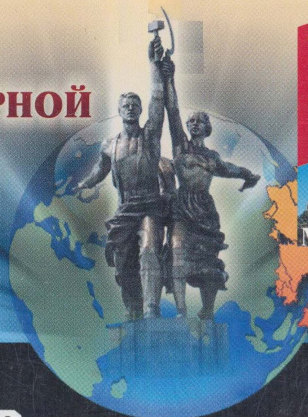


**ШЕДЕВРЫ
НАУЧНО-ПОПУЛЯРНОЙ
ЛИТЕРАТУРЫ**

МАТЕМАТИКА

**НАУКУ —
ВСЕМ!**

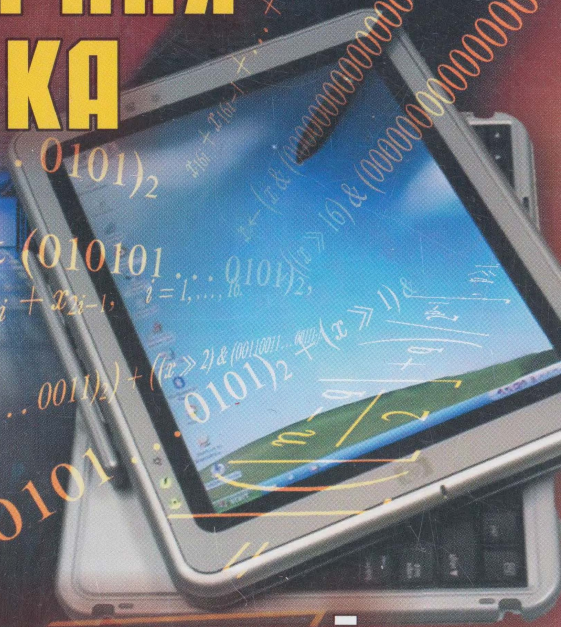


С. Б. Гашков

ЗАНИМАТЕЛЬНАЯ КОМПЬЮТЕРНАЯ АРИФМЕТИКА

$$x \leftarrow (x \gg 1) \& (010101 \dots 0101)_2$$
$$(y_{2i} y_{2i-1})_2 = x_{2i} + x_{2i-1}, \quad i = 1, \dots, 16$$

**Быстрые
алгоритмы операций
с числами
и многочленами**



С. Б. Гашков

ЗАНИМАТЕЛЬНАЯ КОМПЬЮТЕРНАЯ АРИФМЕТИКА

**Быстрые алгоритмы
операций с числами
и многочленами**



**URSS
МОСКВА**

Гашков Сергей Борисович

Занимательная компьютерная арифметика: Быстрые алгоритмы операций с числами и многочленами. — М.: Книжный дом «ЛИБРОКОМ», 2012. — 224 с. (НАУКУ — ВСЕМ! Шедевры научно-популярной литературы (математика).)

В настоящей книге рассматриваются методы быстрого выполнения различных видов вычислений, рассказывается о реализации быстрых алгоритмов как в виде логических схем — математической модели реальных электронных микросхем, так и в виде компьютерных программ. Исследуются также вопросы о том, как измерить сложность того или иного вычислительного алгоритма и оценить время его работы на компьютере. Большая часть материала книги доступна всем, кто знаком лишь со школьным курсом математики, но и опытный читатель может найти в этой книге кое-что новое для себя.

Книга написана на основе лекций, которые автор в разное время читал учащимся физико-математической Школы им. А. Н. Колмогорова при МГУ, на Малом и Большом мехмате, а также на факультетах информационной безопасности и информатики РГГУ.

Издательство «Книжный дом «ЛИБРОКОМ»».

117335, Москва, Нахимовский пр-т, 56.

Формат 60×90/16. Печ. л. 14. Зак. № ЖМ-54.

Отпечатано в ООО «ЛЕНАНД».

117312, Москва, пр-т Шестидесятилетия Октября, 11А, стр. 11.

ISBN 978-5-397-02880-6

© Книжный дом «ЛИБРОКОМ», 2012



10585 ID 124753



Все права защищены. Никакая часть настоящей книги не может быть воспроизведена или передана в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, а также размещение в Интернете, если на то нет письменного разрешения владельца.

Моей матери и памяти моего отца

Содержание

От автора	6
Введение	7
Глава 1. Школьные алгоритмы арифметических операций с многочленами	12
Глава 2. Школьные алгоритмы сложения и умножения чисел	19
Глава 3. Умножение столбиком нескольких чисел	25
Глава 4. Переносы при сложении двоичных чисел и теорема Куммера	30
Глава 5. Минимальные формы двоичной записи с цифрами 0 и ± 1 и первая попытка уменьшить сложность умножения	35
Глава 6. Быстрое умножение многочленов	43
Глава 7. Быстрое умножение чисел	50
<i>Схемная реализация метода Карацубы для умножения двоичных чисел</i>	<i>52</i>
Глава 8. Деление многозначных чисел	62
Глава 9. Как представляются отрицательные числа в компьютере	71
Глава 10. SRT-деление	75
Глава 11. Быстрое деление многочленов	94
Глава 12. Быстрое деление чисел	107

Глава 13. Сравнение сложности умножения, деления, возведения в квадрат и извлечения квадратного корня	120
Глава 14. О сложности преобразования чисел из одной системы в другую	131
Глава 15. Модулярная арифметика и китайская теорема об остатках	143
Глава 16. Сложность операций модулярной арифметики . . .	153
<i>Как найти остаток от деления, не вычисляя частное</i>	157
Глава 17. Умножение и деление на константы	169
Глава 18. Некоторые быстрые алгоритмы работы с битами . .	179
<i>Маленькие хитрости в работе с битами</i>	186
Глава 19. Вычисление некоторых целочисленных элементарных функций	193
<i>Целочисленный квадратный корень</i>	193
<i>Целочисленные логарифмы</i>	196
Глава 20. Быстрые операции с дробно-рациональными функциями	197
<i>Быстрое сложение дробно-рациональных функций</i>	197
<i>Быстрый китайский алгоритм</i>	202
<i>Быстрая интерполяция</i>	203
<i>Еще о быстром умножении многочленов</i>	204
Глава 21. Варианты алгоритма Евклида	206
<i>Алгоритм Евклида с выбором минимального остатка . .</i>	206
<i>Бинарный алгоритм Евклида</i>	210
Глава 22. Еще о схеме Горнера	214
Глава 23. Что можно вычислить на счетах	218
Литература	222

От автора

Предлагаемая вниманию читателя книга является продолжением моей книги «Занимательная компьютерная арифметика: Математика и искусство счета на компьютерах и без них». Название «Быстрые алгоритмы» допускает неоднозначное толкование. Например, естественно подумать, что будут обсуждаться методы быстрого выполнения различных видов вычислений (фактически, подобные вопросы уже рассматривались в предыдущей части книги). Но можно также решить, что будут рассматриваться вопросы о том, как измерить сложность того или иного вычислительного алгоритма и оценить время его работы на компьютере (или объем вычислений, который предстоит выполнить собственными руками). Речь пойдет и о том, и о другом.

Настоящий материал можно изучать независимо от первой части, хотя в нем и содержится небольшое количество ссылок на книгу «Математика и искусство счета на компьютерах и без них». Следующее далее введение в основном повторяет введение к предыдущей книге.

Введение

*Развитие быстрых алгоритмов
происходит медленно.*

Арнольд Шёнхаге

Здесь уместно разъяснить, что означает термин «компьютерная арифметика» или, по крайней мере, как он понимается в книге.

Как и любая область науки, компьютерная арифметика состоит из нескольких связанных между собой направлений и по-разному выглядит с разных точек зрения. В ней есть и многочисленные технические детали, и фундаментальные понятия. Естественно, вторым будет уделяться больше внимания, чем первым, поэтому в технической области изложение не всегда будет исчерпывающим.

С одной стороны, компьютерная арифметика, как и следует ожидать, должна объяснять, как выполняются на компьютере различные операции с числами и/или машинными словами. Компьютеры имеют многочисленные команды для выполнения таких операций, среди которых есть и арифметические, и логические. Разные компьютеры (а к их числу без большой ошибки можно причислить и калькуляторы, да и почти любую цифровую технику) делают это по-разному и имеют различные наборы команд. Знание особенностей системы команд компьютера, конечно, нужно каждому программисту, хотя оно выходит за пределы собственно программирования — чтобы писать программы, в общем-то не обязательно знать, каким образом компьютер выполняет команду сравнения чисел или их деления с остатком. Кроме того, полное понимание процессов, реально происходящих в компьютере во время выполнения программы, требует знания массы технических деталей, связанных с устройством процессора и других

микросхем, составляющих компьютер, и выходит за рамки собственно компьютерной арифметики, относясь к области, которую можно назвать архитектурой вычислительных систем.

Тем не менее компьютерные процессоры и другие большие цифровые микросхемы («чипы») содержат в себе сравнительно небольшие микросхемы для выполнения отдельных арифметических или логических операций. Разобраться в их устройстве, по крайней мере в общих принципах функционирования, не так уж и сложно, потому что электронные микросхемы, как старые, так и самые современные, имеют очень простую математическую модель — схему из функциональных логических элементов, часто называемую в инженерной практике комбинационной схемой, а в теории — булевой логической схемой, или просто схемой (реализующей булевы функции, называемые также функциями алгебры логики). Построение булевых схем, реализующих арифметические и логические операции, относится к сфере первоочередных интересов компьютерной арифметики, и в тоже время составляет существенный раздел теории сложности булевых функций. Впрочем, вопросы связанные с поиском эффективных алгоритмов построения булевых схем и с автоматизацией проектирования реальных электронных микросхем содержат множество технических деталей и выходят за рамки как компьютерной арифметики, так и теории сложности булевых функций, относясь к специальной области компьютерных наук — автоматизации проектирования микросхем. Разумеется, касаться этой области в популярной книге неуместно.

