1976
Сложность вычислений

Микаэль О. Рабин

Еврейский университет в Иерусалиме

Тьюринговская премия ACM 1976 г. была вручена Микаэлю О. Рабину и Дане С. Скотту на ежегодной конференции ACM в Хьюстоне 20 октября. Представляя лауреатов, Бернард А. Галлер, председатель комитета по премиям Тьюринга, зачитал следующее:

«Премия Тьюринга в этом году вручаетяс Микаэлю Рабину и Дане Скотту за самостоятельный и совместный вклад, который не только наметил направление в теоретической информатике, но и явился образцом ясности и изящества для целой области исследований.

Работа Рабина и Скотта 1959 г. «Конечные автоматы и задачи их разрешения» стала классической в теории формальных языков, она представляет собой одно из самых лучших введений в эту область. Эта статья одновременно является обзором и оригинальной исследовательской работой, она технически проста и математически безупречна. Она рекомендована даже студентам младших курсов!

В последующие годы Рабин и Скотт провели самостоятельные исследования, которые подтвердили уровень их ранней работы. Применение Рабином теории автоматов к логике и разработка Скоттом непрерывной семантики для языков программирования — два примера работ, сочетающих глубину и масштабность: первая применяет информатику к математике и вторая — математику к информатике.

Рабин и Скотт показали нам, насколько хорошо математики могут помочь ученому понять свой собственный предмет. Их работа — один из великолепных образцов творческой прикладной математики».

Это была цитата из официального заключения Комитета по премиям Тьюринга, но существует менее формальная сторона сегодняшней презентации. Я хотел бы, чтобы вы поняли, что лауреаты этой премии — реальные люди, сделавшие великолепную работу, но очень похожие на всех присутствующих здесь сегодня. Профессор Микаэль Рабин родился в Германии и эмигрировал со своими родителями в Израиль в 1935 г. Он получил магистерскую степень по математике в Еврейском уни-

верситете и позднее — степень доктора философии по математике в Принстонском университете. После получения степени доктора философии он был преподавателем математики в Принстонском университете и членом исследовательского института в Принстоне. С 1958 г. он являлся преподавателем Еврейского университета в Иерусалиме. С 1972 по 1975 г. он был также ректором Еврейского университета. Ректор избирается Сенатом университета и является его научным руководителем.

Профессор Дана Скотт получил свою степень доктора философии в Принстонском университете в 1958 г. С тех пор он преподавал в Чикагском университете, Калифорнийском университете в Беркли, Станфордском университете, Амстердамском университете, Принстонском университете и Оксфордском университете в Англии.

Профессор Рабин выступит с лекцией «Вычислительная сложность», профессор Скотт — «Логика и языки программирования».

Статья Рабина помещена ниже, статья Скотта начинается на с. 65.

Очерчивается область исследований в теории сложности вычислений с акцентом на взаимосвязь между на первый взгляд различными задачами и методами. Приведены иллюстративные примеры, имеющие практическое и теоретическое значение. Обсуждаются направления новых исследований.

1. ВВЕДЕНИЕ

Теория сложности вычислений относится к количественным аспектам решений вычислительных задач. Обычно имеется несколько возможных алгоритмов решения таких задач, как вычисление значений алгебраических выражений, сортировка файла или синтаксический анализ цепочки символов. С каждым из этих алгоритмов связаны некоторые важные функции стоимости, такие как число шагов вычислений (как функция размера задачи), требуемый объем памяти для вычислений, размер программы и, в случае аппаратной реализации алгоритмов, — размер схемы и ее глубина.

По отношению к данной вычислительной задаче P можно ставить следующие вопросы. Какие хорошие алгоритмы существуют для решения задачи P? Можно ли установить и доказать нижнюю оценку для какой-нибудь функции стоимости, связанной с алгоритмом? Является ли задача практически неразрешимой в том смысле, что не существует алгоритма, решающего ее в практически обозримое время? Эти вопросы можно ставить как для поведения в наихудшем случае, так и для проведения в среднем алгоритмов решения задачи P. В последнее время было предложено расширение класса рассматривае-

мых алгоритмов, включающее рандомизацию в процессе вычислений. Некоторые из упомянутых выше вопросов можно обобщить и на вероятностные алгоритмы.

Вопросы сложности интенсивно изучались в течение последних двух десятилетий как в рамках общей теории, так и для конкретных задач, важных с точки зрения математики и практики. Из многочисленных достижений упомянем следующие:

— быстрое преобразование Фурье, в последнее время сильно продвинутое, с многообразными приложениями, включая приложения к передаче информации;

— установление того, что некоторые из задач автоматического доказательства теорем, возникающих в доказательстве корректности программ, являются практически невыполнимыми;

— получение точной оценки сложности схемы сложения n-разрядных двоичных чисел;

— удивительно быстрые алгоритмы для задач комбинаторики и теории графов и их связь с синтаксическим анализом;

— серьезные сокращения времени вычислений для некоторых важных задач, полученные посредством вероятностных алгоритмов.

Несомненно, работы во всех упомянутых выше направлениях будут продолжены. Кроме того, мы предвидим ответвление от теории сложности важных новых областей. Одна из них — задача обеспечения надежно защищенной связи, нуждающаяся в новой, усиленной теории сложности, которая стала бы прочным ее фундаментом. Другая — исследование весовых функций, относящихся к структурам данных. Огромные размеры предполагаемых баз данных призывают к углублению понимания сложности, внутренне присущей таким процессам, как построение списков и поиск в них. Теория сложности формирует подходы и дает необходимые инструменты для таких исследований.

Настоящая статья, которая является расширенным вариантом Тьюринговской лекции 1976 г., имеет целью предложить читателю взгляд с высоты птичьего полета на эту жизненно важную область. Не пытаясь дать исчерпывающий обзор, мы сосредоточим внимание на основных моментах и вопросах методологии.

2. ТИПИЧНЫЕ ЗАДАЧИ

Мы начнем с перечисления некоторых характерных вычислительных задач, которые важны теоретически (а часто и практически) и которые были предметом интенсивного изучения и анализа. В следующих разделах мы опишем методы, приме-

нявшиеся к этим задачам, и некоторые из полученных важных результатов.

2.1. ЦЕЛОЧИСЛЕННЫЕ ВЫЧИСЛИМЫЕ ФУНКЦИИ НА МНОЖЕСТВЕ ЦЕЛЫХ ЧИСЕЛ

Рассмотрим функции одной или более переменных из множества $N=\{0, 1, 2, \ldots\}$ целых чисел в N. Интуитивно мы признаем, что такие функции, как $f(x)=x!$, $g(x, y)=x^2+y^x$, являются вычислимыми.

