\ 2G_{12} \\ 2G_{13} \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \end{bmatrix}$$

$$
\begin{aligned}
t_0 &= d_0 + d_4 & T_0 &= S_8 + S_{16}\\
t_1 &= d_1 + d_5 & T_1 &= S_9 - S_{10}\\
t_2 &= d_2 + d_6 & T_2 &= S_3 + S_{11}\\
t_3 &= d_3 + d_7 & T_3 &= S_{11} - S_2\\
t_4 &= t_0 + t_2 & T_4 &= S_1 + S_{12}\\
t_5 &= t_1 + t_3 & T_5 &= S_0 - S_{12}\\
D_0 &= d_0 - d_4 & T_6 &= S_{13} - S_3\\
D_1 &= d_1 - d_5 & T_7 &= S_{13} + S_4\\
D_2 &= d_2 - d_6 & T_8 &= S_7 + S_6\\
D_3 &= d_3 - d_7 & T_9 &= S_6 - S_7\\
D_4 &= t_0 - t_2 & T_{10} &= T_0 - T_2\\
D_5 &= t_1 - t_3 & T_{11} &= T_6 + T_8\\
D_6 &= t_4 + t_5 & T_{12} &= T_1 + T_3\\
D_7 &= t_4 - t_5 & T_{13} &= T_7 + T_9\\
D_8 &= D_3 + D_2 & T_{14} &= T_0 + T_4\\
D_9 &= D_0 + D_2 & T_{15} &= -T_6 + T_8\\
D_{10} &= D_9 - D_8 & T_{16} &= T_1 + T_5\\
D_{11} &= D_2 - D_3 & T_{17} &= -T_7 + T_9\\
D_{12} &= D_0 - D_1 & s_0 &= T_{10} + T_{11}\\
D_{13} &= D_4 + D_5 & s_1 &= T_{12} + T_{13}\\
 & & s_2 &= T_{14} + T_{15}\\
 & & s_3 &= T_{16} + T_{17}\\
 & & s_4 &= -T_{10} + T_{11}\\
 & & s_5 &= -T_{12} + T_{13}\\
 & & s_6 &= -T_{14} + T_{15}\\
 & & s_7 &= -T_{16} + T_{17}
\end{aligned}
$$

9-точечная циклическая свертка; 19 вещественных умножений, 74 вещественных сложений

$$
\mathbf{A} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1\\
1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1\\
1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1\\
1 & -1 & 1 & 0 & 0 & 0 & -1 & 1 & -1\\
1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0\\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1\\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1\\
1 & 0 & -1 & 0 & -1 & 1 & -1 & 1 & 0\\
1 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & -1\\
0 & 1 & -1 & 1 & -1 & 0 & -1 & 0 & 1\\
1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1\\
0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1\\
1 & 1 & -2 & 1 & 1 & -2 & 1 & 1 & -2
\end{bmatrix}
$$

$$
\mathbf{C} = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & -1\\
1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 1 & -1 & -1 & 2\\
1 & 0 & 1 & -1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1\\
1 & 0 & 1 & -1 & 0 & 1 & -1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 0 & -1\\
1 & -1 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & -1 & 0 & -1 & -1 & 2\\
1 & -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 1 & -1\\
1 & -1 & -1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & -1\\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & -1 & -1 & -1 & 2\\
1 & 1 & 0 & 1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & -1
\end{bmatrix}
$$

$$
\begin{bmatrix} 9G_0\\ 6G_1\\ 6G_2\\ 6G_3\\ 6G_4\\ 6G_5\\ 6G_6\\ 3G_7\\ 3G_8\\ 3G_9\\ 3G_{10}\\ 3G_{11}\\ 3G_{12}\\ 3G_{13}\\ 3G_{14}\\ 3G_{15}\\ 3G_{16}\\ 3G_{17}\\ 3G_{18} \end{bmatrix} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
-1 & -2 & -1 & 1 & 1 & 0 & 0 & 1 & 1\\
0 & 1 & 1 & -1 & -2 & -1 & 1 & 1 & 0\\
-1 & -1 & 0 & 0 & -1 & -1 & 1 & 2 & 1\\
1 & -2 & 1 & 1 & 1 & -2 & -2 & 1 & 1\\
-2 & 1 & 1 & 1 & -2 & 1 & 1 & 1 & -2\\
-1 & -1 & 2 & 2 & -1 & -1 & -1 & 2 & -1\\
-4 & -1 & 2 & 2 & -1 & -1 & 2 & 2 & -1\\
-2 & 1 & 1 & -2 & -2 & 1 & 4 & 1 & -2\\
2 & 2 & -1 & -4 & -1 & 2 & 2 & -1 & -1\\
-1 & 2 & 2 & -1 & -1 & 2 & 2 & -1 & -4\\
1 & 1 & -2 & -2 & 1 & 4 & 1 & -2 & -2\\
2 & -1 & -4 & -1 & 2 & 2 & -1 & -1 & 2\\
-1 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1\\
1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1\\
1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1\\
0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1\\
1 & 1 & -2 & 1 & 1 & -2 & 1 & 1 & -2
\end{bmatrix} \begin{bmatrix} g_0\\ g_1\\ g_2\\ g_3\\ g_4\\ g_5\\ g_6\\ g_7\\ g_8 \end{bmatrix}
$$