В программировании иногда возникает необходимость выполнять арифметические действия с «длинными» числами, не уместяющимися в разрядную сетку компьютера, например в научных расчетах для достижения требуемой точности, в криптографии для выполнения действий с числами, состоящими из сотен цифр, в рекордных вычислениях, например, с целью найти миллиард знаков числа π и т. д. Для выполнения этих действий нельзя непосредственно использовать команды машинной арифметики, а необходимо писать программы для так называемой длинной арифметики. Такие программы давно написаны и входят в пакеты многих си-

стем компьютерной алгебры. Чтобы самому написать подобные программы и разобраться в работе известных программ нужна компьютерная арифметика. На самом деле используемые в этих программах арифметические алгоритмы весьма близки к алгоритмам, «защитым в железе» электронных микросхем для выполнения арифметических операций. Но вместо двоичной системы счисления, которая используется в микросхемах (хотя используются и микросхемы, реализующие с помощью двоичных элементов десятичную арифметику), в компьютерных программах «длинной арифметики» обычно применяется система счисления с основанием 2^w , где w — длина машинного слова применяемого компьютера, а для записи самих программ вместо двоичной удобно использовать шестнадцатеричную систему (так как это в четыре раза сокращает длину записи чисел).

Использование подходящей (как правило, того или иного варианта позиционной) системы счисления является простейшей, старейшей, и в то же время важнейшей и фундаментальной идеей, используемой в компьютерной арифметике. Поэтому знакомство с ней естественно начинать с рассмотрения различных систем счисления. Книга «Занимательная компьютерная арифметика: Математика и искусство счета на компьютерах и без них» с этого и начинается, причем содержит почти целиком (и в несколько дополненном виде) материал из брошюры автора «Системы счисления и их применения». На самом деле упомянутая брошюра и задумывалась как первая часть популярной книги по компьютерной арифметике, но ее написание несколько затянулось, и первую часть пришлось опубликовать отдельно. Арифметические алгоритмы в нашей книге обычно излагаются для произвольной позиционной системы, но иногда они даются только для десятичной системы (чтобы подчеркнуть удобство их применения в ручных вычислениях, и даже в устном счете), а иногда — только для двоичной (потому что в двоичной системе они имеют наиболее простой вид и наиболее широкую область применения).

Кроме чисто арифметических алгоритмов уместно показалось рассмотреть в книге некоторые вопросы, связанные с вычислениями

с обыкновенными дробями, многочленами и дробно-рациональными функциями. Эти вопросы являются пограничными между компьютерной алгеброй и компьютерной арифметикой и весьма близки к последней. Их включение в книгу оправдано также тем, что, например, алгоритмы арифметических действий с многочленами на самом деле проще аналогичных алгоритмов для чисел.

Особенностью книги является также то, что, помимо описания различных алгоритмов, часто дается обоснованная оценка их сложности (т. е. числа выполняемых ими операций). В случае логических схем оценка их сложности позволяет объективно судить о размерах, которые может иметь реальная электронная микросхема, моделируемая рассматриваемой логической схемой. В случае же программных алгоритмов оценка их сложности позволяет судить о времени их работы. Умение оценивать сложность алгоритма, таким образом, полезно и в программировании и в проектировании (и производстве) электронной техники. Разделы, связанные с оценками сложности, находятся в конце книги и составляют, по-видимому, наиболее сложную ее часть (извините за каламбур). Это не удивительно, поскольку они находятся на границе между собственно компьютерной арифметикой и теорией сложности вычислений.

Чтобы компенсировать неизбежную сложность некоторых разделов книги, в нее показалось разумным вставить различный развлекательный материал, вроде приемов устных вычислений, интересных даже младшеклассникам, алгоритмам вычислений на калькуляторах, применений систем счисления за пределами компьютерной арифметики, исторических сведений о первооткрывателях тех или иных идей, затрагиваемых в книге и т. д. (однако точные ссылки на используемые результаты, как правило, не даются). Рассмотрены также вопросы эффективного выполнения некоторых манипуляций с числами и машинными словами на компьютере в условиях, когда отсутствуют команды, непосредственно их выполняющие. Подобные вопросы несомненно также относятся к компьютерной арифметике.

Книга, претендующая на популярность, не может освещать рассматриваемые вопросы в достаточной полноте. Поэтому некоторые вопросы, связанные с компьютерной арифметикой, затронуты лишь мимоходом или не затронуты вовсе. Это и вопросы оценки точности машинных вычислений, арифметика с плавающей запятой, логические схемы с памятью (автоматные схемы), алгоритмы вычисления элементарных функций, принципы работы компьютерного процессора, физические принципы работы электронных микросхем и их математического моделирования, вопросы проектирования, тестирования и производства чипов и т. д. Многим из подобных вопросов уже посвящаются целые книги (разумеется, не научно-популярные). Из великого множества алгоритмов компьютерной арифметики в книге упоминается лишь небольшая часть.

Предполагается, что читатель немного знает школьную математику (сейчас изучает ее или еще не успел забыть), поэтому иногда в книге используются без пояснений некоторые понятия и обозначения. Но во многих случаях необходимые пояснения даются. В отличие от учебников изложение не всегда систематичное, одним вопросам уделяется больше внимания, другим — меньше, некоторые темы даже не упоминаются, в книге встречаются ссылки как назад, так и вперед и т. д. Все это неизбежные особенности вольного стиля изложения (вообще свойственного, как нам кажется, научно-популярной литературе). Но из-за почти полного отсутствия на русском языке литературы по компьютерной арифметике автор надеется, что ее можно будет использовать и с учебной целью как в школах, так и в вузах.

В заключение автор благодарит своих бывших учеников (и нынешних коллег) А. А. Бурцева и И. С. Сергеева за помощь в работе.

Глава 1

Школьные алгоритмы арифметических операций с многочленами

<...> and then the different branches of arithmetic — Ambition, Distraction, Uglification and Derision ¹⁾.

Льюис Кэрролл.
Алиса в стране чудес

Как ни странно на первый взгляд, проще выполнять арифметические операции с многочленами, чем с натуральными числами. Читатель сейчас же в этом убедится. Однако в школе полиномиальную арифметику проходят позднее обычной и в недостаточном объеме. Дело в том, что строгое определение понятия многочлена несколько абстрактно и требует предварительного определения понятий поля и кольца. Мы будем пользоваться менее строгим, но интуитивно ясным следующим определением.

Многочлен над кольцом K — это выражение вида

$$p(x) = u_n x^n + \dots + u_1 x + u_0,$$

где коэффициенты u_i принадлежат кольцу K , а переменная x — формальный символ, который может принимать любое нужное значение. Под кольцом K понимаем любое множество с операциями сложения, вычитания, умножения, для которых выполняются все общеизвестные правила арифметики. Например, кольцом

¹⁾ <...> а затем проходили четыре действия арифметики — служение, почтение, угождение и давление.

является множество всех целых чисел с обычными арифметическими операциями, обычно его обозначают \mathbb{Z} . Множество всех многочленов над K обычно обозначают $K[x]$. Естественно считать многочлены

$$p(x) = u_n x^n + \dots + u_1 x + u_0$$

и

$$p'(x) = 0 \cdot x^{n+m} + \dots + 0 \cdot x^{n+1} + u_n x^n + \dots + u_1 x + u_0$$

равными, хотя формально они имеют разный вид. Обычно предполагают все же, что в записи

$$p(x) = u_n x^n + \dots + u_1 x + u_0$$

старший коэффициент $u_n \neq 0$. Его обозначают $ld(p)$, а число n называют степенью многочлена и обозначают $\deg p$. Если $ld(p) = 1$, то многочлен называют нормированным. Многочлены нулевой степени назовем константами и отождествим с элементами кольца K .

Для того, чтобы определить сумму и разность многочленов $p(x)$ и $q(x)$, удобно дополнить, если это нужно, один из них старшими членами с нулевыми коэффициентами так, чтобы они приняли вид

$$p(x) = u_n x^n + \dots + u_1 x + u_0$$

и

$$q(x) = v_n x^n + \dots + v_1 x + v_0.$$

Тогда

$$p(x) + q(x) = (u_n + v_n)x^n + \dots + (u_1 + v_1)x + (u_0 + v_0),$$

$$p(x) - q(x) = (u_n - v_n)x^n + \dots + (u_1 - v_1)x + (u_0 - v_0).$$

Для определения произведения многочленов $p(x)$ и $q(x)$ удобно считать, что их старшие коэффициенты ненулевые. Пусть

$$p(x) = u_n x^n + \dots + u_1 x + u_0,$$

$$q(x) = v_m x^m + \dots + v_1 x + v_0.$$

Тогда произведение

$$p(x)q(x) = w_{n+m}x^{n+m} + \dots + w_1x + w_0,$$

где

$$w_k = u_0v_k + u_1v_{k-1} + \dots + u_{k-1}v_1 + u_kv_0,$$

и, естественно, в этой формуле $u_i = 0$ при $i > n$ и $v_j = 0$ при $j > m$. В частности,

$$w_0 = u_0v_0, w_{n+m} = u_nv_m.$$

Отсюда следует, что

$$\deg(p(x)q(x)) = \deg p(x) + \deg q(x),$$

если в кольце K произведение ненулевых элементов всегда отлично от нуля (такие кольца называют кольцами без делителей нуля, примером является кольцо \mathbb{Z}). Для произвольных колец K выполнено лишь неравенство

$$\deg(p(x)q(x)) \leq \deg p(x) + \deg q(x),$$

а также то, что произведение нормированных многочленов нормировано.

Задача 1. Докажите, что множество $K[x]$ с введенными операциями является кольцом, содержащим K в качестве подкольца. Докажите, что если K не имеет делителей нуля, то $K[x]$ — тоже.

Кольцо F называют полем, если в нем всегда возможно деление на ненулевое число (напомним, что деление — это операция, обратная умножению). Примером поля служит множество \mathbb{Q} рациональных чисел, и множество \mathbb{R} действительных чисел. Поля являются кольцами без делителей нуля.

Задача 2. Докажите, что $K[x]$ не является полем.

Хотя деление в кольце $K[x]$ невозможно, но в случае когда K — поле, всегда можно выполнить деление с остатком. Говорят, что при делении многочлена $p(x)$ на $q(x)$ получается частное $h(x)$ и остаток $r(x)$, если $p(x) = q(x)h(x) + r(x)$, и $\deg r(x) < \deg q(x)$.

Задача 3. Докажите, что в кольце $F[x]$ деление с остатком выполнимо всегда и однозначным образом. Докажите, что если $p(x)$ и $q(x)$ принадлежат $K[x]$, K не имеет делителей нуля и $q(x)$ нормированный, то деление с остатком тоже выполнимо и однозначным образом.