А. М. Тьюринг, именем которого столь уместно названы эти лекции, поставил перед собой задачу строго определить, какие именно функции f: $N\to N$, g: $N\times N\to N$ и так далее эффективно вычислимы. Его модель идеализированной вычислительной машины и класс рекурсивных функций, вычислимых этой машиной, слишком хорошо известны, чтобы здесь на них останавливаться.

Здесь нас интересует, как измерить объем вычислительной работы, требуемой для нахождения значения $f(n)$ вычислимой функции f: $N\longrightarrow N$. Кроме того, возможно ли указать функции, которые сложны для вычисления любой программой? Мы вернемся к этому в п. 4.1.

2.2. АЛГЕБРАИЧЕСКИЕ ВЫРАЖЕНИЯ И УРАВНЕНИЯ

Пусть $E(x_1, \ldots, x_n)$ — алгебраическое выражение, построенное из переменных $x_1, \ldots, x_n$ посредством применения алгебраических операций $+$, $-$, $*$, $/$. Например, $E=(x_1+x_2)*(x_3+x_4)$. Нам надо вычислить $E(x_1, \ldots, x_n)$ при $x_1=c_1, \ldots, x_n=c_n$. В более общем случае задача может состоять в вычислении k выражений $E_1(x_1, \ldots, x_n), \ldots, E_k(x_1, \ldots, x_n)$ при $x_1=c_1, \ldots, x_n=c_n$.

Представляют интерес следующие случаи. *Вычисление полиномов*:

$$f(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_0. \qquad (1)$$

Умножение матриц AB, где A и B суть матрицы размера $n\times n$. Здесь требуется найти значения n^2 выражений $a_{i1}b_{1j}+\ldots+a_{in}b_{nj}$, $1\leqslant i, j\leqslant n$, для заданных числовых значений a_{ij}, b_{ij}.

Наш пример решения уравнений есть система

$$a_{i1}x_1+\ldots+a_{in}x_n=b_i, \quad 1\leqslant i\leqslant n, \qquad (2)$$

n линейных уравнений с n неизвестными $x_1, \ldots, x_n$. Нам нужно найти (вычислить) неизвестные при заданных коэффициентах a_{ij}, b_{ij}, $1\leqslant i, j\leqslant n$.

Мы не будем здесь обсуждать интересный вопрос о приближенных решениях алгебраических и трансцендентных уравнений, к которому также применимы методы теории сложности.

2.3. КОМПЬЮТЕРНАЯ АРИФМЕТИКА

Сложение. Даны два n-разрядных числа $a=\alpha_{n-1}\alpha_{n-2}\ldots\alpha_0$, $b=\beta_{n-1}\beta_{n-2}\ldots\beta_0$ (например, $n=4, a=1011, b=1100$); требуется найти $n+1$ разрядов суммы $a+b=\gamma_n\gamma_{n-1}\ldots\gamma_0$.

Умножение. Для тех же a, b найти $2n$ разрядов произведения $a*b=\delta_{2n}\delta_{2n-1}\ldots\delta_0$.

Решать такие арифметические задачи можно *аппаратно.* В этом случае основание системы счисления есть 2 и $\alpha_i, \beta_i=0, 1$. Для фиксированного n мы хотим построить схему с $2n$ входами и (для сложения) с $n+1$ выходами. Когда $2n$ разрядов (бит) чисел a и b подаются на входы, на $n+1$ выходах получаются $\gamma_n, \gamma_{n-1}, \ldots, \gamma_0$. Аналогичная ситуация имеет место для умножения.

С другой стороны, мы можем рассуждать о реализации арифметики посредством алгоритма, т. е. *программным образом.* Необходимость в этом может возникать по разнообразным причинам. Например, в арифметическом устройстве может быть предусмотрено только сложение, умножение тогда должно быть реализовано подпрограммой.

Программная реализация арифметики возникает также в контексте *арифметики многократной точности.* Наш компьютер имеет длину слова k, и надо сложить и умножить числа длины nk (n-словные числа). Мы выбираем в качестве основания число 2^k, так что $0\leqslant\alpha_i, \beta_i<2^k$, и используем алгоритмы для нахождения $a+b$, $a*b$.

2.4. СИНТАКСИЧЕСКИЙ АНАЛИЗ ВЫРАЖЕНИЙ В КОНТЕКСТНО-СВОБОДНЫХ ЯЗЫКАХ

Область теории сложности ни в коей мере не ограничивается алгебраическими или арифметическими вычислениями. Рассмотрим *контекстно-свободные грамматики*, из которых в качестве примера возьмем следующую. *Алфавит* грамматики G состоит из символов $t, x, y, z, (,), +, *$. Среди этих символов t *нетерминальный*, а все остальные — *терминальные*. *Продукции* (или правила преобразований) G суть следующие:

$$1.\ t\to(t+t),\quad 2.\ t\to t*t,\quad 3.\ t\to x,$$
$$4.\ t\to y,\quad 5.\ t\to z.$$

Начиная с t, мы может последовательно преобразовывать слова, пользуясь этими продукциями. Например,

$$\bar{t}\to^1(\bar{t}+t)\to^3(x+\bar{t})\to^2(x+\bar{t}*t)\to^4(x+y*\bar{t})\to^5(x+y*z). \qquad (3)$$

Число над стрелкой обозначает номер используемой продукции, а символом $\bar{t}$ обозначен тот нетерминальный символ, который должен быть переписан. Последовательность вида (3) называется *выводом*, и мы говорим, что $(x+y*z)$ выводимо из t. Множество всех слов u, выводимых из t и содержащих только терминальные символы, называется *языком*, *порожденным* G, и обозначается $L(G)$. Приведенная выше G — только пример, и обобщение на произвольные контекстно-свободные грамматики очевидно.

Контекстно-свободные грамматики и языки обычно появляются в связи с языками программирования и, конечно, при анализе естественных языков. Непосредственно возникают две вычислительные задачи. По данной грамматике G и слову W (т. е. по цепочке символов) в алфавите G установить, верно ли, что $W \in L(G)$. Это проблема *вхождения*.

Проблема синтаксического анализа состоит в следующем. Для данного слова W из $L(G)$ найти вывод (как последовательность продукций из G, подобную (3)) слова W из инициального символа G. Иначе говоря, мы хотим построить *дерево синтаксического анализа*. Нахождение дерева синтаксического анализа для алгебраического выражения, например, является существенным шагом в процессе компиляции.