$$
\begin{aligned}
t_0 &= d_0 - d_6 & T_0 &= S_1 + S_2\\
t_1 &= d_1 - d_7 & T_1 &= S_4 + S_5\\
t_2 &= d_2 - d_8 & T_2 &= S_{14} + S_{15}\\
t_3 &= d_3 - d_6 & T_3 &= T_0 + T_1\\
t_4 &= d_4 - d_7 & T_4 &= S_1 + S_3\\
t_5 &= d_5 - d_8 & T_5 &= S_4 + S_6\\
t_6 &= d_0 + d_3 + d_6 & T_6 &= S_{13} + S_{15}\\
t_7 &= d_1 + d_4 + d_7 & T_7 &= -T_3 + S_7\\
t_8 &= d_2 + d_5 + d_8 & T_8 &= T_4 + T_5\\
t_9 &= t_0 + t_2 & T_9 &= S_{10} - T_6\\
t_{10} &= t_3 + t_5 & T_{10} &= S_8 + T_2 + T_7\\
D_0 &= t_6 + t_7 + t_8 & T_{11} &= T_8 + S_{11} + T_9\\
D_1 &= t_{10} + t_4 & T_{12} &= T_4 - T_5 + T_2\\
D_2 &= t_9 + t_1 & T_{13} &= T_7 + T_8 + S_9 + T_6\\
D_3 &= D_2 - D_1 & T_{14} &= T_3 + S_{12} + T_9 + T_2\\
D_4 &= t_{10} - t_4 & T_{15} &= T_0 - T_1 + T_6\\
D_5 &= t_9 - t_1 & T_{16} &= S_{16} - S_{18}\\
D_6 &= D_5 - D_4 & T_{17} &= S_{17} - S_{18}\\
D_7 &= t_3 & T_{18} &= S_0 + T_{16}\\
D_8 &= t_0 - t_3 & T_{19} &= S_0 - T_{16} - T_{17}\\
D_9 &= t_0 & T_{20} &= S_0 + T_{17}\\
D_{10} &= t_5 & s_0 &= T_{13} - T_{10} + T_{18}\\
D_{11} &= t_2 - t_5 & s_1 &= T_{14} - T_{11} + T_{19}\\
D_{12} &= t_2 & s_2 &= T_{15} - T_{12} + T_{20}\\
D_{13} &= -D_{11} + t_0 - t_4 & s_3 &= -T_{13} + T_{18}\\
D_{14} &= D_8 + t_5 - t_1 & s_4 &= -T_{14} + T_{19}\\
D_{15} &= -D_{14} + D_{13} & s_5 &= -T_{15} + T_{20}\\
D_{16} &= t_6 - t_8 & s_6 &= T_{10} + T_{18}\\
D_{17} &= t_7 - t_8 & s_7 &= T_{11} + T_{19}\\
D_{18} &= D_{16} + D_{17} & s_8 &= T_{12} + T_{20}
\end{aligned}
$$

Приложение Б

НАБОР МАЛЫХ БПФ-АЛГОРИТМОВ ВИНОГРАДА

Ниже приводятся малые БПФ-алгоритмы Винограда для длин $n = 2, 3, 4, 5, 7, 8, 9$ и 16. Алгоритмы записываются в виде матричного равенства

$$\mathbf{V} = \mathbf{CBAv}.$$

Матрица $\mathbf{B}$ является диагональной и выписываются только ее диагональные элементы. Матрицы $\mathbf{A}$ и $\mathbf{C}$ выписываются полностью. Кроме того, в обозначениях

$$\mathbf{a} = \mathbf{Av},\ \mathbf{b} = \mathbf{Ba},\ \mathbf{V} = \mathbf{Cb}$$

выписываются последовательности сложений, которыми можно заменить умножения на матрицы $\mathbf{A}$ и $\mathbf{B}$. Тривиальные сложения (сложения чисто вещественных или чисто мнимых чисел) отмечены звездочкой, но включены в полное число сложений. Они перестают быть тривиальными, если входные данные являются комплексными. Во всех алгоритмах мнимая единица j из диагональной матрицы передвинута в матрицу постсложений $\mathbf{C}$.

2-точечное преобразование Фурье; 0 (2) вещественных умножений, 2 вещественных сложения

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$
\begin{aligned}
a_0 &= v_0 + v_1 & B_0 &= 1 & V_0 &= b_0\\
a_1 &= v_0 - v_1 & B_1 &= 1 & V_1 &= b_1
\end{aligned}
$$

3-точечное преобразование Фурье; 2 (3) вещественных умножения, 6 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & -j \\ 1 & 1 & j \end{bmatrix}$$

$$
\begin{aligned}
a_2 &= v_1 + v_2 & \theta &= 2\pi/3 & V_0 &= b_0\\
a_1 &= v_1 - v_2 & B_0 &= 1 & T_0 &= b_0 + b_1\\
a_0 &= v_0 + a_2 & B_1 &= \cos\theta - 1 & V_1 &= T_0 - jb_2\\
 & & B_2 &= \sin\theta & V_2 &= T_0 + jb_2
\end{aligned}
$$

4-точечное преобразование Фурье; 0 (4) вещественных умножений, 8 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -j \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & j \end{bmatrix}$$

$$\begin{aligned} a_2 &= v_0 - v_2 \\ a_3 &= v_1 - v_3 \\ t_0 &= v_0 + v_2 \\ t_1 &= v_1 + v_3 \\ a_0 &= t_0 + t_1 \\ a_1 &= t_0 - t_1 \end{aligned} \qquad \begin{aligned} B_0 &= 1 \\ B_1 &= 1 \\ B_2 &= 1 \\ B_3 &= 1 \end{aligned} \qquad \left.\begin{aligned} V_0 &= b_0 \\ V_1 &= b_2 - jb_3 \\ V_2 &= b_1 \\ V_3 &= b_2 + jb_3 \end{aligned}\right]^{*}$$

5-точечное преобразование Фурье; 5 (6) вещественных умножений, 17 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & -j & j & 0 \\ 1 & 1 & -1 & -j & 0 & -j \\ 1 & 1 & -1 & j & 0 & j \\ 1 & 1 & 1 & j & -j & 0 \end{bmatrix}$$

$$\begin{aligned} t_0 &= v_1 + v_4 \\ t_1 &= v_2 + v_3 \\ a_4 &= v_3 - v_2 \\ a_5 &= v_1 - v_4 \\ a_1 &= t_0 + t_1 \\ a_2 &= t_0 - t_1 \\ a_3 &= a_4 + a_5 \\ a_0 &= v_0 + a_1 \end{aligned} \qquad \begin{aligned} &\theta = 2\pi/5 \\ B_0 &= 1 \\ B_1 &= \tfrac{1}{2}(\cos\theta + \cos 2\theta) - 1 \\ B_2 &= \tfrac{1}{2}(\cos\theta - \cos 2\theta) \\ B_3 &= \sin\theta \\ B_4 &= \sin\theta + \sin 2\theta \\ B_5 &= \sin 2\theta - \sin\theta \end{aligned} \qquad \left.\begin{aligned} V_0 &= b_0 \\ T_0 &= b_0 + b_1 \\ T_1 &= b_3 - b_4 \\ T_2 &= b_3 + b_5 \\ T_3 &= T_0 + b_2 \\ T_4 &= T_0 - b_2 \\ V_1 &= T_3 - jT_1 \\ V_2 &= T_4 - jT_2 \\ V_3 &= T_3 + jT_1 \\ V_4 &= T_4 + jT_2 \end{aligned}\right]^{*}$$