Обозначим $M(m, n)$ наименьшее количество арифметических операций, выполняемых над числами, которые требуются для перемножения многочлена степени меньшей m и многочлена степени меньшей n . Величину $M(m, n)$ и подобные ей будем называть сложностью (в данном случае операции умножения многочленов указанных степеней).

Для краткости положим $M(n) = M(n, n)$. Обозначим $D(n, m)$ сложность операции деления многочлена степени $n - 1$ на многочлен степени $m - 1$ с нахождением частного и остатка (в случае $n < m$ очевидно, что $D(n, m) = 0$).

Для сложности операций сложения и вычитания многочленов степени меньшей n справедливы очевидные равенства $A(n) = n$ и $S(n) = n$.

Теорема 1. Школьный алгоритм умножения многочленов имеет сложность

$$M(n, m) \leq 2nm - n - m + 1.$$

Школьный алгоритм деления многочленов имеет сложность

$$D(n, m) \leq 2m(n - m) + n + m + 1.$$

Если делитель нормированный, то получается оценка

$$D(n, m) \leq 2m(n - m + 1).$$

Доказательство. Под школьным алгоритмом умножения многочленов

$$p(x) = u_{n-1}x^{n-1} + \dots + u_1x + u_0$$

и

$$q(x) = v_{m-1}x^{m-1} + \dots + v_1x + v_0$$

подразумевается умножение $p(x)$ на все одночлены многочлена $q(x)$, что требует mn операций умножения коэффициентов, и последующее сложение результатов, которое требует $(m-1)(n-1)$ операций сложения. Под школьным алгоритмом деления многочлена $p(x)$ на $q(x)$ подразумевается умножение $q(x)$ на $(u_{n-1}/v_{m-1})x^{n-m}$, которое требует одной операции деления и m операций умножения, и последующее вычитание из $p(x)$ произведения $(u_{n-1}/v_{m-1})x^{n-m}q(x)$, которое требует m вычитаний, и дает в результате многочлен $p_1(x)$ степени не выше $n-2$. Одночлен $(u_{n-1}/v_{m-1})x^{n-m}$ является старшим членом частного, остальные члены которого и остаток находятся в результате деления $p_1(x)$ на $q(x)$. Отсюда имеем неравенство

$$D(n, m) \leq D(n-1, m) + 1 + 2m,$$

из которого, учитывая, что

$$D(m-1, m) = 0,$$

выводим неравенство

$$D(n, m) \leq (1 + 2m)(n - m + 1) = 2m(n - m) + n + m + 1,$$

Интересно отметить, что при растущем m оценка сложности деления даже ниже, чем у умножения. Если многочлен q нормирован, $n - m + 1$ операций деления фактически делать не приходится и оценка сложности еще понижается.

В случае $K = \mathbb{Z}$ можно ввести понятие *псевдоделения*. *Псевдо-частным* и *псевдоостатком* при делении $p(x)$ на $q(x)$ называются такие многочлены $h(x)$ и $r(x)$, для которых

$$\deg r(x) < \deg q(x)$$

и

$$ld(q)^{\deg p - \deg q + 1} p(x) = q(x)h(x) + r(x).$$

□

Обозначим $l(n)$ наименьшее число умножений, требующееся для возведения x в n -ю степень. (Эта функция встречается в разделе «Аддитивные цепочки и флаги с молоком» в *Гашков С. Б. Занимательная компьютерная арифметика*. М.: URSS, 2012.)

Теорема 2. Псевдочастное и псевдоостаток всегда существуют и определены однозначно. Сложность $D_p(n, m)$ псевдоделения многочлена степени $n - 1$ на многочлен степени $m - 1$ удовлетворяет неравенствам

$$D_p(n, m) \leq D(n, m) + n + l(n - m + 1),$$

$$D(n, m) \leq D_p(n, m) + 2n + l(n - m + 1).$$

Доказательство. Пусть

$$p(x) = u_{n-1}x^{n-1} + \dots + u_1x + u_0$$

и

$$q(x) = v_{m-1}x^{m-1} + \dots + v_1x + v_0.$$

Обычный алгоритм деления находит частное h_0 и остаток r_0 , для которых $p(x) = q(x)h_0(x) + r_0(x)$ и $\deg r_0(x) < \deg q(x)$, причем коэффициенты частного и остатка являются дробями со знаменателем $(v_{m-1})^{n-m+1}$. Тогда в качестве псевдочастного и псевдоостатка можно взять многочлены с целыми коэффициентами

$$h(x) = v_{m-1}^{n-m+1} h_0(x), \quad r(x) = v_{m-1}^{n-m+1} r_0(x).$$

Действительно, умножая обе части равенства

$$p(x) = q(x)h_0(x) + r_0(x)$$

на

$$v_{m-1}^{n-m+1} = ld(v)^{\deg p - \deg q + 1},$$

получаем равенство

$$ld(q)^{\deg p - \deg q + 1} p(x) = q(x)h(x) + r(x).$$

Однозначность псевдочастного и псевдоостатка следует из того, что они являются частным и остатком от деления многочлена $ld(q)^{\deg p - \deg q + 1} p(x)$ на $q(x)$. Коэффициенты первого из них можно

вычислить со сложностью не выше $n + l(n - m + 1)$, а последующее деление — со сложностью $D(n, m)$. Отсюда

$$D_p(n, m) \leq D(n, m) + n + l(n - m + 1).$$

Если мы нашли псевдочастное и псевдоостаток $h(x)$ и $r(x)$, то частное и остаток находятся по формулам

$$\frac{h(x)}{(v_{m-1}^{n-m+1})} = h_0(x), \quad \frac{r(x)}{(v_{m-1}^{n-m+1})} = r_0(x)$$

со сложностью не большей $l(n - m + 1) + n - m + 1 + m - 1$, откуда

$$D(n, m) < D_p(n, m) + 2n + l(n - m + 1). \quad \square$$

Задача 4. Покажите, что алгоритм псевдоделения, чуть увеличив его сложность, можно выполнить без операции деления чисел.

Глава 2

Школьные алгоритмы сложения и умножения чисел

Полковник: Сержант, почему мало оштрафованных?

Сержант: Виноват, задумался, почему три плюс три — шесть, а трижды три — девять.

Полковник: Ты сюда поставлен не складывать и умножать, а отнимать и делить!

Из анекдота о ГИБДД

При оценке сложности операций над натуральными числами надо вначале договориться о том, как задавать эти числа. Для этого мы будем использовать классическую позиционную систему с произвольным натуральным основанием b . Под b -ичной позиционной записью числа n понимается его единственное представление в виде

$$n = n_m b^m + \dots + n_1 b + n_0,$$

где все n_i — целые неотрицательные и меньше b . Для дальнейшего объяснения удобно здесь ввести функции $\nu_b(n)$ — сумму цифр b -ичной позиционной записи числа n , и $\lambda_b(n)$ — длину этой записи. Очевидно, что

$$\nu_b(n) = n_0 + \dots + n_m, \quad \lambda_b(n) = \lceil \log_b(n+1) \rceil.$$

Задача 5. Докажите неравенство

$$\lambda_b(n) + \lambda_b(m) - 1 \leq \lambda_b(nm) \leq \lambda_b(n) + \lambda_b(m).$$

Задача 6. Вычислите суммы

$$\lambda_b(1) + \dots + \lambda_b(n) \quad \text{и} \quad \nu_b(1) + \dots + \nu_b(n).$$

Обозначим $M(m, n)$ наименьшее количество операций сложения, вычитания и умножения, выполняемых над цифрами, которые требуются для перемножения m -значного и n -значного чисел и положим для краткости $M(n) = M(n, n)$.

Значение величины $M(m, n)$ существенно зависит от того, что понимать под элементарными операциями. Будем считать, что операция с сложения двух цифр x и y дает, вообще говоря, двузначное число, младший разряд которого $c_1(x, y)$ равен $x + y \bmod a$, а старший разряд $c_2(x, y)$ равен $p_a(x + y)$, где $p_a(n) = 1 \Leftrightarrow n \geq a$.

Операция умножения двух цифр x и y дает тоже двузначное число, младший разряд которого $\mu_1(x, y)$ равен $x \cdot y \bmod a$, а старший разряд $\mu_2(x, y)$ равен $\lfloor xy/a \rfloor$. Очевидны следующие связи между введенными функциями: $x + y \bmod a$ (сумма по модулю a) равна $x + y - ap_a(x + y)$, а $x \cdot y \bmod a$ (произведение по модулю a) равно $xy - a \lfloor xy/a \rfloor$. Тогда справедлива следующая лемма.

Лемма 1. Сложение двух n -значных чисел можно выполнить за $2n - 1$ операцию, вычитание из большего меньшее — за $2n - 1$ операцию, а сложение n -значного числа с m -значным (при $n > m$) можно выполнить за $n + m - 1$ операцию.