2.5. СОРТИРОВКА ФАЙЛОВ

Файл из записей $R_1, R_2, \ldots, R_n$ хранится во внешней или основной памяти. Индекс i записи R_i указывает расположение этой записи в памяти. Каждая запись R имеет ключ (например, номер страховки в файле подоходных налогов) $k(R)$. Вычислительная задача состоит в переупорядочивании файла в памяти в последовательность $R_{i_1}, R_{i_2}, \ldots, R_{i_n}$ таким образом, чтобы ключи располагались в возрастающем порядке:

$$k(R_{i_1}) < k(R_{i_2}) < \ldots < k(R_{i_n}).$$

Мы подчеркиваем как различие между ключом и записью, которая может быть существенно длиннее ключа, так и требование действительного переупорядочивания записей. Эти особенности делают проблему более реалистичной и более сложной, чем просто сортировка чисел.

2.6. МАШИННОЕ ДОКАЗАТЕЛЬСТВО ТЕОРЕМ

С возникновением компьютеров появилось вполне понятное желание наделить их способностью рассуждать, что привело к приложению значительных усилий в этом направлении. В ча-

стности, были предприняты попытки сделать компьютер способным к логическим выводам и математическим рассуждениям и осуществить это посредством доказательства теорем чистой логики или посредством вывода теорем в математических теориях. Мы рассмотрим важный пример теории сложения натуральных чисел.

Рассмотрим систему $\mathcal{N}=\langle N, +\rangle$, состоящую из натуральных чисел $N=\{0, 1, \ldots\}$ и операции сложения $+$. Формальный язык L, употребляемый для обсуждения свойств $\mathcal{N}$, — это так называемый язык исчисления предикатов первого порядка. Он имеет переменные $x, y, z, \ldots$, значениями которых являются натуральные числа, символ операции $+$, равенство $=$, обычные пропозициональные связки и кванторы $\forall$ («для всех» и $\exists$ («существует»).

Такое предложение, как $\exists x \forall y (x+y=y)$, есть формальная запись высказывания: «Существует число x, такое, что для всех чисел y $x+y=y$». Это предложение фактически истинно в $\mathcal{N}$.

Множество всех предложений языка L, истинных в $\mathcal{N}$, будем называть *теорией* $\mathcal{N}$ ($Th(\mathcal{N})$) и обозначать $PA=Th(\mathcal{N})$. Например,

$$\forall x \forall y \exists z [x+z=y \vee y+z=x] \in PA .$$

Мы будем также пользоваться названием «арифметика Пресбургера», введенным в честь Пресбургера, получившего важные результаты для $Th(\mathcal{N})$.

Проблема разрешения для PA состоит в нахождении алгоритма, если в самом деле такой алгоритм существует, определяющего для каждого заданного предложения F языка L, верно $F \in PA$ или нет.

Такой алгоритм для PA построил Пресбургер [12]. Со времени появления этой работы многие исследователи пытались построить эффективные алгоритмы для этой задачи и реализовать их программно. Эти усилия предпринимались часто в рамках конкретных проектов в области автоматизации программирования и вертификации программ. Это делалось потому, что свойства программ, которые пытаются установить, иногда сводимы к утверждениям о сложении натуральных чисел.

3. ЦЕНТРАЛЬНЫЕ ВОПРОСЫ И МЕТОДОЛОГИЯ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ

В предыдущем разделе мы перечислили некоторые типичные вычислительные задачи. Позднее мы представим результаты, которые были получены для них. Здесь мы опишем в общем виде главные вопросы, которые там возникают, и центральные понятия теории сложности.

3.1. ОСНОВНЫЕ ПОНЯТИЯ

Класс однородных вычислительных задач мы будем называть *проблемой*. Индивидуальные случаи проблемы P мы будем называть *частными случаями* проблемы P. Таким образом, P есть множество всех своих частных случаев. Такое описание проблемы есть только предмет соглашения и удобства обозначений. Мы можем, например, говорить о проблеме умножения матриц. Частные случаи этой проблемы суть (для любого целого n) пары матриц, которые нужно перемножить.

С каждым частным случаем $I \in P$ проблемы P мы связываем размер $|I|$ (обычно целое число). Эта функция $|I|$ не единственна, и ее выбор диктуется теоретическими и практическими соображениями, связанными с тем, с какой точки зрения интересна эта проблема.

Возвращаясь к примеру умножения матриц, отметим, что разумной мерой для пары $I=(A, B)$ $(n \times n)$-матриц, которые надо перемножить, является $|I|=n$. Если мы изучаем объем памяти, требующейся для алгоритма умножения матриц, то подходящей может быть мера $|I|=n^2$. Наоборот, не кажется правдоподобным, что функция размера $|I|=n^n$ может естественным образом возникнуть в каком-нибудь контексте.

Пусть P — проблема и AL — алгоритм, решающий ее. При решении частного случая $I \in P$ алгоритм AL выполняет некоторую последовательность вычислений S_I. С S_I мы связываем некоторые числовые характеристики. Существенными являются, например, следующие характеристики: (1) длина S_I, которая характеризует время вычисления; (2) глубина S_I, т. е. число уровней параллельных шагов, на которые S_I может быть разложена; она соответствует времени, которое S_I потребовалось бы при параллельных вычислениях; (3) объем памяти, требуемый для вычисления S_I; (4) вместо общего числа шагов в S_I мы можем подсчитывать число шагов некоторого вида, таких, как арифметические операции при алгебраических вычислениях, число сравнений при сортировке или число обращений к памяти.

Для аппаратной реализации алгоритмов мы обычно определяем размер $|I|$, так, чтобы все частные случаи I одинакового размера n решались при помощи одной и той же схемы C_n. Сложность схемы C определяется разными способами, например, как число элементов, глубина, снова связанная с временем вычислений, или выбираются другие меры сложности, такие как число модулей, связанное с технологией, используемой при построении схемы.