*) Тривиальные сложения

7-точечное преобразование Фурье; 8 (9) вещественных умножений, 36 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 1 & -1 \\ 0 & 1 & 1 & -1 & 1 & -1 & -1 \\ 0 & 1 & 0 & 1 & -1 & 0 & -1 \\ 0 & 0 & -1 & -1 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & -1 & 1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & -j & -j & -j & 0 \\ 1 & 1 & -1 & 0 & -1 & -j & j & 0 & j \\ 1 & 1 & 0 & -1 & 1 & j & 0 & -j & j \\ 1 & 1 & 0 & -1 & 1 & -j & 0 & j & -j \\ 1 & 1 & -1 & 0 & -1 & j & -j & 0 & -j \\ 1 & 1 & 1 & 1 & 0 & j & j & j & 0 \end{bmatrix}$$

$$\begin{aligned} t_0 &= v_1 + v_6 \\ t_1 &= v_1 - v_6 \\ t_2 &= v_2 + v_5 \\ t_3 &= v_2 - v_5 \\ t_4 &= v_4 + v_3 \\ t_5 &= v_4 - v_3 \\ t_6 &= t_2 + t_0 \\ a_4 &= t_2 - t_0 \\ a_2 &= t_0 - t_4 \\ a_3 &= t_4 - t_2 \\ t_7 &= t_5 + t_3 \\ a_7 &= t_5 - t_3 \\ a_6 &= t_1 - t_5 \\ a_8 &= t_3 - t_1 \\ a_1 &= t_6 + t_4 \\ a_5 &= t_7 + t_1 \\ a_0 &= v_0 + a_1 \end{aligned} \qquad \begin{aligned} &\theta = 2\pi/7 \\ B_0 &= 1 \\ B_1 &= \tfrac{1}{3}(\cos\theta + \cos 2\theta + \cos 3\theta) - 1 \\ B_2 &= \tfrac{1}{3}(2\cos\theta - \cos 2\theta - \cos 3\theta) \\ B_3 &= \tfrac{1}{3}(\cos\theta - 2\cos 2\theta + \cos 3\theta) \\ B_4 &= \tfrac{1}{3}(\cos\theta + \cos 2\theta - 2\cos 3\theta) \\ B_5 &= \tfrac{1}{3}(\sin\theta + \sin 2\theta - \sin 3\theta) \\ B_6 &= \tfrac{1}{3}(2\sin\theta - \sin 2\theta + \sin 3\theta) \\ B_7 &= \tfrac{1}{3}(\sin\theta - 2\sin 2\theta - \sin 3\theta) \\ B_8 &= \tfrac{1}{3}(\sin\theta + \sin 2\theta + 2\sin 3\theta) \end{aligned} \qquad \left.\begin{aligned} T_0 &= b_0 + b_1 \\ T_1 &= b_2 + b_3 \\ T_2 &= b_4 - b_5 \\ T_3 &= -b_2 - b_4 \\ T_4 &= b_6 + b_7 \\ T_5 &= b_8 - b_7 \\ T_6 &= -b_8 - b_6 \\ T_7 &= T_0 + T_1 \\ T_8 &= T_0 + T_2 \\ T_9 &= T_0 + T_3 \\ T_{10} &= T_4 + b_5 \\ T_{11} &= T_5 + b_5 \\ T_{12} &= T_6 + b_5 \\ V_0 &= b_0 \\ V_1 &= T_7 - jT_{10} \\ V_2 &= T_9 - jT_{12} \\ V_3 &= T_8 + jT_{11} \\ V_4 &= T_8 - jT_{11} \\ V_5 &= T_9 + jT_{12} \\ V_6 &= T_7 + jT_{10} \end{aligned}\right]^{*}$$

* Тривиальные сложения

8-точечное преобразование Фурье; 2 (8) вещественных умножений, 26 вещественных сложений

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -j & -j \\ 0 & 0 & 1 & 0 & 0 & -j & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & j & -j \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & -j & j \\ 0 & 0 & 1 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & j & j \end{bmatrix}$$

$t_0 = v_0 + v_4$
$a_3 = v_0 - v_4$
$t_1 = v_1 + v_5$
$t_2 = v_1 - v_5$
$t_3 = v_2 + v_6$
$a_4 = v_2 - v_6$
$t_4 = v_3 + v_7$
$t_5 = v_3 - v_7$
$t_6 = t_0 + t_3$
$a_2 = t_0 - t_3$
$t_7 = t_1 + t_4$
$a_3 = t_1 - t_4$
$a_4 = t_2 - t_5$
$a_7 = t_2 + t_5$
$a_0 = t_6 + t_7$
$a_1 = t_6 - t_7$

$\theta = 2\pi/8$
$B_0 = 1$
$B_1 = 1$
$B_2 = 1$
$B_3 = 1$
$B_4 = \cos\theta$
$B_5 = 1$
$B_6 = 1$
$B_7 = \sin\theta$

$V_0 = b_0$
$V_4 = b_1$
$V_2 = b_2 - jb_3$ *
$V_6 = b_2 + jb_3$
$T_0 = b_3 + b_4$
$T_1 = b_3 - b_4$
$T_2 = b_6 + b_7$
$T_3 = b_6 - b_7$
$V_1 = T_0 - jT_2$ *
$V_5 = T_0 + jT_2$
$V_7 = T_1 - jT_3$
$V_3 = T_1 + jT_3$

* Тривиальные сложения.

9-точечное преобразование Фурье; 10 (11) вещественных умножений, 44 вещественных сложения

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 1 & 1 & 1 & 0 & 0 & j & j & j \\ 1 & 0 & -1 & 1 & 0 & -1 & j & 0 & -j & -j & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & j & 0 & 0 & 0 \\ 1 & 0 & -1 & 1 & -1 & 0 & -j & 0 & j & j & -j \\ 1 & 0 & -1 & 1 & -1 & 0 & -j & 0 & -j & -j & j \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & -j & 0 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 & -1 & j & 0 & j & j & 0 \\ 1 & 0 & -1 & 1 & 1 & 1 & 0 & 0 & -j & -j & -j \end{bmatrix}$$

$t_0 = v_1 + v_8$
$t_1 = v_2 + v_7$
$a_1 = v_3 + v_6$
$t_2 = v_4 + v_5$
$a_2 = t_0 + t_1 + t_2$
$t_3 = v_1 - v_8$
$t_4 = v_7 - v_2$
$t_5 = v_3 - v_6$
$a_7 = v_4 - v_5$
$a_6 = t_3 + t_4 + a_7$
$a_3 = t_0 - t_1$
$a_4 = t_1 - t_2$
$a_8 = t_4 - t_5$
$a_9 = t_4 - a_7$
$a_0 = v_0 + a_1 + a_2$
$a_5 = -a_4 - a_3$
$a_{10} = -a_8 + a_9$