Доказательство. Определим трехместный оператор $S(x, y, z)$ равенствами

$$S_1(x, y, z) = x + y + z \bmod a,$$

$$S_2(x, y, z) = p_a(x + y + z).$$

Непосредственно проверяется, что

$$S_1(x, y, z) = c_1(c_1(x, y), z),$$

$$S_2(x, y, z) = c_2(x, y) \vee c_2(c_1(x, y), z),$$

где операция дизъюнкции $x \vee y = \max(x, y)$. Из этих равенств видно, что S вычисляется с помощью двух операций сложения и операции дизъюнкции (применяемой только к числам 0 или 1). Далее мы не будем учитывать при вычислении сложности операции дизъюнкции, так как в компьютерных вычислениях время их выполнения мало по сравнению со временем сложения, а в ручных вычислениях (в том числе и с помощью непрограммируемого калькулятора) она вообще не замечается. Непосредственно проверяется, что оператор

$$A(x_1, \dots, x_n, y_1, \dots, y_n),$$

осуществляющий сложение n -разрядных чисел $\overline{x_n \dots x_1}$ и $\overline{y_n \dots y_1}$ и дающий в результате $(n+1)$ -разрядное число $\overline{z_{n+1} \dots z_1}$, где

$$z_i = A_i(x_1, \dots, x_n, y_1, \dots, y_n), \quad 1 \leq i \leq n+1,$$

вычисляется схемой, составленной из $2n-1$ операции

$$\begin{aligned} (z_1, s_1) &= c(x_1, y_1), \\ (z_2, s_2) &= S(x_2, y_2, s_1), \\ &\dots\dots\dots, \\ (z_n, s_n) &= S(x_n, y_n, s_{n-1}), \end{aligned}$$

В случае $y_n = \dots = y_{m+1} = 0$ в силу равенства $S(x, 0, z) = c(x, z)$ операций нужно на $n-m$ меньше, т.е. для сложения n -значного числа с m -значным достаточно

$$2n-1-(n-m) = n+m-1 \text{ операции.}$$

Для вычисления оператора вычитания годится та же схема, только в определении оператора S и операции c надо заменить сумму по модулю a на разность по модулю a . При этом s_i будет величиной займа из следующего разряда, а s_{n+1} задает знак разности. Если $s_{n+1} = 1$, то он будет отрицательный (тогда для вычисления модуля разности надо заменить все разряды результата на их дополнения до числа $a-1$, кроме младшего, который надо заменить на дополнение до a). \square

Лемма 2. Справедливо неравенство

$$M(n, 1) \leq 3n - 2.$$

Доказательство. Рассмотрим оператор $\Pi(x_1, \dots, x_n, y)$ умножения n -разрядного числа $\overline{x_n \dots x_1}$ на цифру y , в результате которого получается $(n + 1)$ -разрядное число $\overline{z_{n+1} \dots z_1}$, где

$$z_i = \Pi_i(x_1, \dots, x_n, y), \quad 1 \leq i \leq n + 1.$$

Он вычисляется схемой, составленной из $3n - 2$ операций

$$(z_1, s_1) = (\mu_1(x_1, y), 0)(z_2, s_2) = S(\mu_1(x_2, y), \mu_2(x_1, y), s_1),$$

.....,

$$(z_n, s_n) = S(\mu_1(x_n, y), \mu_2(x_{n-1}, y), s_{n-1}) = c((\mu_2(x_n, y), s_n), 0),$$

так как для вычисления оператора

$$\begin{aligned} S(\mu_1(x_2, y), \mu_2(x_1, y), s_1) &= S(\mu_1(x_2, y), \mu_2(x_1, y), 0) = \\ &= c(\mu_1(x_2, y), \mu_2(x_1, y)) \end{aligned}$$

требуется не три, а только две операции. □

Теорема 3. Школьный алгоритм умножения дает оценку

$$M(n, m) \leq 5nm - 2n - m - 2.$$

Доказательство. Согласно школьному алгоритму для умножения n -разрядного числа на m -разрядное нужно m раз применить оператор

$$\Pi(x_1, \dots, x_n, y)$$

(это потребует $m(3n - 2)$ операций) и $m - 1$ раз — оператор

$$A(x_1, \dots, x_{n+1}, y_1, \dots, y_{n+1})$$

(для этого требуется $(m - 1)(2n + 1) - 1$ операций при $m > 1$, так как при первом применении оператора первый операнд на один разряд

короче, и поэтому на 1 операцию нужно меньше). При выполнении операций в двоичной системе пренебрегать операцией дизъюнкции представляется неестественным, более того, в этом случае естественно считать, что операция сложения s состоит из операций $x + y \bmod 2$ и $p_2(x + y) = x \cdot y \bmod 2 = x \& y$, а операция умножения μ сводится только к одной операции конъюнкции $x \& y$. Тогда леммы 1 и 2 превращаются в соотношения

$$A(n) \leq 5n - 3, \quad S(n) \leq 5n - 3,$$

$$A(n, m) \leq 2(n + m) + \min(m, n) - 3, \quad M(n, 1) = n,$$

где через A и S обозначены сложности сложения и вычитания. Можно доказать, что все эти неравенства следует заменить на равенства. \square

Доказанная теорема в двоичном случае заменяется неравенством

$$M(n, m) \leq 6nm - 5n - 3m.$$

Пользуясь тем, что от перестановки сомножителей произведение не меняется, эту оценку можно заменить на следующую:

$$M(n, m) \leq 6nm - 3(n + m) - 2 \max(m, n),$$

и подобным же образом заменить оценку теоремы 3. Это означает, что столбиком чуть быстрее умножать более длинное число на менее длинное.

Величина констант в полученных оценках сложности умножения особого значения не имеет, так как эти неравенства чрезвычайно далеки от точных значений. Чтобы подчеркнуть порядок указанных оценок, сложность как умножения чисел, так и умножения многочленов, можно записать в виде $M(n, m) = O(n, m)$, где символ O , например в равенстве $f = O(g)$, означает, что при некоторой константе C справедливо неравенство $f \leq Cg$.

Задача 7. Докажите последнее утверждение.

Задача 8. С какой сложностью можно возвести в квадрат многочлен степени меньшей n , используя школьный метод?

Задача 9. Докажите, что n -разрядное двоичное число возводится в квадрат школьным методом со сложностью не больше $5,5n^2 - 8,5n$. Попробуйте улучшить эту оценку до $3n^2 + O(n)$, учитывая, что складывать надо не n^2 чисел, а приблизительно вдвое меньшее количество.

Глава 3

Умножение столбиком нескольких чисел

*Сом и делит, сом и множит, сом
сложить и вычесть может.*

Борис Заходер. Сомомнение

Обобщение полученных выше оценок на случай сложения или умножения многих чисел не представляет труда, если интересоваться только порядком величины, но вызывает трудности уже в случае сложения при попытке получить достаточно точные константы. Отметим, что при больших числах n_i школьный алгоритм умножения имеет асимптотически одинаковую сложность при любом способе расстановки скобок в произведении.

Покажем, что сложность $M(n_1, \dots, n_k)$ школьного алгоритма умножения чисел с длинами записи n_1, \dots, n_k зависит только от порядка сомножителей, но не зависит от порядка выполнения операций умножения. Положим для краткости

$$s(n_1, \dots, n_k) = \sum_{1 \leq i < j \leq k} n_i n_j,$$
$$l(n_1, \dots, n_k) = \sum_{i=1}^{k-1} (2i-1)n_i + (2k-2)n_k.$$

Теорема 4. Справедливо равенство

$$M(n_1, \dots, n_k) = 5s(n_1, \dots, n_k) - l(n_1, \dots, n_k) - 2k + 2.$$

Сложность умножения будет минимальна, когда $n_1 \leq n_2 \leq \dots \leq n_k$, т. е. выгоднее умножать числа, начиная с самых больших.

Доказательство. Пусть надо перемножить n_i -значные числа x_i . Рассмотрим все возможные формулы Φ , указывающие порядок выполнения операций. Определим понятие формулы по индукции. Произвольную переменную x_i назовем формулой. Если Φ_1 и Φ_2 — формулы, то $\Phi = (\Phi_1 \circ \Phi_2)$ тоже назовем формулой. Будем рассматривать только формулы, содержащие все переменные x_i по одному разу. Формулу Φ будем изображать бинарным деревом T с листьями n_1, \dots, n_k . Определим T по индукции. Если Φ есть переменная x_i , то T есть лист n_i . Если формулам Φ_j , $j = 1, 2$, сопоставлены деревья T_j , то формуле $\Phi = (\Phi_1 \circ \Phi_2)$ сопоставим дерево T , получающееся объединением T_j и присоединением их к корню дерева T с помощью двух ребер. Весом дерева T назовем $v(T)$ — сумму весов его листьев. Очевидно, что

$$v(T) = v(T_1) + v(T_2)$$

и $v(T)$ равно числу разрядов в произведении, вычисляемом формулой Φ .

Раскрасим все ребра дерева T в два цвета по индукции следующим образом: считая, что ребра деревьев T_j уже раскрашены согласно предположению индукции, покрасим ребро, соединяющее с корнем дерево T_1 , в белый цвет, если $v(T_1) < v(T_2)$, а второе ребро — в черный, и сделаем наоборот, если $v(T_1) \geq v(T_2)$. Каждый из листьев n_i соединяется в дереве с корнем единственным путем, состоящим из a_i белых и b_i черных ребер. Назовем длиной пути число $d_i = a_i + 2b_i$, а сумму $\sum_i d_i n_i$ назовем длиной дерева и обозначим ее $d(T)$. Докажем по индукции, что если мы перемножим числа x_i по формуле Φ , то сложность этого вычисления будет

$$5s(n_1, \dots, n_k) - d(T) - 2k + 2,$$

где n_1, \dots, n_k — это все листья дерева T . База индукции ($k = 2$) следует из теоремы 3. Допустим, что согласно предположению индукции сложность вычислений по формулам Φ_j равна

$$L(\Phi_j) = 5s(\Phi_j) - d(T_j) - 2k_j + 2,$$

где через Φ_j обозначен для краткости список всех k_j — листьев дерева T_j , $j = 1, 2$.

Тогда сложность вычисления по формуле Φ равна

$$L(\Phi) = L(\Phi_1) + L(\Phi_2) + 5v(T_1)v(T_2) - av(T_1) - bv(T_2) - 2,$$

где $a = 1$, $b = 2$, если ребро, идущее в корень дерева T_1 из корня дерева T , белое, и наоборот. Так как длины путей, ведущих к листьям в дереве T , длиннее на a , чем соответствующие пути в дереве T_1 , и длиннее на b , чем соответствующие пути в дереве T_2 , то

$$d(T) = d(T_1) + d(T_2) + av(T_1) + bv(T_2).$$

Непосредственно проверяется, что

$$s(n_1, \dots, n_k) = s(\Phi_1) + s(\Phi_2) + v(T_1)v(T_2), \quad k = k_1 + k_2.$$

Поэтому

$$\begin{aligned} L(\Phi) &= \sum_{j=1}^2 (5s(\Phi_j) - d(T_j) - 2k_j + 2) + \\ &\quad + 5v(T_1)v(T_2) - av(T_1) - bv(T_2) - 2 = \\ &= 5(s(\Phi_1) + s(\Phi_2) + v(T_1)v(T_2)) - \\ &\quad - 2(k_1 + k_2) + 2 - (d(T_1) + d(T_2) + av(T_1) + bv(T_2)) = \\ &= 5s(n_1, \dots, n_k) - d(T) - 2k + 2, \end{aligned}$$

что и требовалось доказать. \square

Теперь нужно максимизировать $d(T)$. Для этого потребуются две леммы.