После того как выбрана мера μ вычисления S, функция F_{AL} сложности вычисления может быть определена несколь-

ми способами, два главных из них — сложность в *наихудшем случае* и сложность *поведения в среднем*. Первое понятие определяется следующим образом:

$$F_{AL}(n) = \max\{\mu(S_I) \mid I \in P, \mid I \mid = n\}. \quad (4)$$

Для того чтобы определить поведение в среднем, мы задаем распределение вероятностей p на каждом множестве $P_n = \{I \mid I \in P, |I| = n\}$. Так, для $I \in P$, $|I| = n$, величины $p(I)$ — вероятность появления I среди всех других частных случаев размера n. *Поведение в среднем* алгоритма AL тогда определяется так:

$$M_{AL}(n) = \sum_{I \in P_n} p(I)\,\mu(S_I). \quad (5)$$

В п. 4.7 мы рассмотрим применимость предположения о распределении вероятностей.

Анализ алгоритмов связан со следующим вопросом. Для заданной функции размера $|I|$ и меры вычисления $\mu(S_I)$ точно определить для данного алгоритма AL, решающего проблему P, либо сложность F_{AL} для наихудшего случая, либо, при подходящих предположениях, поведение в среднем M_{AL}. В настоящей статье мы не будем рассматривать вопросы анализа, а предположим, что функция сложности известна или по крайней мере достаточно хорошо определена для наших целей.

3.2. ВОПРОСЫ

Теперь мы располагаем всеми необходимыми понятиями для постановки основного вопроса теории сложности: насколько успешно или с какой стоимостью может быть решена заданная вычислительная проблема P? Мы не имеем в виду никакого конкретного алгоритма решения P. Наша цель — рассмотреть *все* возможные алгоритмы решения P и попытаться сформулировать утверждение о вычислительной сложности, внутренне присущей P. Не надо забывать, что предварительный шаг в изучении сложности проблемы P — выбор меры $\mu(S)$, которая должна быть использована. Иначе говоря, мы должны решить, исходя из математических или практических соображений, *какую именно* сложность мы хотим исследовать. Сделав такой выбор, мы идем дальше.

В общих чертах (более детальные примеры и иллюстрации пока отложим) опишем основные интересующие нас вопросы. Кроме последнего пункта, они окажутся сгруппированными в пары.

(1) Найти эффективные алгоритмы для проблемы P.

(2) Установить нижние оценки сложности, внутренне присущей P.
(3) Поискать точные решения P.
(4) Найти алгоритмы для приближенных решений.
(5) Изучить сложность для наихудшего случая.
(6) Изучить сложность P в среднем.
(7) Найти последовательные алгоритмы для P.
(8) Найти параллельные алгоритмы для P.
(9) Изучить программно реализованные алгоритмы.
(10) Изучить аппаратно реализованные алгоритмы.
(11) Рассмотреть возможность решения посредством вероятностных алгоритмов.

Под (1) мы понимаем поиск хороших с точки зрения практики алгоритмов для данной проблемы. Дело в том, что непосредственно очевидные алгоритмы часто можно заменить алгоритмами гораздо лучшими. Улучшения в 100 раз вполне возможны. Но даже экономия вдвое может иногда означать различие между осуществимостью и неосуществимостью.

В то время как всякий алгоритм AL для P дает *верхнюю оценку* величины F_{AL} сложности P, нас интересует *нижняя оценка*. Типичный результат состоит в том, что всякий алгоритм AL, решающий P, удовлетворяет неравенству $g(n) \leqslant F_{AL}(n)$ по крайней мере для $n_0 < n$, где $n_0 = n_0\ (AL)$. В некоторых удачных случаях верхняя и нижняя оценки совпадают. Сложность такой проблемы в этом случае известна полностью. В любой ситуации знание нижней оценки представляет интерес математически и, кроме того, руководит нами в поиске хороших алгоритмов, указывая, какие попытки заведомо будут безуспешны.

Идея (4) приближенного решения проблемы существенна по той причине, что иногда практически удовлетворительное приближенное решение гораздо проще вычислить, чем точное решение.

Основные вопросы (1) и (2) можно изучать в комбинации с одной или несколькими вопросами (3)—(11). Так, например, мы можем изучать верхнюю оценку среднего времени, требуемого для сортировки n процессорами, работающими параллельно. Или мы можем изучать число логических элементов, необходимых для сортировки n входных двоичных разрядов.

Может показаться, что с многообразными возможностями выбора меры сложности и разнообразными вопросами, которые могут возникать, теория сложности вычислений станет коллекцией изолированных результатов и не связанных друг с другом методов. Положение, которое мы пытаемся подчеркнуть приводимыми ниже примерами, — это большая степень взаимосвязи

разных вопросов внутри этой области и общность идей и методов, здесь преобладающих.

Мы увидим, что эффективные алгоритмы для параллельных вычислений полиномов переносятся на схемы для быстрого сложения n-разрядных двоичных чисел. Идея быстрого преобразования Фурье приводит к хорошим алгоритмам для умножения чисел с многократной точностью. На более высоком уровне отношение между программными и аппаратными алгоритмами вполне аналогично отношению между последовательными и параллельными вычислениями. Современные программы построены в расчете на единственный процессор и потому являются последовательными, тогда как аппаратная реализация содержит много идентичных подсхем, которые можно рассматривать как примитивные процессоры, работающие параллельно. В наших примерах снова и снова возникает метод предварительной обработки, еще раз показывая общность всех возникающих в данной области вопросов.

4. РЕЗУЛЬТАТЫ

4.1. СЛОЖНОСТЬ ОБЩЕРЕКУРСИВНЫХ ФУНКЦИЙ

В [13, 14] автором начато изучение классификации вычислимых целочисленных функций по сложности их вычисления. Подход был принят аксиоматический, так что понятия и результаты применимы ко всякому осмысленному классу алгоритмов и всякой мере сложности, определенной на множестве вычислений.

Пусть K — класс алгоритмов, возможно базирующийся на некоторой модели математических машин, так что для каждой вычислимой функции f: $N \longrightarrow N$ существует алгоритм $AL \in K$, вычисляющий ее. Мы не уточняем меру $\mu(S)$ на вычислениях S, а предполагаем, что μ удовлетворяет некоторым естественным аксиомам. Эти аксиомы выполняются на всех конкретных примерах мер в п. 3.1. В качестве размера целого числа n берется $|n| = n$. Вычисление функции f есть проблема, где для каждого частного случая n мы должны найти $f(n)$. По (4) из п. 3.1 мы имеем для каждого алгоритма AL для f функцию $F_{AL}(n)$ сложности вычисления, измеряющую работу, требуемую для вычисления f по алгоритму AL.