$\theta = 2\pi/9$
$B_0 = 1$
$B_1 = \frac{1}{2}$
$B_2 = -\frac{1}{2}$
$B_3 = \frac{1}{3}(2\cos\theta - \cos 2\theta - \cos 4\theta)$
$B_4 = \frac{1}{3}(\cos\theta + \cos 2\theta - 2\cos 4\theta)$
$B_5 = \frac{1}{3}(\cos\theta - 2\cos 2\theta + \cos 4\theta)$
$B_6 = \sin 3\theta$
$B_7 = \sin 3\theta$
$B_8 = -\sin\theta$
$B_9 = -\sin 4\theta$
$B_{10} = -\sin 2\theta$

$T_0 = -b_3 - b_4$
$T_1 = b_5 - b_4$
$T_2 = -b_4 - b_5$
$T_3 = b_8 - b_{10}$
$T_4 = b_9 + b_2 + b_2$
$T_5 = T_4 - b_1$
$T_6 = T_4 + b_2$
$T_7 = T_3 - T_0$
$T_8 = T_1 + T_5$
$T_9 = T_0 - T_1 + T_5$
$T_{10} = b_7 - T_2$
$T_{11} = b_7 - T_3$
$T_{12} = b_7 + T_2 + T_3$
$V_0 = b_0$
$V_1 = T_7 + jT_{10}$ *
$V_2 = T_8 - jT_{11}$
$V_3 = T_6 + jb_6$
$V_4 = T_9 + jT_{12}$
$V_5 = T_9 - jT_{12}$
$V_6 = T_6 - jb_6$
$V_7 = T_8 + jT_{11}$
$V_8 = T_7 - jT_{10}$

*) Тривиальные сложения.

16-точечная циклическая свертка; 10 (18) вещественных умножений, 74 вещественных сложений

$$\mathbf{A} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0
\end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & j & 0 & j & j & j & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & j & 0 & j & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 & 0 & 0 & -j & 0 & j & j & 0 & -j \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & j & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & j & 0 & -j & j & 0 & -j \\
0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -j & 0 & j & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & -j & 0 & -j & j & j & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & j & 0 & j & -j & -j & 0 \\
0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & j & 0 & -j & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & -j & 0 & j & -j & 0 & j \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -j & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 & 0 & 0 & j & 0 & -j & -j & 0 & j \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -j & 0 & -j & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & -j & 0 & -j & -j & -j & 0
\end{bmatrix}$$

$$\begin{aligned}
t_0 &= v_0 + v_8 \\
t_1 &= v_4 + v_{12} \\
t_2 &= v_2 + v_{10} \\
t_3 &= v_2 - v_{10} \\
t_4 &= v_6 + v_{14} \\
t_5 &= v_6 - v_{14} \\
t_6 &= v_1 + v_9 \\
t_7 &= v_1 - v_9 \\
t_8 &= v_3 + v_{11} \\
t_9 &= v_3 - v_{11} \\
t_{10} &= v_5 + v_{13} \\
t_{11} &= v_5 - v_{13} \\
t_{12} &= v_7 + v_{15} \\
t_{13} &= v_7 - v_{15} \\
t_{14} &= t_0 + t_1 \\
t_{15} &= t_2 + t_4 \\
t_{16} &= t_{14} + t_{15} \\
t_{17} &= t_6 + t_{10} \\
t_{18} &= t_6 - t_{10} \\
t_{19} &= t_8 + t_{12} \\
t_{20} &= t_8 - t_{12} \\
t_{21} &= t_{17} + t_{19} \\
a_{16} &= t_7 + t_{13} \\
a_8 &= t_7 - t_{13} \\
a_{17} &= t_{11} + t_9 \\
a_9 &= t_{11} - t_9 \\
a_0 &= t_{16} + t_{21} \\
a_1 &= t_{16} - t_{21} \\
a_2 &= t_{14} - t_{15} \\
a_3 &= t_0 - t_1 \\
a_4 &= v_0 - v_8 \\
a_5 &= t_{18} - t_{20} \\
a_6 &= t_3 - t_5 \\
a_7 &= a_8 + a_9 \\
a_{10} &= t_{19} - t_{17} \\
a_{11} &= t_4 - t_2 \\
a_{12} &= v_{12} - v_4 \\
a_{13} &= t_{18} + t_{20} \\
a_{14} &= t_3 + t_5 \\
a_{15} &= a_{16} + a_{17}
\end{aligned}$$

$$\begin{aligned}
&\theta = 2\pi/16 \\
&B_0 = 1 \\
&B_1 = 1 \\
&B_2 = 1 \\
&B_3 = 1 \\
&B_4 = 1 \\
&B_5 = \cos 2\theta \\
&B_6 = \cos 2\theta \\
&B_7 = \cos 3\theta \\
&B_8 = \cos\theta + \cos 3\theta \\
&B_9 = -\cos\theta + \cos 3\theta \\
&B_{10} = 1 \\
&B_{11} = 1 \\
&B_{12} = 1 \\
&B_{13} = -\sin 2\theta \\
&B_{14} = -\sin 2\theta \\
&B_{15} = -\sin 3\theta \\
&B_{16} = -\sin\theta + \sin 3\theta \\
&B_{17} = -\sin\theta - \sin 3\theta
\end{aligned}$$

$$\begin{aligned}
T_0 &= b_3 + b_5 \\
T_1 &= b_3 - b_5 \\
T_2 &= b_{11} + b_{13} \\
T_3 &= b_{13} - b_{11} \\
T_4 &= b_4 + b_6 \\
T_5 &= b_4 - b_6 \\
T_6 &= b_8 - b_7 \\
T_7 &= b_9 - b_7 \\
T_8 &= T_4 + T_6 \\
T_9 &= T_4 - T_6 \\
T_{10} &= T_5 + T_7 \\
T_{11} &= T_5 - T_7 \\
T_{12} &= b_{12} + b_{14} \\
T_{13} &= b_{12} - b_{14} \\
T_{14} &= b_{15} + b_{16} \\
T_{15} &= b_{15} - b_{17} \\
T_{16} &= T_{12} + T_{14} \\
T_{17} &= T_{12} - T_{14} \\
T_{18} &= T_{13} + T_{15} \\
T_{19} &= T_{13} - T_{15} \\
V_0 &= b_0 \\
V_1 &= T_8 + jT_{16} \\
V_2 &= T_0 + jT_2 \\
V_3 &= T_{11} - jT_{19} \\
V_4 &= b_2 + jb_{10} \\
V_5 &= T_{10} + jT_{18} \\
V_6 &= T_1 + jT_3 \\
V_7 &= T_9 - jT_{17} \\
V_8 &= b_1 \\
V_9 &= T_9 + jT_{17} \\
V_{10} &= T_1 - jT_3 \\
V_{11} &= T_{10} - jT_{18} \\
V_{12} &= b_2 - jb_{10} \\
V_{13} &= T_{11} + jT_{19} \\
V_{14} &= T_0 - jT_2 \\
V_{15} &= T_8 - jT_{16}
\end{aligned}$$

*) Тривиальные сложения.