Лемма 3. Если $a_{[i]}$ и $b_{[i]}$ — упорядоченные в порядке возрастания последовательности a_i и b_i , то

$$\sum_{i=1}^n a_{[i]} b_{[n-i+1]} \leq \sum_{i=1}^n a_i b_i \leq \sum_{i=1}^n a_{[i]} b_{[i]}.$$

Доказательство. Без ограничения общности считаем, что $a_i = a_{[i]}$. Пусть b'_i — такая перестановка b_i , для которой $\sum_{i=1}^n a_i b'_i$ макси-

мальна. Если $b'_i > b'_j$ и $i < j$, то, меняя местами b'_i и b'_j , получаем перестановку b''_i , для которой

$$\sum_{i=1}^n a_i b''_i - \sum_{i=1}^n a_i b'_i = a_i b'_j + a_j b'_i - a_i b'_i - a_j b'_j = (a_j - a_i)(b'_i - b'_j) > 0,$$

что ведет к противоречию. \square

Лемма 4. Максимум $d(T)$ по всем деревьям T с листьями n_1, \dots, n_k равен

$$l(n_1, \dots, n_k) = \sum_{i=1}^k (2i - 1)n_i + (2k - 2)n_k.$$

Доказательство. Доказательство проведем индукцией по k . База ($k = 2$) очевидна. Пусть T — дерево, обеспечивающее максимум. Докажем, что T можно выбрать так, что d_i не убывают. Допустим противное: $d_i > d_j$, $i < j$. Поменяем местами n_i и n_j . Согласно лемме 3 $d(T)$ при этом не уменьшается, но раскраска может стать неправильной. Заменяем ее на правильную и заметим, что $d(T)$ не уменьшится.

Действительно, если в некотором поддереве T , состоящем из поддеревьев T_j , выполнено неравенство $v(T_1) \geq v(T_2)$, но ребро, ведущее в T_1 , белое, то после перекраски оно станет черным, и для нового раскрашенного дерева T' длина

$$\begin{aligned} d(T') &= d(T_1) + d(T_2) + 2v(T_1) + v(T_2) = \\ &= d(T_1) + d(T_2) + v(T_1) + 2v(T_2) + v(T_1) - v(T_2) = \\ &= d(T) + v(T_1) - v(T_2) \geq d(T). \end{aligned}$$

После выполненных операций длина дерева не может возрасти, так как она максимальна. Поэтому $n_i = n_j$ и такие значения можно переставлять между собой в максимальном дереве как угодно. Переставим их так, чтобы последовательность d_i стала монотонной.

Зная теперь, что d_i не убывает, заметим, что соседний с листом n_k лист n_i соединяется ребром с той же вершиной дерева,

что и n_k , иначе путь до листа n_i был бы длиннее. Выбросим из T эти два ребра и получим дерево T' с листьями $n_1, \dots, \widehat{n_i}, \dots, n_i + n_j$ (знак $\widehat{n_i}$ означает пропуск числа n_i). Тогда

$$d(T) = d(T') + n_i + 2n_k,$$

значит, T максимально тогда и только тогда, когда максимально T' .

Но по предположению индукции T' — линейное дерево с длинами листьев $1, 3, \dots, 2k-5, 2k-4$. Тогда T — тоже линейное дерево с длинами листьев $1, 3, \dots, 2k-3, 2k-2$. Если $i < k-1$, то оно не будет максимальным, так как при перестановке «младших» листьев его длина увеличивается (а раскраска остается правильной). Поэтому $i = k-1$ и лемма доказана. \square

Из леммы немедленно следует утверждение теоремы.

В случае же сложения надо поступать наоборот: начинать сложение с самых маленьких чисел и заканчивать самыми большими.

Задача 10. Докажите, что

$$\begin{aligned} A(n_1, \dots, n_k) &\leq n_1 + 2(n_2 + \dots + n_{k-1}) + n_k + \lambda_b(1) + \dots + \lambda_b(k-1) - k + 1 < \\ &< n_1 + 2(n_2 + \dots + n_{k-1}) + n_k + (k-1)(\lambda_b(k-1) - 1). \end{aligned}$$

Указание. Складывайте числа, начиная с маленьких. Примените лемму 2 и неравенство $\lambda_b(x_1 + \dots + x_k) \leq \lambda_b(\max x_i) + \lambda_b(k)$.

Задача 11. Докажите, что если складывать k чисел, начиная с самых больших, то оценка сложности может возрасти (асимптотически) в $(k-1)$ раз.

Глава 4

Переносы при сложении двоичных чисел и теорема Куммера

*Большинство специалистов по теории чисел
не проявляют интереса к арифметике.*

Бересфорд Парлетт, 1979
(цитируется по книге Д. Кнута
«Искусство программирования»)

Последние две задачи предыдущего раздела справедливы только для оценки сложности компьютерных вычислений. В ручных же вычислениях переносы в старшие разряды в действительности не выполняются, если они оказываются нулевыми. Разберемся в этом вопросе.

Лемма 5. Если b -ичная запись числа n оканчивается ровно m нулями, то

$$\nu_b(n-1) + 1 - \nu_b(n) = (b-1)m,$$

и при прибавлении к числу $n-1$ единицы m будет равно количеству переносов в следующий разряд (возможно, $m=0$).

Доказательство. Заметим, что b -ичная запись числа $n-1$ оканчивается ровно m цифрами $b-1$. После прибавления единицы происходит m -кратный перенос в старшие разряды и получается число n , у которого все цифры, кроме $m+1$ последних, совпадают с такими же цифрами числа $n-1$, причем $(m+1)$ -я от конца цифра на 1 больше такой же цифры числа $n-1$, а последние m цифр — нули. □

Теорема 5 (теорема Куммера¹⁾). Количество переносов в следующий разряд при сложении чисел k и $n - k$ в b -ичной системе счисления равно

$$\frac{\nu_b(n - k) + \nu_b(k) - \nu_b(n)}{b - 1}.$$

Доказательство. Утверждение вытекает из приведенной выше леммы. Действительно, если переносов не происходило, то величина

$$\frac{\nu_b(n - k) + \nu_b(k) - \nu_b(n)}{b - 1}$$

равна нулю. Каждый перенос уменьшает один разряд в числе n на b и увеличивает следующий разряд на 1, в результате рассматриваемая величина возрастает на $b - 1$, а значит, величина

$$\frac{\nu_b(n - k) + \nu_b(k) - \nu_b(n)}{b - 1}$$

возрастает на 1. □

Во втором томе «Искусства программирования» Д. Кнута показано, что для случайных независимых n -разрядных b -ичных чисел u и v среднее число переносов $(\nu_b(u) + \nu_b(v) - \nu_b(u + v))/(b - 1)$ равно

$$\frac{1}{2} \left(n - \frac{1 - b^{-n}}{b - 1} \right) < \frac{n}{2}.$$

¹⁾ Эрнст Эдуард Куммер (1810–1893) — знаменитый немецкий математик, доказавший теорему Ферма для многих значений n и заложивший основы теории алгебраических чисел.



Эрнст Эдуард Куммер

Значит, средняя сложность сложения n -разрядных чисел меньше $3n/2$. Отвлекаясь от темы, заметим, что на самом деле эта теорема была нужна Куммеру в связи со следующим фактом.

Наибольший показатель степени простого числа p , делящей биномиальный коэффициент

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

равен

$$\text{ord}_p \binom{n}{k} = \frac{\nu_p(n-k) + \nu_p(k) - \nu_p(n)}{p-1}.$$

Для доказательства этого факта полезна следующая формула Лежандра.

Теорема 6 (теорема Лежандра²⁾). Наибольший показатель степени простого числа p , делящей факториал $n!$, равен

$$\text{ord}_p n! = \frac{n - \nu_p(n)}{p-1}.$$

Доказательство. Действительно, если согласно предположению индукции

$$\text{ord}_p(n-1)! = \frac{n-1 - \nu_p(n-1)}{p-1}$$

и $\text{ord}_p(n) = m$, то p -ичная запись числа n оканчивается ровно m нулями, значит, согласно лемме 5

$$\text{ord}_p n! = \text{ord}_p(n-1)! + m = m + \frac{n-1 - \nu_p(n-1)}{p-1} = \frac{n - \nu_p(n)}{p-1}. \quad \square$$

Возвращаясь опять к сложению чисел, покажем, что сложность его выполнения на самом деле почти не зависит от порядка выполнения операций, если учитывать только те переносы в старшие разряды, которые в действительности выполняются.

²⁾ Адриен Мари Лежандр (1752–1833) — выдающийся французский математик.

Обозначим сложность вычисления суммы чисел n_1, \dots, n_k через $A(n_1, \dots, n_k)$. Для краткости положим

$$\nu_b(n_1, \dots, n_k) = \frac{1}{b-1} \left[\sum_{i=1}^k \nu_b(n_i) - \nu_b\left(\sum_{i=1}^k n_i\right) \right].$$

Пусть $n[i]$ — последовательность n_i , переставленная в неубывающем порядке.