Теорема [13, 14]. *Для всякой вычислимой функции* g: $M \longrightarrow N$ *существует вычислимая функция* f: $N \longrightarrow \{0, 1\}$, *такая, что для всякого алгоритма* $AL \in K$, *вычисляющего* f, *существует число* n_0, *такое, что*

$$g(n) < F_{AL}(n) \text{ для } n_0 < n. \qquad (6)$$

Мы требуем, чтобы f была функцией со значениями 0 и 1, потому что в противном случае мы могли бы построить сложную функцию, просто позволив $f(n)$ расти очень быстро, так что записывать результат станет слишком сложно.

Ограничение $n_0 < n$ в (6) необходимо. Для каждого f и k мы можем построить алгоритм, содержащий внутри себя таблицу значений $f(n)$, $n \leqslant k$, делающий вычисления тривиальными для $n \leqslant k$.

Основной смысл приведенной теоремы заключается в том, что (6) при подходящем $n_0 = n_0$ (AL) имеет место для *любого* алгоритма вычисления f. Таким образом, внутренне присущая сложность вычисления f больше, чем g.

Начиная с [14], Блюм [1] ввел другие, но по существу эквивалентные аксиомы для функции сложности. Блюм получил много интересных результатов, включая теорему об *ускорении*. Эта теорема показывает существование вычислимых функций, для которых не существует наилучшего алгоритма. Точнее, для всякого алгоритма вычисления такой функции найдется другой алгоритм, вычисляющий ее гораздо быстрее.

Исследования в этой области абстрактной теории сложности значительно продвинулись в последнее десятилетие. Это послужило основой для теории сложности вычислений по двум причинам: во-первых, из-за постановки самого вопроса о стоимости вычислений и, во-вторых, подчеркиванием необходимости решать и сравнивать все возможные алгоритмы решения данной проблемы.

С другой стороны, абстрактная теория сложности не ведет к пониманию конкретных вычислительных задач и их оценке с точки зрения практики. Это показывают следующие примеры.

4.2. АЛГЕБРАИЧЕСКИЕ ВЫЧИСЛЕНИЯ

Начнем с примера вычисления полиномов. Выберем в качестве меры число арифметических операций и обозначим через (nA, kM) стоимость n сложений/вычитаний и k умножений/делений. Переписывая полином (1) в виде

$$f(x) = (\ldots((a_n x + a_{n-1})x + a_{n-2}))\,x + \ldots + a_0,$$

мы убеждаемся, что общий полином n-й степени можно вычислить за (nA, kM). В духе вопросов (1) и (2) п. 3.2 мы спрашиваем, может ли искусный алгоритм использовать меньшее число операций. Довольно тонкие математические рассуждения показывают, что приведенное выше число операций оптимально, так что вопрос полностью решен.

Т. Моцкин в [9] ввел в рассмотрение важную идею *предварительной обработки* для вычислений. Во многих важных прило-

жениях требуется вычислить один и тот же полином при многих различных значениях аргументов $x=c_1$, $x=c_2$, ... Он предложил следующую стратегию предварительной обработки коэффициентов полинома (1). Вычислить раз и навсегда все числа $\alpha_0(a_0, \ldots, a_n), \ldots, \alpha_n(a_0, \ldots, a_n)$ по данным коэффициентам $a_0, \ldots, a_n$. При вычислении $f(c)$ использовать $\alpha_0, \ldots, \alpha_n$. Такой подход имеет смысл в том случае, когда стоимость предварительной обработки мала по сравнению с общей экономией при вычислении $f(c_1), f(c_2), \ldots$, т. е. когда ожидаемое число значений аргументов, для которых $f(x)$ должно быть вычислено, очень велико. Моцкин получил следующий результат.

Теорема. *С использованием предварительной обработки полином степени n можно вычислить за* $(nA, ([n/2]+2)M)$.

Снова можно доказать, что этот результат по существу наилучший из возможных. Что можно сказать о параллельных вычислениях? Если мы используем k процессоров и должны вычислить выражение, требующее по крайней мере m операций, то самое лучшее, на что можно надеяться, — это время вычислений $(m/k)-1+\log_2 k+O(1)$.

А именно, предположим, что все процессоры непрерывно заняты, тогда $m-k$ операций выполняются за время $(m/k)-1$. Оставшиеся k операций должны объединяться посредством бинарных операций над k входами в один выход, и это требует времени по крайней мере $\log_2 k$. Ввиду вышесказанного следующий результат, принадлежащий Манро и Патерсону [10], близок к наилучшему.

Теорема. *Полином (1) может быть вычислен k процессорами, работающими параллельно, за время* $(2n/k)+\log_2 k+O(1)$.

Прогресс в развитии аппаратных средств позволяет надеяться, что мы сможем употреблять большое число процессоров для одной задачи. Брент [3] наряду с другими изучал следствия неограниченного параллелизма и доказал следующее.

Теорема. *Пусть $E(x_1, \ldots, x_n)$ — арифметическое выражение, где каждая переменная появляется лишь однажды. Выражение E может быть вычислено при условии неограниченного параллелизма за время* $4\log_2 n$.

Другой важный объект — быстрое преобразование Фурье (БПФ). Операция свертки, которая имеет много приложений, таких, как преобразование сигналов, — пример вычисления, сильно упрощаемого применением БПФ. Пусть $a_1, \ldots, a_n$ — последовательность n чисел, $b_1, b_2, \ldots$ — входной поток чисел. Определим для $i=1, 2, \ldots$

$$c_i = a_1 b_i + a_2 b_{i+1} + \ldots + a_n b_{i+n-1}. \qquad (7)$$

Нам нужно вычислить значения c_1, c_2, ... Из (7) может показаться, что стоимость вычисления одного c_i равна $2n$ операций. Если мы вычисляем величины c_i блоками размера n, т. е. $c_1, \ldots, c_n$, затем $c_{n+1}, \ldots, c_{2n}$ и т. д., используя БПФ, то стоимость вычисления каждого блока примерно равна $5n \log_2 n$, так что стоимость одного c_i равна $5 \log_2 n$.

Используя хитроумную комбинацию алгебраических и теоретико-числовых идей, Виноград [20] недавно улучшил время вычисления свертки для малых n и дискретного преобразования Фурье для малых и средних n. Для $n \sim 1000$, например, его метод примерно вдвое быстрее традиционного алгоритма БПФ.

Очевидные методы для умножения $(n \times n)$-матриц и для решения системы. (2) n линейных уравнений с n неизвестными требует примерно n^3 операций. Штрассен [17] получил следующий удивительный результат.