ЛИТЕРАТУРА

Монографии

[1] Nussbaumer, H. J., Fast Fourier Transform and Convolution Algorithms, 2nd ed., Springer-Verlag, Berlin, 1982. [Имеется перевод: Нуссбаумер Г. Быстрое преобразование Фурье и алгоритмы вычисления свертки. — М.: Радио и связь, 1985.]

[2] Kronsjö L. I., Algorithms: Their Complexity and Efficiency, John Wiley, New York, 1979.

[3] Aho, A. V., J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974. [Имеется перевод: Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979.]

[4] Elliott, D. F., and R. Rao, Fast Transforms: Algorithms, Analyses, and Applications, Academic Press, New York, 1983.

Сборники статей

[1] McClellan, J. H., and C. M. Rader, Number Theory in Digital Signal Processing, Prentice-Hall, Englewood Cliffs, N. J., 1979. [Имеется перевод: Макклеллан Дж. Х., Редер Ч. М. Применение теории чисел в цифровой обработке сигналов. — М.: Радио и связь, 1983.]

Глава 1

[1] Cooley, J. W., P. A. W. Lewis, and P. D. Welch, Historical Notes on the Fourier Transform, IEEE Trans. Audio Electroacoust. AU—15 (1967) : 76—79.

[2] Oppenheim, A. V. and R. W. Shafer, Digital Signal Processing, Prentice-Hall, Englewood Cliffs, N. J., 1975.

[3] Rabiner, L. R., and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Englewood Cliffs, 1975. [Имеется перевод; Рабинер Л. Р., Гоулд Б. Теория и применение цифровой обработки сигналов. — М. Мир., 1978.]

[4] Winograd S., A New Algorithm for Inner Product, IEEE Trans. Comp. C-17 (1968): 693—694.

[5] Cooley, J. W., and J. W. Tukey, An Algorithm for the Machine Computation of Complex Fourier Series, Math. Comp. 19 (1965): 297—301.

[6] Stockham, T. G., High Speed Convolution and Correlation, Spring Joint Comput. Conf., AFIPS Conf. Proc. 28 (1966): 229—233.

[7] Good, I. J., The Interaction Algorithm and Practical Fourier Analysis, J. Royal Statist. Soc., Ser. B 20 (1958); 361—375; addendum, 22 (1960): 372—375.

[8] Thomas, L. H., Using a Computer to Solve Problems in Physics, in Applications of Digital Computers, Ginn and Co., Boston, Mass., 1963.

[9] Winograd, S., On Computing the Discrete Fourier Transform, Proc. Nat. Acad. Sci. USA, 73 (1976): 1005—1006.

[10] Winograd, S., On Computing the Discrete Fourier Transform, Math. Comp., 32 (1978): 175—199. [Имеется перевод: см. сб. статей [1], с. 117 русск. изд.]

[11] Butler, J., and R. Lowe, Beam-forming Matrix Simplifies Design of Electronically Scanned Antennas, Electronic Design 9 (1961): 170—173.

[12] Agarwal, R. C., and J. W. Cooley, New Algorithms for Digital Convolution, IEEE Trans. Acoust., Speech, Signal Proc. ASSP-25 (1977): 392—410. [Имеется перевод: см. сб. статей [1], с. 91 русск. изд.]

[13] Levinson, N., The Wiener RMS Error Criteria in Filter Design and Prediction, J. Math. Phys. 25 (1947): 261—278.

Глава 2

[1] Birkhoff, G., and S. MacLane, A Survey of Modern Algebra, Macmillan, New York, 1941; rev. ed., 1953.

[2] Van der Waerden, B. L., Modern Algebra, 2 vols. trans. by F. Blum and T. J. Benac, Frederic Ungar Publishing Co., New York, 1949, 1953. [Имеется перевод нем. изд. 1967 г.: Ван дер Варден Б. Л. Алгебра. — М.: Наука, 1976; 2-е изд. 1979.]

[3] Thrall, R. M. and L. Tornheim, Vector Spaces and Matrices, John Wiley, New York, 1957.

[4] Fraleigh, J. B., A First Course in Abstract Algebra, 2nd. ed. Addison-Wesley, Reading, Mass. 1976.

[5] Strang, G., Linear Algebra and Its Applications, 2nd ed., Academic Press, New York, 1980. [Имеется перевод изд. 1976 г.: Стренг Г. Линейная алгебра и ее применения. — М.: Мир, 1980.]

[6] Baumslag, B. and B. Chandler, Theory and Problems of Group Theory, Schaum's Outline Series, McGraw-Hill, New York, 1968.

[7] Pollard, J. M., The Fast Fourier Transform in a Finite Field, Math. Comp. 25 (1971): 365—374. [Имеется перевод: см. сб. статей [1], с. 147 русск. изд.]

Глава 3

[1] Agarwal, R. C., and J. W. Cooley, New Algorithms for Digital Convolution, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—25 (1977): 392—410. [Имеется перевод: см. сб. статей [1], с. 91 русск. изд.]

[2] Winograd S., On Computing the Discrete Fourier Transform, Math. Comp. 32 (1978): 175—199. [Имеется перевод: см. сб. статей [1], с. 117 русск. изд.]

[3] Winograd, S., Some Bilinear Forms Whose Multiplicative Complexity Depends on the Feild of Constants, Math. Syst. Theor. 10 (1977): 169—80. [Имеется перевод: см. сб. статей [1], с. 225 русск. изд.]

[4] Winograd S., Arithmentic Complexity of Computations, CBMS-NSF Regional Conf. Series Appl. Math., Siam Publications 33, 1980.