Теорема 7. При любом порядке суммирования справедливо неравенство

$$A(n_1, \dots, n_k) \geq \sum_{i=1}^{k-1} \nu_b(n[i]) + \nu_b(n_1, \dots, n_k),$$

причем равенство достигается, если числа складывать, начиная с больших. При любом порядке суммирования справедливо также неравенство

$$A(n_1, \dots, n_k) \leq \sum_{i=1}^{k-1} \nu_b(n[i]) + \nu_b(n_1, \dots, n_k) \lambda_b(k).$$

Доказательство. Предположим, без ограничения общности, что последовательность n_i не убывает. Докажем по индукции неравенство

$$A(n_1, \dots, n_k) \geq \sum_{i=1}^{k-1} \nu_b(n_i) + \nu_b(n_1, \dots, n_k).$$

База ($k = 2$) очевидна в силу теоремы Куммера. Пусть сложение выполняется по формуле $\Phi = (\Phi_1 + \Phi_2)$, где Φ_1 вычисляет сумму $x = n_i + \dots + n_j + n_l$, а Φ_2 — сумму $y = n_s + \dots + n_q + n_r$. По предположению индукции сложности этих вычислений не меньше

$$\lambda_b(n_i) + \dots + \lambda_b(n_j) + \nu_b(n_i, \dots, n_j, n_l),$$

$$\lambda_b(n_s) + \dots + \lambda_b(n_q) + \nu_b(n_s, \dots, n_q, n_r).$$

Сложность вычисления суммы $\Phi_1 + \Phi_2$ не меньше

$$\min(\lambda_b(x), \lambda_b(y)) + \nu_b(x, y).$$

Складывая эти неравенства и учитывая равенство

$$\nu_b(x, y) + \nu_b(n_i, \dots, n_j, n_l) + \nu_b(n_s, \dots, n_q, n_r) = \nu_b(n_1, \dots, n_k),$$

получаем нужное нам неравенство. Достижимость этого неравенства при сложении чисел в порядке возрастания аналогичным образом устанавливается по индукции. Для доказательства верхней оценки

$$A(n_1, \dots, n_k) \leq \sum_{i=1}^{k-1} \nu_b(n[i]) + \nu_b(n_1, \dots, n_k) + \lambda_b(k)$$

складываем числа в произвольном порядке и применяем предыдущее равенство и неравенства

$$\lambda_b(n_1 + \dots + n_k) \leq \lambda_b(\max n_i) + \lambda_b(k),$$

$$\lambda_b(1) + \dots + \lambda_b(k) \leq k\lambda_b(k).$$

□

Задача 12. Докажите, что наибольший показатель степени простого числа p , делящей биномиальный коэффициент

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

равен

$$\text{ord}_p \binom{n}{k} = \frac{\nu_p(n-k) + \nu_p(k) - \nu_p(n)}{p-1}.$$

Указание. Утверждение следует из формулы Лежандра и формулы для биномиального коэффициента.

Глава 5

Минимальные формы двоичной записи с цифрами 0 и ± 1 и первая попытка уменьшить сложность умножения

А потом произошло кое-что еще — то, о чем у Люмаса не упоминалось ни словом: все буквы алфавита исчезли, превратившись в цифры, а потом и цифры — все, кроме единицы и нуля, исчезли, оставив мне целый океан нулей и единиц, которые так и сыпались по стенам вокруг меня.

Скарлетт Томас.

Наваждение Люмаса (2006)

В позиционных системах счисления по заданному основанию b можно, кроме обычных цифр, использовать и отрицательные цифры: $-1, -2, \dots, -(b-1)$. Правда, это приводит к неоднозначности в записи чисел. Зато таким образом можно уменьшить количество ненулевых цифр в записи и их величину.

Рассмотрим двоичную систему с отрицательными цифрами. Назовем запись данного числа в этой системе минимальной формой этого числа, если она содержит наименьшее число ненулевых цифр.

Задача 13. Покажите, что определенная таким образом минимальная форма не всегда единственна.

Определим минимальную форму по-другому. Назовем запись минимальной формой, если в ней нет соседних ненулевых цифр. Для этого определения не очевидна ни единственность минимальной формы, ни ее минимальность в прежнем смысле. Однако и то и другое верно.

Теорема 8. Минимальная форма определена однозначно и содержит наименьшее количество ненулевых цифр среди всех возможных форм двоичной записи числа с использованием отрицательных единиц.

Доказательство. Пусть $(a_n \dots a_0)$ — произвольная запись числа a , т. е.

$$a = 2^n a_n + \dots + 2a_1 + a_0,$$

где $a_i = 0$ или ± 1 . Далее вместо -1 будем писать $\bar{1}$. Обозначим $\nu(a)$ количество ненулевых цифр в этой записи, и $|a|$ — количество пар соседних ненулевых цифр. Заметим, что

$$2^k + 2^{k+1} + \dots + 2^{m-1} = 2^m - 2^k,$$

поэтому, выполняя в записи a следующие преобразования:

$$\alpha \bar{\alpha} \rightarrow 0\alpha,$$

$$0\alpha \dots \alpha \rightarrow \alpha 0 \dots 0\bar{\alpha},$$

$$00\alpha\alpha \rightarrow 0\alpha 0\bar{\alpha},$$

$$\alpha 0\bar{\alpha}\bar{\alpha} \rightarrow 0\alpha 0\alpha,$$

$$0\alpha\alpha 0\alpha\alpha \rightarrow \alpha 00\bar{\alpha} 0\bar{\alpha},$$

$$0(0\alpha) \dots (0\alpha) 0\alpha\alpha \rightarrow 0\alpha(0\bar{\alpha}) \dots (0\bar{\alpha}) 0\bar{\alpha},$$

$$\bar{\alpha}(0\alpha) \dots (0\alpha)(0\alpha)\alpha \rightarrow 0\bar{\alpha}(0\bar{\alpha}) \dots (0\bar{\alpha}) 0\bar{\alpha},$$

$$0\alpha\alpha(0\alpha) \dots (0\alpha 0)\alpha\alpha \rightarrow \alpha 00\bar{\alpha}(0\bar{\alpha}) \dots (0\bar{\alpha}) 0\bar{\alpha}$$

(где $\alpha = 1, \bar{1}$; $\bar{\alpha} = \bar{1}, 1$), мы не меняем записываемого числа, не увеличиваем величин $\nu(a)$ и $|a|$ и всегда уменьшаем их сумму. Будем выполнять эти преобразования, пока это возможно. Так как величина $\nu(a) + |a|$ не может неограниченно уменьшаться, то в конце концов получим запись числа a , в которой нигде не будут встречаться рядом ненулевые цифры. Действительно, разных цифр рядом быть не может, а также трех и более подряд одинаковых ненулевых цифр — тоже. Если пары соседних равных ненулевых

цифр встречаются, то рядом с ними не могут стоять пары нулей, и сами пары ненулевых цифр не могут стоять рядом, разделяясь одним нулем. Между ними не может также находиться ни цифра противоположного знака, ни пара нулей. И, наконец, пары ненулевых цифр, между которыми не встречается таких же пар, всегда совпадают, и промежуток между ними заполняется чередующимися нулями и цифрой, образующей эти пары. Но и такие комбинации цифр не могут встречаться в построенной записи, значит, она вообще не содержит пар ненулевых цифр.

Докажем, что запись, не содержащая таких пар, для данного числа только одна. Допустим, что есть две разные такие записи a и b . Тогда они заканчиваются одинаковым числом нулей в младших разрядах, иначе, если бы одна заканчивалась k нулями, а другая $m > k$ нулями, то наше число делилось бы на 2^m , а с другой стороны, делилось бы только на 2^k , но не на 2^{k+1} , что невозможно. Аналогично получаем, что их последние ненулевые цифры равны, так как в противном случае наше число имело бы при делении на 2^{m+2} (где m — число нулей в его конце) в остатке разные числа 2^m и $2^{m+2} - 2^m$ (так как в конце одной записи стоят цифры $010 \dots 0$, а в конце другой — цифры $0\bar{1}0 \dots 0$ ввиду отсутствия пар ненулевых цифр в обоих записях). Отбрасывая равные последние цифры от обеих записей, получаем более короткие различные записи для равных чисел. Повторяя для этих записей проведенное рассуждение, получим, наконец, что число ± 1 или 0 имеет две разные записи, а это невозможно.

И наконец, докажем, что построенная запись минимальна в том смысле, что содержит минимальное число ненулевых цифр. Действительно, из любой записи можно с помощью рассмотренных преобразований получить построенную запись (как только что доказано, всегда одну и ту же). Но при выполнении этих преобразований величина $\nu(a)$ не возрастала, значит, построенная запись имеет значение этой величины, равное наименьшему возможному. Теорема доказана. \square

Преобразование обычной двоичной записи числа a к минимальной форме более удобно проводить следующим образом:

вычислить обычную двоичную запись $\gamma_{n+1} \dots \gamma_1$ числа $3a$ и вычислить обычные разности $\delta_{i-1} = \gamma_i - \alpha_i$, где $i = 2, \dots, n+1$, α_i — цифры записи числа a , а $a_n = a_{n+1} = 0$, тогда $\delta_n \dots \delta_1$ — минимальная форма числа a .

Теорема 9. Минимальная форма максимум на единицу длиннее обычной записи, но содержит не более $n/2$ ненулевых цифр. При смене знака у числа меняются знаки у всех цифр его минимальной формы.

Доказательство. Двоичную запись числа a можно представить в виде одного или нескольких блоков

$$1 \dots 101 \dots 101 \dots 10 \dots 01 \dots 1,$$

не содержащих соседних нулей и отделяющихся друг от друга блоками из нескольких нулей. Число $3a$ получается сложением чисел $2a$ и a , которое можно представить в виде

$$\begin{array}{r} \dots 0011 \dots 1011 \dots 1011 \dots 101 \dots 1011 \dots 1100 \dots \\ + \dots 001 \dots 1101 \dots 1101 \dots 110 \dots 1101 \dots 1110 \dots \\ \hline \dots 101 \dots 1001 \dots 1001 \dots 100 \dots 1001 \dots 101 \dots \end{array}$$

Если какой-то из единичных блоков состоит из двух единиц, то соответствующая часть записи суммирования имеет вид

$$\begin{array}{r} 01101 \\ + 10110 \\ \hline 00100 \end{array}$$

(так как из предыдущего разряда идет перенос и в старшие разряды уходит перенос) или вид

$$\begin{array}{r} 1100 \dots \\ + 0110 \dots \\ \hline 001 \dots \end{array}$$

(в следующий разряд идет перенос). Если же какой-то из единичных блоков состоит только из одной единицы, то соответствующая

часть записи суммирования имеет вид

$$\begin{array}{r} 0101 \\ + 1010 \\ \hline 0000 \end{array}$$

(так как из предыдущего разряда идет перенос и в старшие разряды уходит перенос) или вид

$$\begin{array}{r} 1101 \dots 010100 \dots \\ + 0110 \dots 101010 \dots \\ \hline 0011 \dots 11111 \dots \end{array}$$

или вид

$$\begin{array}{r} 0010101 \dots 011 \\ + 001010 \dots 1011 \\ \hline 100000 \dots 001 \end{array}$$

(здесь идет перенос в следующий разряд).