Теорема. *Две $(n \times n)$-матрицы могут быть перемножены с использованием самое большое $4{,}7n^{2{,}81}$ операций. Система n линейных уравнений с n неизвестными может быть решена посредством $4{,}8n^{2{,}81}$ операций.*

Маловероятно, что показатель $\log_2 7 \sim 2{,}81$ на самом деле наилучший из возможных, но ко времени написания данной статьи все попытки улучшить этот результат не дали успеха.

4.3. НАСКОЛЬКО БЫСТРО МЫ МОЖЕМ СКЛАДЫВАТЬ И УМНОЖАТЬ?

Этот очевидно важный вопрос подвергся тщательному анализу. Простое рассуждение показывает, что если использовать элементы с r входами, то схема сложения n-разрядных двоичных чисел требует временной сложности по крайней мере $\log_r n$. Эта нижняя оценка фактически достижима.

Следует отметить, что в ходе замечаний п. 3.2 об аналогии между параллельными алгоритмами и аппаратной реализацией алгоритмов один из наилучших результатов по схемам сложения (Брент [2]) использует булевы тождества, которые немедленно переводимы в эффективный алгоритм параллельного вычисления полиномов.

Приведенные выше результаты относятся к двоичному представлению чисел, которые нужно сложить. Может ли быть, что при достаточно разумном кодировании чисел $0 \leqslant a < 2^n$ сложение по модулю 2^n выполнимо быстрее, чем за $\log_r n$? На этот вопрос ответил Виноград [19]. При весьма общих предположениях относительно кодирования нижняя оценка остается $\log_r n$.

Если обратиться к арифметике многократной точности, то интересные вопросы возникают в связи с умножением. Очевид-

ный метод перемножения чисел длины n содержит n^2 двоичных операций. Ранние попытки улучшения использовали простые алгебраические тождества и привели к упрощению до $O(n^{1,58})$ операций.

Шенхаге и Штрассен [16] использовали связь между умножением натуральных чисел и умножением многочленов и применили быстрое преобразование Фурье (БПФ) для получения следующей теоремы.

Теорема. *Два n-разрядных двоичных числа могут быть перемножены за $O(n \log n \log\log n)$ операций.*

Получение нижних оценок сложности умножения целых чисел должно относиться к конкретной вычислительной модели. При очень разумных предположениях Патерсон, Фишер и Мейер [11] серьезно уменьшили разрыв между верхней и нижней оценками, показав следующее.

Теорема. *Для умножения n-разрядных двоичных чисел необходимо по крайней мере $O(n \log n/\log\log n)$ операций.*

4.4. СКОРОСТЬ СИНТАКСИЧЕСКОГО АНАЛИЗА

Синтаксический анализ выражений в контекстно-свободных грамматиках на первый взгляд кажется требующим дорогостоящих возвратных вычислений. Динамическое вычисление, которое одновременно отслеживает предков всех подцепочек анализируемой цепочки, приводит к алгоритму, требующему $O(n^3)$ шагов для синтаксического анализа слова длины n. Коэффициент при n^3 зависит от грамматики. Долгое время этот результат оставался наилучшим, хотя для специальных классов контекстно-свободных грамматик были получены и лучшие оценки.

Фишер и Мейер заметили, что на основе штрассеновского алгоритма матричного умножения можно получить алгоритм поразрядного умножения двух булевых $(n \times n)$-матриц, требующий $O(n^{2,81}c(n))$ поразрядных булевых операций. Здесь $c(n) = \log n \log\log n \log\log\log n$ и, таким образом, имеет вид $O(n^a)$ для всякого $0 < a$.

Валиант [18] обнаружил, что синтаксический анализ столь же сложен, как и умножение булевых матриц. Следовательно, поскольку $\log_2 7 < 2{,}81$, справедлива следующая теорема.

Теорема. *Выражения длины n в контекстно-свободном языке $L(G)$ могут быть разобраны за время $d(G)n^{2,81}$.*

Мы снова видим, как результаты из алгебраической теории сложности приносят плоды в области комбинаторных вычислений.

4.5. ОБРАБОТКА ДАННЫХ

Из области приложений теории сложности к обработке данных мы рассмотрим известнейший пример, а именно сортировку. Мы следуем формулировкам, данным в п. 2.5.

Хорошо известно, что сортировка n чисел в памяти с произвольным доступом требует примерно $n \log n$ сравнений. Это и худший случай поведения некоторых алгоритмов, и поведение в среднем других алгоритмов в предположении, что все перестановки равновероятны.

Переупорядочивание записей $R_1, R_2, \ldots, R_n$ ставит дополнительные проблемы, потому что файл обычно расположен в последовательной или близкой к последовательной памяти, такой, как магнитная лента или диск. Ограниченность оперативной памяти позволяет переносить в нее для переупорядочивания лишь небольшое число записей за один раз. Все же можно разработать алгоритмы физического переупорядочивания файлов за время $cn \log n$, где c зависит от характеристик рассматриваемой системы.

Один поучительный результат в этой области принадлежит Флойду [6]. В его модели файл располагается на нескольких страницах $P_1, \ldots, P_m$ и каждая страница содержит k записей, так что P_i содержит записи $R_{i1}, \ldots, R_{ik}$. Для наших целей мы можем предположить без ограничения общности, что $m=k$. Задача состоит в перераспределении записей таким образом, что R_{ij} окажется на странице P_j для всех $1 \leqslant i, j \leqslant k$. Объем быстрой памяти позволяет прочитать две страницы $P_{i'}, P_{j'}$, переупорядочить их записи и вывести обе страницы на внешний носитель. Используя рекурсию, аналогичную применяемой в БПФ, Флойд доказал следующее.

Теорема *Переупорядочивание записей описанным выше способом может быть выполнено за $k \log_2 k$ пересылок в быструю память. Этот результат наилучший из возможных.*

Нижняя оценка устанавливается посредством рассмотрения подходящей энтропийной функции. Она применима в предположении, что внутри быстрой памяти записи только перетасовываются. Неизвестно, можно ли построить алгоритм с меньшим числом пересылок страниц, если допустить вычисления с записями, рассматриваемыми как цепочки бит.

4.6. ПРАКТИЧЕСКИ НЕВЫПОЛНИМЫЕ ЗАДАЧИ

Машинное доказательство теорем порождает вычислительные задачи, которые требуют столь большого числа вычислительных шагов, что являются практически невыполнимыми

В попытках прогонять программы для решения проблемы разрешения пресбургеровской арифметики (*PA*) на ЭВМ вычисления заканчивались лишь в простейших случаях. Теоретическое обоснование этого эмпирического факта дает следующий результат Фишера и Рабина [5].