Глава 4

[1] Cooley, J. W. and J. W. Tukey, An Algorithm for the Machine Computation of Complex Fourier Series, Math. Comp. 19 (1965): 297—301.

[2] Singleton, R. C., An Algorithm for Computing the Mixel Radix Fast Fourier Transform, IEEE Trans. Audio Electroacoust. AU-17 (1969): 93—103.

[3] Good, I. J., The Interaction Algorithm and Practical Fourier Analysis, J. Royal Statist. Soc., Ser. B 20 (1958): 36—375; addendum, 22 (1960): 372—375.

[4] Thomas, L. H., Using a Computer to Solve Problems in Physics, Applications of Digital Computers, Ginn and Co., Boston, Mass. 1963.

[5] Good, I. J., The Relationship between Two Fast Fourier Transform, IEEE Trans. Comp. C-20 (1971): 310—317. [Имеется перевод: см. сб. статей [1], с. 136 русск. изд.]

[6] Rader, C. M., and N. M. Brenner, A New Principle for Fast Fourier Transformation, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—24 (1976): 264—265.

[7] Preuss, R. D., Very Fast Computation of the Radix-2 Discrete Fourier Transform, IEEE Trans. Aucoust., Speech, Signal Proc. ASSP-30 (1982): 595—607.

[8] Rader, C. M., Discrete Fourier Transforms When the Number of Data Samples Is Prime, Proc. IEEE 56 (1968): 1107—1108. [Имеется перевод: см. сб. статей [1], с. 89 русск. изд.]

[9] Bluestein, L. I., A. Linear Filtering Approach to the Computation of the Discrete Fourier Transform, IEEE Trans. Audio Electroacoust. AU-18 (1970): 451—455.

[10] Winograd, S., On Computing the Discrete Fourier Transform, Math. Comp. 32 (1978): 175—199.
[11] Winograd S., On Computing The Discrete Fourier Transform, Math Comp. 32 (1978): 175—199. [Имеется перевод: см. сб. статей [1], с. 117 русск. изд.]
[12] Johnson, H. W., and C. S. Burrus, Large DFT Modules: 11, 13, 17, 19 and 25, Tech. Rep. 8105, Dep. Elec. Eng., Rice University, Houston, Texas, 1981.
[13] Goertzel, G., An Algorithm for the Evaluation of Finite Trigonometric Series, Amer. Math. Monthly 65 (1968): 34—35.
[14] Sarwate, D. V., Semi-Fast Fourier Transforms Over GF (2^m), IEEE Trans. Comp. C-27 (1978): 283—284.

Глава 5

[1] Ore, O., Number Theory and its History, McGraw-Hill, New York, 1948.
[2] Hardy, G. H., and E. M. Wright, The Theory of Numbers, Oxford University Press, Oxford, England, 1960.
[3] Birkhoff, G., and S. MacLane, A Survey of Modern Algebra, Macmillan, New York, 1941; rev. ed., 1953.
[4] Van der Waerden, B. L., Modern Algebra, 2 vols. transl. by F. Blum and T. J. Benac, Frederic Ungar Publishing Co., New York, 1949, 1953. [Имеется перевод: см. гл. II [2].]
[5] Berlecamp, E. R., Algebraic Coding Theory, McGraw-Hill, New York, 1968. [Имеется перевод: Берлекэмп Э. Алгебраическая теория кодирования. — М.: Мир, 1971.]
[6] McClellan, J. H., and C. M. Rader, Number Theory in Digital Signal Processing, Prentice-Hall, Englewood Cliffs, N. J., 1979. [Имеется перевод: см. сб. статей [1].]

Глава 6

[1] Rader, C. M., Discrete Convolution via Mersenne Transforms, IEEE Trans. Comp. C—21 (1972): 1269—1273.
[2] Agarwal, R. C., and C. S. Burrus, Fast Digital Convolution Using Fermat Transforms, Soutwest IEEE Conf. Rec. Houston (1973); 538—543.
[3] Agarwal, R. C., and C. S. Burrus, Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering, IEEE Trans. Acoust., Speech, Signal Proc. ASSP 22 (1974): 87—97. [Имеется перевод: см. сб. статей [1], с. 156 русск. изд.]
[4] Agarwal, R. C., and C. S. Burrus, Number Theoretic Transforms to Implement Fast Digital Convolution, Proc. IEEE 63 (1975): 550—560.
[5] McClellan, J. H., Hardware Realization of a Fermat Number Transform, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—24 (1976): 216—225.
[6] Leibowitz, L. M., A Simplified Binary Arithmetic for the Fermat Number Transform, IEEE Trans. Acoust., Speech, Signal Proc. ASSSP—24 (1976): 356—359. [Имеется перевод: см. сб. статей [1], с. 202 русск. изд.]
[7] Reed, I. S., and T. K. Truong, The Use of Finite Fields to Compute Convolutions, IEEE Trans. Inf. Theor. IT—21 (1975): 208—213. [Имеется перевод: см. сб. статей [1], с. 207 рус. изд.].
[8] Nussbaumer, H. J., Digital Filtering Using Complex Mersenne Transforms, IBM Journal Res. Devel. 20 (1976): 498—504. [Имеется перевод: см. сб. статей [1], с. 216 русск. изд.]
[9] Rice, B., Winograd Convolution Algorithms over Finite Fields, Congressus Numberatium 29 (1980): 827—857.
[10] Preparata, F. P. and D. W. Sarwate, Computational Complexity of Fourier Transforms over Finite Fields, Math. Comp. 31 (1977): 740—751.

[11] Chevillat, P. R., Transform-Domain Filtering with Number Theoretic Transforms, and Limited Word Length, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—26 (1978): 284—290
[12] Szabo, N. S., and R. I. Tanaka, Residue Arithmetic and Its Application to Computer Technology, McGraw-Hill, New York, 1967.
[13] Jenkins, W. K. and B. J. Leon, The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters, IEEE Trans. Circuits Syst. CAS—24 (1977): 191—201.