Таким образом, в числах a и $3a$ соответствующие блоки имеют вид

$$\begin{array}{r} 001010 \dots 1011 \dots 1101 \dots 1101 \dots 110 \dots 1101 \dots 1110 \dots 1010100 \dots \\ + 100000 \dots 0011 \dots 1001 \dots 1001 \dots 100 \dots 1001 \dots 1011 \dots 11111 \dots \end{array}$$

Значит, соответствующий блок, состоящий из цифр δ_i , имеет вид

$\dots 010-1 \dots 0-100 \dots 0-10 \dots 0-10 \dots 0-10 \dots 0-10 \dots 0-101 \dots 010 \dots$,

поэтому в этой записи нет соседних ненулевых цифр, значит, она минимальна. Остальные утверждения легко следуют из выше доказанных. \square

Одно из возможных применений указанной минимальной формы — уменьшение числа арифметических операций в задаче возведения данного числа в высокую степень за счет использования операции деления (задача рассматривается в разделе «Аддитивные цепочки и флаги с молоком»¹⁾). Мы уже приводили

¹⁾ Гашков С. Б. Занимательная компьютерная арифметика. М.: URSS, 2012.

конкретный пример такого применения, а сейчас сформулируем общую теорему. Хотя далее, как и раньше, мы будем говорить о вычислении на калькуляторе, на самом деле, конечно, все сказанное справедливо и для компьютерных вычислений. Далее для краткости вместо словосочетания «число арифметических операций алгоритма» будем писать «сложность алгоритма».

Обозначим число ненулевых цифр в записи числа n в виде минимальной формы через $\nu(n)$, а уменьшенную на единицу длину этой записи — через $\lambda(n)$. Мы используем те же обозначения, что и в разделе «Аддитивные цепочки и фляги с молоком», но заметим, что новая функция $\lambda(n)$ может быть на единицу больше старой, зато новая функция $\nu(n)$ не может быть больше старой, а часто меньше ее, иногда почти в два раза.

Теорема 10. При использовании калькулятора с одной ячейкой памяти сложность вычисления x^n равно $\nu(n) + \lambda(n) - 1$.

Доказательство. Используя полученную минимальную форму, запишем n в виде суммы

$$2^{\lambda(n)} \alpha_{\lambda(n)} + \dots + 2\alpha_1 + \alpha_0, \quad \alpha_i = 0, \pm 1,$$

содержащей $\nu(n)$ ненулевых слагаемых. Далее, как и в обычном бинарном методе возведения в степень, используем аналог схемы Горнера, а цифрам минус единица сопоставляем операцию деления на основание степени. Полученное обобщение бинарного метода использует не более $\lambda(n)$ возведений в квадрат и $\nu(n) - 1$ умножений и делений. Верхняя оценка доказана.

Аналогично обычному бинарному методу можно доказать соотношения

$$\lambda(2n) + \nu(2n) = \lambda(n) + \nu(n) + 1, \quad \nu(n \pm 1) \leq \nu(n) + 1.$$

Из последнего неравенства следует, что

$$\lambda(n \pm 1) + \nu(n \pm 1) \leq \lambda(n) + \nu(n) + 1.$$

Действительно, случай $n \pm 1 = n + 1$ рассматривается аналогично обычному бинарному методу, а в случае $n \pm 1 = n - 1$ очевидно $\lambda(n - 1) \leq \lambda(n)$, и из неравенства $\nu(n - 1) \leq \nu(n) + 1$ следует нужная нам оценка. \square

Используя доказанное неравенство, можно, аналогично обычному бинарному методу (см. раздел «Аддитивные цепочки и флаги с молоком»), в случае, когда содержимое ячейки памяти никогда не обновляется, получить аналогичную нижнюю оценку сложности возведения в степень $\nu(n) + \lambda(n) - 1$. Читателю предоставляется возможность самому убедиться в этом во всех деталях.

Недостатком двоичной системы при ее ручном использовании является то, что из-за увеличения длины записи по сравнению с десятичной системой соответственно возрастает и сложность умножения.

Использование минимальной формы позволяет уменьшить сложность ручного умножения двоичных чисел. Опишем алгоритм умножения, предложенный в начале 50-х гг. прошлого века Э. Д. Бутом²⁾.

Для этого данные n - и m -разрядные двоичные числа преобразуем в их минимальные формы и заметим, что эти формы содержат не более $n + 1$ и $m + 1$ разрядов, причем из них не более $a = n/2 + 1$ и $b = m/2 + 1$ ненулевых разрядов соответственно. Сложность преобразования не превосходит $3n + 3m - 4$. Умножая минимальные формы с помощью школьного алгоритма, замечаем, что число нетривиальных умножений не превосходит ab , так как ненулевых строк не более b и в каждой из них нетривиальных умножений не более a . Отметим, что каждое нетривиальное умножение по существу не сложнее нетривиального умножения в обычной двоичной системе, и будем



Эндрю Бут

²⁾ Эндрю Дональд Бут (1918–2009) — британский физик, инженер, изобретатель памяти на магнитных барабанах для компьютеров.

считать, что оно выполняется с единичной сложностью, так же как и нетривиальное сложение (операция нетривиальна, если оба операнда не нули). Заметим также, что число нетривиальных сложений не превосходит $(b-1)(a+n-1)$, так как всего сложений различных строк требуется не более $b-1$, а каждое из них состоит не более чем из n переносов (переносы могут быть как 1, так и -1) и не более чем из $a-1$ нетривиальных сложений (в складываемых строчках имеется не более $a-1$ стоящих друг под другом ненулевых цифр). Поэтому сложность умножения не превосходит $(b-1)(a+n-1) + ab \leq mn + (m+n)/2 + 1$. Полученный результат содержит не более $n+m+2$ разрядов (так как он получается при сложении $m+1$ не более чем $(n+1)$ -разрядных чисел с соответствующими сдвигами). Его можно привести к обычной двоичной записи, сделав не более $n+m+2$ операции (заменяем блоки соседних цифр вида $10 \dots 0 - 1$ на соответствующие блоки вида $01 \dots 1$, блоки вида $1 - 1$ — на блоки 01 , блоки без отрицательных цифр оставляем без изменения). Значит, полная сложность операции умножения не превосходит $mn + (m+n)/2 + 1 + 3n + 3m - 4 + n + m + 2 = mn + 3,5(m+n)$.

Глава 6

Быстрое умножение многочленов

Осенью 1960 г. в МГУ на мехмате начал работать семинар по математическим вопросам кибернетики под руководством А. Н. Колмогорова, где им была сформулирована гипотеза $n^2 < \dots$ Я активно стал размышлять над ней и через неделю обнаружил, что метод, которым я надеялся получить нижнюю оценку величины $M(n)$, дает оценку вида $M(n) = O(n^{\log_2 3}) < \dots$ На следующем заседании семинара мой метод умножения был рассказан самим А. Н. Колмогоровым и на этом семинар прекратил свою работу $< \dots$ После этого началась бурная деятельность в прикладной математике, которая получила название «быстрые вычисления». Она продолжается до сих пор.

Анатолий Карацуба.

Сложность вычислений (Труды МИ РАН, 1995)

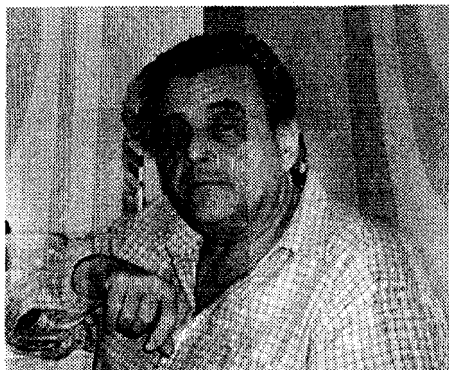
Мало кто знает, что относительно недавно были открыты гораздо более быстрые алгоритмы умножения и деления многозначных чисел и многочленов. Первый такой алгоритм придумал в 1962 г., отвечая на вопрос А. Н. Колмогорова, аспирант мехмата МГУ А. А. Карацуба ¹⁾.

Впоследствии, в 1963 г. студентом мехмата МГУ А. Л. Тоомом, а в 1970 г. немецкими



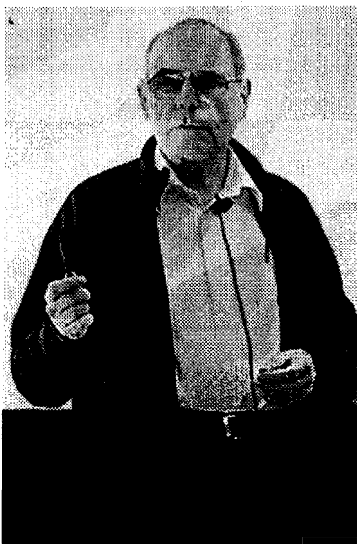
А. А. Карацуба

¹⁾ Анатолий Алексеевич Карацуба (1937–2008) — выдающийся специалист по теории чисел, профессор.



А. Л. Тоом

математиками Ф. Штрассеном и А. Шёнхаге были построены еще более быстрые алгоритмы для умножения чисел и многочленов.



Фолкер Штрассен

В 2007 г. американский математик Мартин Фюрер предложил для умножения чисел чуть более быстрый алгоритм, чем алгоритм Шёнхаге—Штрассена. Однако описание этих алгоритмов сильно выходит за рамки нашей книжки.

Идею метода Карацубы можно пояснить на следующем примере. Пусть перемножаются 8-значные числа

$$U = \overline{u_1 \dots u_8} \quad \text{и} \quad V = \overline{v_1 \dots v_8}.$$

Представим их как двузначные числа в 10^4 -значной системе счисления:

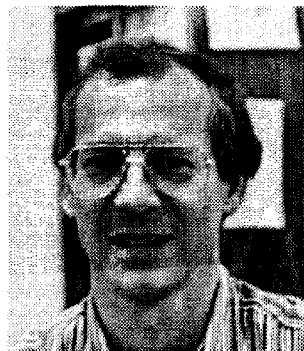
$$U = U_1 U_2, \quad V = V_1 V_2.$$

Тогда их произведение можно представить в следующем виде:

$$UV = U_1 V_1 10^8 + ((U_1 - U_2)(V_2 - V_1) + U_1 V_1 + U_2 V_2) 10^4 + U_2 V_2.$$

Эта формула сводит умножение 8-значных чисел к трем операциям умножения и шести операциям сложения—вычитания

4-значных чисел (с учетом переносов в следующие разряды). Обычный способ требует четырех умножений и трех сложений—вычитаний, но так как три раза сложить 4-значные числа можно быстрее, чем один раз перемножить, то метод Карацубы уже 8-значные числа перемножает быстрее. В общем случае он требует для перемножения n -значных чисел по порядку не больше $n^{\log_2 3} < n^{1,585}$ операций над цифрами.