Теорема. *Существует константа $c>0$, такая, что для каждого разрешающего алгоритма AL для PA существует число n_0, такое, что для каждого $n>n_0$ существует предложение H языка L (языка для сложения чисел), удовлетворяющее следующим условиям:* (1) $l(H)=n$, (2) *AL требует более чем* $2^{2^{cn}}$ *шагов для определения того, верно ли* $H \in PA$, *т. е. верно ли, что H истинно в* $\langle N, + \rangle$. *Здесь* $l(H)$ *обозначает длину H.*

Константа c зависит от обозначений, используемых при формулировании свойств $\langle N, + \rangle$. Во всяком случае, она не слишком мала. Очень быстрый рост неустранимой нижней оценки $2^{2^{cn}}$ показывает, что, даже пытаясь решить проблему разрешения для этой очень простой и «элементарной» математической теории, мы сталкиваемся с практически невыполнимыми вычислениями. Мейер [8] привел примеры теорий с еще более ошеломляюще сложными проблемами разрешения.

Простейший уровень логического вывода — исчисление высказываний. Из пропозициональных переменных p_1, p_2, ... мы можем строить формулы, такие, как $[p_1 \wedge \sim p_1] \vee [p_2 \wedge \sim p_2]$, посредством пропозициональных связок. *Проблема выполнимости* состоит в выяснении для пропозициональной формулы $G(p_1, \ldots, p_n)$ существования таких истинностных значений переменных $p_1, \ldots, p_n$, что формула G истинна. Например, значения $p_1=F$ (ложь), $p_n=T$ (истина) удовлетворяют приведенной выше формуле.

Прямой алгоритм для проблемы выполнимости требует примерно 2^n шагов для формулы с n переменными. Неизвестно, существуют ли неэкспоненциальные алгоритмы для проблемы выполнимости.

Большая важность этого вопроса была осознана в результате исследований Кука [4]. Можно определить естественный процесс так называемого полиномиального сведения одной вычислительной задачи P к другой задаче Q. Если P полиномиально сводима к Q и Q разрешима за полиномиальное время, тогда то же справедливо относительно P. Две взаимно сводимые задачи называются полиномиально эквивалентными. Кук показал, что проблема выполнимости эквивалентна так называемой задаче клик в графах. Карп [7] указал большое число задач, эквивалентных проблеме выполнимости. Среди них такие задачи, как двоичное целочисленное программирование, существо-

вание гамильтонова цикла в графе, целочисленная задача о коммивояжере и многие другие.

В силу таких эквивалентностей если за полиномиальное время разрешима одна из этих задач, то все остальные тоже разрешимы. Вопрос о том, имеет ли выполнимость полиномиальную сложность, называется проблемой $P=NP$ и справедливо считается самой знаменитой задачей теории сложности вычислений.

4.7. ВЕРОЯТНОСТНЫЕ АЛГОРИТМЫ

Как упомянуто в п. 3.1, изучение поведения в среднем, т. е. ожидаемого времени работы алгоритма, основано на предположении о распределении вероятностей в пространстве частных случаев задачи. Это предположение наталкивается на некоторые методологические трудности. Мы можем постулировать некоторое распределение, например считать все частные случаи равновероятными, но на практике источник частных случаев решаемой задачи может вести себя совершенно иначе. Распределение может меняться во времени и часто будет нам неизвестным. В экстремальной ситуации большинство частных случаев, которые действительно встречаются, суть в точности те, для которых алгоритм работает хуже всего.

Можно ли воспользоваться вероятностными моделями вычислений каким-нибудь другим способом, при котором мы будем полностью контролировать ситуацию? *Вероятностный алгоритм* AL для задачи P использует датчик случайных чисел. Когда решается частный случай $I \in P$, генерируется короткая последовательность $r=(b_1, \ldots, b_k)$ случайных чисел, и они используются в AL для *точного* решения P. После фиксации случайного выбора r алгоритм выполняется вполне детерминированно.

Мы говорим, что такой AL решает P за ожидаемое время $f(n)$, если для всякого $I \in P$, $|I|=n$, AL решает I за ожидаемое время, меньшее или равное $f(n)$. Под ожидаемым временем мы понимаем среднее время решения I посредством AL для всех возможных способов выбора последовательности r (которые мы предполагаем равновероятными).

Отметим различие между этим понятием и хорошо известным методом Монте-Карло. В последнем методе мы строим для задачи стохастический процесс, который ее моделирует, и, «измеряя» его, получаем приближенные решения задачи. Поэтому метод Монте-Карло есть по существу аналоговый метод решения. Наши вероятностные алгоритмы, наоборот, используют случайные числа $b_1, \ldots, b_k$, чтобы определить ветвление в алгоритмах, в остальном детерминированных, и получить точное, а не приближенное решение.

Может показаться маловероятным, что такая консультация с «бросанием кости» может ускорить вычисления. Автор систематически изучал [15] вероятностные алгоритмы. Оказывается, что в некоторых случаях этот подход дает драматические улучшения.

Ближайшей парой в множестве точек $x_1, \ldots, x_n \in R^k$ (k-мерного пространства) называется пара x_i, x_j, $i \neq j$, для которой расстояние $d(x_i, x_j)$ минимально. Вероятностный алгоритм находит ближайшую пару за ожидаемое время $O(n)$, т. е. быстрее любого обычного алгоритма.

Задача определения того, является ли натуральное число n простым, становится нерешаемой для больших n. Известные методы не срабатывают при $n \sim 10^{60}$, если они применяются не к числам специального вида. Вероятностный алгоритм, построенный автором, работает со скоростью $O((\log n)^3)$. На компьютере средней мощности число $2^{400} - 593$ было распознано как простое за несколько минут. Этот метод работает столь же хорошо для многих других чисел сравнимого размера.

Полный потенциал этих идей еще не известен и достоин дальнейшего изучения.

5. НОВЫЕ НАПРАВЛЕНИЯ

Из возможных путей дальнейших исследований упомянем только два.

5.1. БОЛЬШИЕ СТРУКТУРЫ ДАННЫХ

Коммерческие нужды требуют создания еще больших баз данных. В то же время нынешние и, более того, возникающие технологии будущего позволяют создавать гигантскую память с разными степенями свободы доступа.