Глава 7

[1] Nussbaumer, H. J., Digital Filtering Using Polynomial Transforms, Electron. Lett. 13 (1977): 386—387.
[2] Blahut, R. E., Fast Convolution of Rational Sequences, Abstr. 1983 IEEE Internat. Sympos. Inf. Theor., St. Jovite, Quebec, Canada, 1983.
[3] Beth, T., W. Fumy, and R. Muhlfeld, On Algebraic Discrete Fourier Transforms, Abstr. 1982 IEEE Internat. Sympos. Inf. Theor., Les Arcs, France, 1982.
[4] Arambepola, B., and P. J. W. Rayner, Efficient Transforms for Multidimensional Convolutions, Electron. Lett. 15 (1979): 189—190.
[5] Agarwal, R. C., and J. W. Cooley, New Algorithms for Digital Convolution, IEEE Trans. ASSP—25 (1977): 392—410. [Имеется перевод: см. сб. статей [1], с. 91 русск. изд.]
[6] Nussbaumer, H. J., New Algorithms for Convolution and DFT Based on Polynomial Transforms, Proc. 1978 IEEE Internat. Conf. Acoust., Speech, Signal Proc. (1978): 638—641.
[7] Pitas, I., and M. G. Strintzis, On the Multiplicative Complexity of Two-Dimensional Fast Convolution Methods, Abstr. 1982 IEEE Internat. Sympos. Inf. Theor., Les Arcs, France, 1982.
[8] Winograd, S., On Computing the Discrete Fourier Transform, Math. Comp. 32 (1978): 175—199. [Имеется перевод: см. сб. статей [1], с. 117 русск. изд.]

Глава 8

[1] Rivard, G. E., Direct Fast Fourier Transform of Bivariate Functions, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—25 (1977): 250—252.
[2] Harris, D. B., J. H. McClellan, D. S. K. Chan, and H. W. Schuessler, Vector Radix Fast Fourier Transform, Rec. 1977 IEEE. Internat. Conf. Acoust., Speech, Signal Proc. (1977): 548—551.
[3] Mersereau, R., and T. C. Speake, A Unified Treatment of Cooley—Tukey Algorithms for the Evaluation of the Multidimensional DFT, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—29 (1981): 1011—1018.
[4] Winograd, S., On Computing the Discrete Fourier Transform, Proc. Nat. Acad. Sci. USA 73 (1976): 1005—1006.
[5] Kolba, D. P., and T. W. Parks, A Prime Factor FFT Algorithm Using High Speed Convolution, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—25 (1977): 281—294. [Имеется перевод: см. сб. статей [1], с. 72 русск. изд.]
[6] Silverman, H. F., An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA), IEEE Trans. Acoust., Speech, Signal Proc. ASSP—25 (1977): 152—165.
[7] Johnson, H. W., and C. S. Burrus, The Design of Optimal DFT Algorithms Using Dynamic Programming, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—31 (1983): 378—387.
[8] Nussbaumer, H. J., and P. Quandalle, Fast Computation of Discrete Fourier Transforms Using Polynomial Transforms, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—27 (1979): 169—181.

[9] Nussbaumer, H. J., and P. Quandalle, New Polynomial Transform Algorithms for Fast DFT Computations, Proc. IEEE 1979 Internat. Acoust., Speech, Signal Proc. Conf. (1979): 510—513.
[10] Auslander, L., E. Feig, and S. Winograd, New Algorithms for the Multidimensional Discrete Fourier Transform, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—31 (1983): 388—403.

Глава 9

[1] Stockham, T. G., High Speed Convolution and Correlation, 1966 Spring Joint Comput. Conf., AFIPS Proc. 28 (1966): 229—233.
[2] Agarwal, R. C., and C. S. Burrus, Fast One-Dimensional Digital Convolution by Multidimensional Techniques, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—22 (1974): 1—10. [Имеется перевод: см. сб. статей [1], с. 172 русск. изд.]
[3] DuBois, E. and A. N. Venetsanopoulos, Convolution Using a Conjugate Symmetry Property for the Generalized Discrete Fourier Transform, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—26 (1978): 165—170.
[4] Winograd, S., On the Complexity of Symmetric Filters, Proc. Internat. Sympos. Circuits Syst. Tokyo (1979): 262—265.
[5] Winograd S., Arithmetic Complexity of Computations, CBMS—NSF Regional Conf. Ser. Appl. Math. Siam Publications 33, 1980.
[6] Hopcroft, J. E., and J. Musinski, Duality Applied to the Complexity of Matrix Multiplication and Other Bilinear Forms, SIAM J. Comput. 2 (1973): 159—173.
[7] Rader, C. M., An Improved Algorithm for High Speed Autocorrelation with Applications to Spectral Estimation, IEEE Trans. Audio Electroacoust., AU—18 (1970): 439—441.
[8] Burrus, C. S., and P. W. Eschenbacher, An In-Place Prime Factor FFT Algorithm, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—29 (1979):806—817.
[9] Liu, B., and F. Mintzer, Calculation of Narrow-Band Spectra by Direct Decimation, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—26 (1978): 529—534.
[10] Crochiere, R. E., and L. R. Rabiner, Interpolation and Decimation of Digital Signals — A Tutorial Review, Proc. IEEE 69 (1981): 330—331.
[11] Kolba, D. P., and T. W. Parks, A Prime Factor FFT Algorithm Using High Speed Convolution. IEEE Trans. Acoust., Speech, Signal Proc. ASSP—25 (1977): 281—294. [Имеется перевод: см. сб. статей [1], с. 72 русск. изд.]
[12] Silverman, H. F., An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA), IEEE Trans. Acoust., Speech, Signal Proc. ASSP—25 (1977): 152—165.
[13] Morris, P. L., A Comparative Study of Time Efficient FFT and WFTA Programs for General Purpose Computers, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—26 (1978): 141—150.
[14] Nawab, H., and J. H. McClellan, Bounds on the Minimum Number of Data Transfers in WFTA and Programs, IEEE Trans. Acoust. Speech, Signal Proc. ASSP—27 (1979): 393—398.
[15] Patterson, R. W., and J. H. McClellan, Fixed-Point Error Analysis of Winograd Fourier Transform Algorithms, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—26 (1978): 447—455.
[16] Panda, G., R. N. Pal, and B. Chatterjee, Error Analysis of Good—Winograd Algorithm Assuming Correlated Truncation Errors, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—31 (1983): 508—512.

Глава 10

[1] Knuth, D. E., The Art of Computer Programming, Vol. 1; Fundamental Algorithms, Addison-Wesley, Reading, Mass., 1968. [Имеется перевод: Кнут Д. Искусство программирования, том 1.—М.: Мир, 1976.]