Многие текущие исследования по базам данных нацелены на языки взаимодействия между пользователем и системой. Но огромные размеры списков и других предполагаемых структур могут сделать требуемые операции над этими структурами очень дорогостоящими, если не будет достигнуто углубленное понимание алгоритмов выполнения этих операций.

Мы можем начать с проблемы нахождения теоретической, но в то же время практически значимой модели таких списков. Эта модель должна быть достаточно гибкой, чтобы ее можно было применять к различным типам использующихся сейчас структур списков.

Какие операции применяются к спискам? Мы можем перечислить некоторые: поиск в списке, сбор мусора, доступ к различным точкам списка, вставки, исключения, слияние списков.

Можно ли систематизировать изучение этих и других важных операций? Каковы разумные функции стоимости, связанные с этими операциями?

Наконец, глубокое количественное понимание структур данных могло бы быть основой рекомендаций для выбора архитектуры компьютеров. Дают ли параллельные вычисления заметное ускорение выполнения различных операций над структурами данных? Какими полезными свойствами можно наделить списки при использовании ассоциативной памяти? Это, разумеется, только примеры.

5.2. ЗАЩИЩЕННЫЕ ЛИНИИ СВЯЗИ

В защищенных линиях связи применяются некоторые схемы кодирования, и мы можем поставить основные вопросы теории сложности вычислений по отношению к таким системам. Проиллюстрируем это посредством системы блочного кодирования.

В блочном кодировании употребляют цифровые устройства, которые входные слова длины n кодируют при помощи ключа. Если x — слово длины n и z — ключ (предположим, что ключи тоже длины n), то пусть $E_z = y$, $l(y) = n$, обозначает результат кодирования x ключом z. Сообщение $w = x_1, x_2, \ldots, x_k$ длины kn кодируется как $E_z(x_1)E_z(x_2) \ldots E_z(x_k)$.

Если противник способен добыть текущий ключ z, он может декодировать связи между сторонами, так как мы предполагаем, что он владеет кодирующим и декодирующим оборудованием. Он может также вставить поддельные сообщения, которые будут правильно закодированы. В коммерческой связи это даже более опасно, чем нарушение секретности.

С точки зрения секретности следует также принимать в расчет возможность того, что противник владеет некоторым количеством сообщений $w_1, w_2, \ldots$ открытым текстом и в закодированном виде. Может ли ключ z быть вычислен по этим данным?

Недостаточно доказать, что такое вычисление невыполнимо. Дело в том, что результаты современной теории сложности дают нам информацию для наихудшего случая. Так, если, скажем, для большинства вычислений, связанных с поисками ключа, установлена нижняя оценка вычислительной сложности 2^n, то задача считается невыполнимой. Но если алгоритм обнаружит ключ в практически обозримое время в одном случае из тысячи, то возможности обмана будут недопустимо велики.

Таким образом, нам необходима такая теория сложности, которая позволяет нам установить и доказать, что некоторое вычисление неосуществимо практически всегда. Например, система блочного кодирования надежна, если любой алгоритм

определения ключа будет заканчиваться в практически обозримое время только в $O(2^{-n})$ случаев. Мы весьма далеки от создания такой теории, особенно на данном этапе, когда проблема $P=NP$ еще не решена.

ЛИТЕРАТУРА

1. Blum M. A machine independent theory of the complexity of recursive funotions. J. ACM 14 [1967], 322—336.
2. Brent R. P. On the addition of binary numbers. IEEE Trans. Comptrs. C-19 [1970], 758—759.
3. Brent R. P. The parallel evaluation of algebraic expressions in logarithmic time. Complexity of Sequential and Parallel Numerical Algorithms, J. F. Traub, Ed., Academic Press, New York, 1973, pp. 83—102.
4. Cook S. A. The complexity of theorem proving procedures. Proc. Third Annual ACM Symp. on Theory of Comptng. 1971, pp. 151—158.
5. Fisher M. J., and Rabin M. O. Super-exponential complexity of Presburger arithmetic. In Complexity of Computations (SIAM—AMS Proc., Vol. 7), R. M. Karp, Ed., 1974, pp. 27—41.
6. Floyd R. W. Permuting information in idealized two-level storage. In Complexity of Computer Computations. R. Miller and J. Thatcher, Eds., Plenum Press, New York, 1972, pp. 105—109.
7. Karp R. M. Reductibility among combinatorial problems. In Complexity of Computer Computations. R. Miller and J. Thatcher, Eds., Plenum Press, New York, 1972, pp. 85—103.
8. Meyer A. R. The inherent computational complexity of theories of order. Proc. Int. Cong. Math., Vol. 2, Vancouver, 1974, pp. 477—482.
9. Motzkin T. S. Evaluation of polynomials and evaluation of rational functions. Bull. Amer. Math. Soc. 61 (1955), 163.
10. Munro I., and Paterson, M. Optimal algorithms for parallel polynomial evaluation. J. Comptr. Syst. Sci. 7 (1973), 189—198.
11. Paterson M., Fisher, M. J., and Meyer, A. R. An improved overlap argument for on-line multiplication. Proj. MAC Tech. Report 40, M. I. T. (1974).
12. Presburger M. Ueber die Vollstaendigkeit eines gewissen Systems Arithmetic ganzer Zahlen in welchem die Addition als einzige Operation hervortritt. Comptes-rendues du I Congres de Mathematiciens de Pays Slaves, Warsaw, 1930, pp. 92—101, 395.
13. Rabin M. O. Speed of computation and classification of recursive sets. Third Convention Sci. Soc., Israel, 1959, pp. 1—2.
14. Rabin M. O. Degree of difficulty of computing a function and a partial ordering of recursive sets. Tech. Rep. No. 1, O. N. R., Jerusalem, 1960.
15. Rabin M. O. Probabilistic algorithms. In Algorithms and Complexity, New Directions and Recent Trends, J. F. Traub, Ed., Academic Press, New York, 1976, pp. 29—39.
16. Schonhage A., and Strassen, V. Schnelle Multiplication grosser Zahlen. Computing 7 (1971), 281—292.
17. Strassen V. Gaussian elimination is not optimal. Num. Math. 13 (1969), 354—356.
18. Valiant L. G. General context-free recognition in less than cubic time. Rep., Dept. Comptr. Sci., Carnegie-Mellon U., Pittsburgh., Pa., 1974.
19. Winograd S. On the time required to perform addition. J. ACM 12 (1965), 277—285.
20. Winograd S. On computing the discrete Fourier transform. Proc. Natl. Acad. Sci. USA 73 (1976), 1005—1006.