[2] Aho, A. V., J. E. Horcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974. [Имеется перевод: см. Монографии [3].]
[3] Williams, J. M. J., Algorithm 232; Heapsort, Comm ACM 7 (1964): 347—348.
[4] Hoare, C. A. R., Quicksort, Comp. J. 5, 1962.
[5] Steiglitz, K., An Introduction to Discrete Systems, John Wiley, New York, 1974.
[6] Fiduccia, C. M., Polynomial Evaluation via the Division Algorithm — The Fast Fourier Tranform Revisited, Proc. 4th Annual ACM Symp. Theory Comput. (1972): 88—93.
[7] Eklundh, J. O., A Fast Computer Method for Matrix Transposing, IEEE Trans. Comp. C—21 (1972): 801—803.
[8] Winograd, S., Signal Processing and Complexity of Computation, Proc. Int. Conf. Acoust., Speech, Singnal Proc. (1980): 94—101.
[9] Volder, J. E., IRE Trans. Electr. Comp. EC—8 (1959): 330—334.
[10] Walther, J. S., Proc. 1971 Spring Joint Comp. Conf. (1971): 379—385.
[11] Blahut, R. E., and D. E. Waldecker, Half-Angle Sine-Cosine Generator, IBM Tech. Disclosure Bull. 13, no. 1 (1970): 222—223.

Глава 11

[1] Levinson, N., The Wiener RMS Error Criterion in Filter Design and Prediction, J. Math. Phys. 25 (1947): 261—278.
[2] Durbin, J., The Fitting of Time—Series Models, Rev. Internat. Statist. Inst. 23 (1960): 233—244.
[3] Trench, W. F., An Algorithm for the Inversion of Finite Toeplitz Matrices, J. SIAM 12, no. 3 (1964): 512—522.
[4] Berlecamp, E. R., Algebraic Coding Theory, McGraw-Hill, New York, 1968. [Имеется перевод: см. гл. V [5].]
[5] Zohar, S., The Solution of a Toeplitz Set of Linear Equations, J. Assoc. Comput. Math. 21 (1974): 272—276.
[6] Massey, J. L., Shoft—Register Synthesis and BCH Decoding, IEEE Trans. Inf, Theor. IT—15 (1969): 122—127.
[7] Welch, L. R., and R. A. Scholtz, Continue Fractions and Berlecamp's Algorithm, IEEE Trans. Inf. Theor. IT—25 (1979): 19—27.
[8] Blahut, R. E., Theory and Practice of Error Control Codes, Addison-Wesley, Reading, Mass., 1983. [Имеется перевод: Блейхут Р. Теория и практика кодов, контролирующих ошибки. — М.: Мир, 1986.]
[9] Sugiyama, Y., M. Kasahara, S. Hirasawa, and T. Namekawa, A Method for Solving Key Equation for Decoding Goppa Codes, Inf. Control 27 (1975): 87—99.
[10] Brent, R. P., F. G. Gustavson, and D. Y. Y. Yun, Fast Solution of Toeplitz Systems of Equations and Computation of Pade Approximants, J. Algorithms 1 (1980): 259—295.
[11] Vieira, A. and T. Kailath, On Another Approach to the Schur—Cohn Criterion, IEEE Trans. Circuits Syst. CAS—24, no. 4 (1977): 218—220.
[12] Wiggins, R. A., and E. A. Robinson, Recursive Solution to the Multichannel Filtering Problem, J. Geophys. Res. 70 (1965): 1885—1891.
[13] Dickinson, B. W., Efficient Solution of Linear Equations with Banded Toeplitz Matrices, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—27 (1979): 421—423.
[14] Dickinson, B. W., M. Morf, and T. Kailath, A Minimal Realization Algorithm for Matrix Sequences, IEEE Trans. Automatic Control AC—19 (1974): 31—38.
[15] Friedlander, B., M. Morf, T. Kailath, and L. Ljung, New Inversion Formulas for Matrices Classified in Terms of Their Distance from Toeplitz Matrices, Linear Algebra and Its Application 27 (1979): 31—60.

[16] Morf, M., B. W. Dickinson, T. Kailath, and A. C. G. Vieira, Efficient Solution of Covariance Equations for Linear Prediction, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—25 (1977): 429—433.
[17] Monden, Y., and S. Arimoto, Generalized Rouche's Theorem and Its Application to Multivariate Autoregressions, IEEE Trans. Acoust., Speech, Signal Proc. ASSP—28 (1980): 733—738.

Глава 12

[1] Viterbi, A. J., Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm, IEEE Trans. Inf. Theor. IT—13 (1967): 260—269.
[2] Wozencrafi, J. M., Sequential Decoding for Reliable Communication, 1957 Nat. IRE Conv. Rec. 5, part 2 (1957): 11—25.
[3] Wozencraft, J. M., and B. Reiffen, Sequential Decoding, MIT Press, Cambridge, Mass., 1961.
[4] Fano, R. M., A. Heuristic Discussion of Probabilistic Decoding, IEEE Trans. Inf. Theor. IT—9 (1963): 64—74.
[5] Зигангиров К. Ш. Некоторые процедуры последовательного декодирования. — Проблемы передачи информации, 1966, вып. 2, с. 13—25.
[6] Jelinek, F. A. Fast Sequential Decoding Algorithm Using a Stack, IBM J. Res. Devel. 13 (1969): 675—685.
[7] Jelinek, F., Probabilistic Information Theory, McGraw-Hill, New York, 1968.
[8] Forney, G. D., Convolutional Codes III: Sequential Decoding, Inform. Contr. 25 (1974): 267—297.
[9] Chevillat, P. R., and D. J. Costello, Jr., An Analysis of Sequential Decoding for Specific Time-Invariant Convolutional Codes, IEEE Trans. Inf. Theor. IT—24 (1978): 443—451.
[10] Haccoun, D., and M. J. Ferguson, Generalized Stack Algoritms for Decoding Convolutional Codes, IEEE Trans. Inf. Theor. IT—21 (1975): 638—651.
[11] Gallager, R. G., Information Theory and Reliable Communications, John Wiley, New York, 1968. [Имеется перевод: Галлагер Р. Теория информации и надежная связь. — М.: Сов. радио, 1974.]
[12] Rader, C. M., Memory Management in a Viterbi Decoder, IEEE Trans. Communicat. COM—29 (1981): 1399—1401.
[13] Jacobs, I. M., and E. R. Berlecamp, A Lower Bound to the Distribution of Computations for Sequential Decoding, IEEE Trans. Inf. Theor. IT—13 (1967): 167—174.
[14] Savage, J. E., Sequental Decoding—The Computation Problem, Bell Syst. Tech. J. 45 (1966): 149—175.
[15] Jelinek, F., An Upper Bound on Moments of Sequential Decoding Effort, IEEE Trans. Inf. Theor. IT—15 (1969). 140—149.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