Мартин Фюрер

Далее мы рассмотрим вопрос о сложности умножения более подробно.

Обозначим $M(n)$ наименьшее количество операций сложения, вычитания и умножения (выполняемых над коэффициентами многочленов и промежуточными числовыми результатами), требующихся для перемножения двух многочленов степеней меньших n .

Лемма 6. Справедливо неравенство

$$M(n) \leq 2M(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor) + 4\lfloor n/2 \rfloor + 2n - 4.$$

Доказательство. Применим равенство

$$\begin{aligned} & (f_1 x^{\lceil n/2 \rceil} + f_0)(g_1 x^{\lfloor n/2 \rfloor} + g_0) = \\ & = f_1 g_1 x^{2\lfloor n/2 \rfloor} + ((f_1 + f_0)(g_1 + g_0) - f_1 g_1 - f_0 g_0) x^{\lfloor n/2 \rfloor} + f_0 g_0, \end{aligned}$$

где степени многочленов f_1 и g_1 меньше $\lceil n/2 \rceil$, а степени многочленов f_0 и g_0 меньше $\lfloor n/2 \rfloor$ и заметим, что для вычисления произведений $f_1 g_1$, $f_0 g_0$ требуется не более $M(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor)$ операций, для вычисления сумм $f_1 + f_0$, $g_1 + g_0$, $f_1 g_1 + f_0 g_0$ нужно не более $2\lfloor n/2 \rfloor + 2\lfloor n/2 \rfloor - 1$ операция (так как число операций равно наименьшему из количеств ненулевых коэффициентов у складываемых многочленов), для вычисления произведения

$(f_1 + f_0)(g_1 + g_0)$ используется не более $M(\lceil n/2 \rceil)$ операций, для вычисления разности

$$(f_1 + f_0)(g_1 + g_0) - f_1g_1 - f_0g_0$$

достаточна $n - 1$ операция, так как

$$(f_1 + f_0)(g_1 + g_0) - f_1g_1 - f_0g_0 = f_1g_0 + f_0g_1,$$

значит, степень этого многочлена равна $\lfloor n/2 \rfloor + \lfloor n/2 \rfloor - 2 = n - 2$, сложение многочленов f_0g_0 и $f_1g_1x^{2\lfloor n/2 \rfloor}$ выполняется «бесплатно», так как они не имеют подобных членов, причем в их сумме отсутствует член вида $x^{2\lfloor n/2 \rfloor - 1}$, поэтому для сложения многочленов

$$f_0g_0 + f_1g_1x^{2\lfloor n/2 \rfloor} \quad \text{и} \quad (f_1g_0 + f_0g_1)x^{\lfloor n/2 \rfloor}$$

достаточно $n - 2$ операции. В итоге требуется дополнительно $4\lfloor n/2 \rfloor + 2n - 4$ операции. \square

Оценку сложности метода Карацубы можно представить в следующем виде.

Теорема 11 (теорема Карацубы). Если n кратно 2^k , то справедливо неравенство

$$M(n) \leq 3^k \left(M\left(\frac{n}{2^k}\right) + \frac{8n}{2^k} - 2 \right) - 8n + 2,$$

а при любом n — неравенство

$$M(n) < \frac{35}{3} n^{\log_2 3}.$$

Доказательство. Пусть $2^k m = n$. Тогда неравенство

$$M(n) \leq 3^k (M(m) + 8m - 2) - 8n + 2$$

доказывается индукцией по k . База ($k = 1$) уже доказана выше. Шаг индукции обосновывается тем же неравенством.

Выберем k так, чтобы $2^k < n \leq 2^{k+1}$. Тогда если $3 \cdot 2^{k-1} < n$, то

$$\begin{aligned} M(n) &\leq M(2^{k+1}) < 3^{k-1}(M(4) + 30) \leq 3^{k-1} \cdot 55 < \\ &< 55 \cdot \left(\frac{n}{3}\right)^{\log_2 3} < \frac{35}{3} n^{\log_2 3}. \end{aligned}$$

Если же $n \leq 3 \cdot 2^{k-1}$, то

$$M(n) \leq M(3 \cdot 2^{k-1}) < 3^{k-1}(M(3) + 22) \leq 3^{k-1} \cdot 35 \leq \frac{35}{3} n^{\log_2 3}. \quad \square$$

Рассмотрим вопрос сложности умножения многочленов разных степеней.

Теорема 12. При $m \geq n$ справедливо неравенство

$$M(m, n) \leq \lceil m/n \rceil M(n) + m - m/n,$$

и при $m/n \rightarrow \infty$ в предположении, что $M(n)/n \rightarrow \infty$ при $n \rightarrow \infty$, справедливо асимптотическое неравенство

$$M(m, n) \lesssim \frac{mM(n)}{n}.$$

Доказательство. Представим многочлен степени меньшей m в виде

$$f_0 + f_1 x^n + \dots + f_k x^{nk},$$

где $k = \lceil m/n \rceil - 1$ и степени многочленов f_i меньше n . Пользуясь тождеством

$$(f_0 + f_1 x^n + \dots + f_k x^{nk})g = (f_0 g + f_1 g x^n + \dots + f_k g x^{nk})$$

и неравенством $m/n \geq \lceil m/n \rceil - 1$, получим оценку

$$\begin{aligned} M(m, n) &\leq \lceil m/n \rceil M(n) + (n-1)(\lceil m/n \rceil - 1) < \\ &< \lceil m/n \rceil M(n) + m - m/n < mM(n)/n + m + M(n) = \\ &= (mM(n)/n)(1 + n/M(n) + n/m). \end{aligned}$$

Используя вместе с этой теоремой любой быстрый способ умножения многочленов равной степени, получаем быстрый способ умножения многочленов разных степеней. Например, применяя метод Карацубы, получаем при $m \geq n$ оценку

$$M(m, n) \lesssim \frac{35}{3} mn^{\log_2(3/2)}.$$

Очевидно, она при больших n лучше, чем оценка сложности школьного алгоритма. \square

Задача 14. Проверьте, что умножение методом Карацубы многочленов степени 7 и выше требует меньшего числа операций, чем школьный метод. Проверьте, что умножение методом Карацубы многочленов степени 5 также требует меньшего числа операций, чем школьный метод. Если предположить, что умножение коэффициентов выполняется вдвое медленнее сложения—вычитания, то метод Карацубы всегда быстрее школьного метода.

Обозначим число умножений в методе Карацубы (которое можно назвать мультипликативной сложностью этого метода) через K_n , где $(n - 1)$ — степень умножаемых многочленов.

Задача 15. Докажите, что

$$K_{2n} = 3K_n, \quad K_{2n+1} = 2K_{n+1} + K_n, \quad K_1 = 1$$

и выведите отсюда, что

$$K_n \leq C \cdot n^{\log_2 3}.$$

Задача 16 (Д. Кнут). Пусть двоичное разложение числа n

$$n = 2^{e_1} + \dots + 2^{e_t}, \quad e_1 > \dots > e_t \geq 0.$$

Докажите, что

$$K_n = 3^{e_1} + \sum_{l=2}^t 3^{e_l} 2^{e_1 - e_l + 3}.$$

Задача 17. Докажите, что сложность возведения в квадрат удовлетворяет рекуррентному неравенству

$$Q(n) \leq 2Q(\lceil n/2 \rceil) + Q(\lfloor n/2 \rfloor) + 3\lfloor n/2 \rfloor + 2n - 4.$$

Задача 18. Докажите, что при использовании метода Карацубы для возведения в квадрат, сложность этой операции асимптотически равна $7/8$ сложности умножения произвольных многочленов той же степени методом Карацубы.

Задача 19. Две правильные алгебраические дроби степеней меньших n и m при $m \geq n$ можно сложить со сложностью

$$3\lceil m/n \rceil M(n) + O(m).$$

Задача 20. Оценку леммы 6 можно усилить до следующей:

$$M(2n) \leq 3M(n) + 7n - 3.$$

Указание. Достаточно заметить, что при подсчете числа сложений некоторые операции мы посчитали дважды (это не сразу бросается в глаза, но если читатель попробует на конкретном примере, скажем, при $n = 3$, аккуратно выписать все выполняемые операции сложения, то заметит, что некоторые операции дублируются; результаты таких операции можно запомнить и потом использовать, когда понадобится, не вычисляя вновь).

Задача 21. Используя задачу 20, оценку теоремы 11 можно уточнить следующим образом:

$$M(n) \leq 3^k \left(M\left(\frac{n}{2^k}\right) + \frac{7n}{2^k} - \frac{3}{2} \right) - 7n + \frac{3}{2},$$

$$M(n) < \left(\frac{65}{6}\right) n^{\log_2 3}.$$

Глава 7

Быстрое умножение чисел

Я был в это время студентом старшего курса в Гарварде и, вдохновленный вопросом Кобхэма «Сложнее ли умножение, чем сложение», наивно пытался доказать, что умножение требует $\Omega(n^2)$ шагов на многоленточной машине Тьюринга. Статья Тоома сильно удивила меня <...> Я отметил в своей диссертации, что метод Тоома может быть приспособлен к многоленточным машинам Тьюринга с тем, чтобы выполнить умножение за $n^{1+\epsilon}$ шагов, то, что, я уверен, Тоома не удивило бы.

Стивен А. Кук. Лекция при получении премии Тьюринга 1982 г.

Перейдем теперь к умножению чисел. Обозначим $M(n)$ наименьшее количество операций сложения, вычитания и умножения, выполняемых над числами, меньшими b , требующихся для перемножения двух n -значных чисел, записанных в позиционной системе счисления по основанию b .

Метод умножения почти такой же, как и для многочленов. Объясним, как внести необходимые изменения в лемму из предыдущего раздела.

Лемма 7. Справедливы неравенства

$$M(2n) \leq 3M(n) + 19n,$$

$$M(2n + 1) \leq 2M(n + 1) + M(n) + 17n + 10.$$

Доказательство. Применим тождества

$$\begin{aligned} & (f_1 b^{\lceil n/2 \rceil} + f_0)(g_1 b^{\lceil n/2 \rceil} + g_0) = \\ & = f_1 g_1 b^{2\lceil n/2 \rceil} + (f_1 g_1 + f_0 g_0 - (f_1 - f_0)(g_1 - g_0))b^{\lceil n/2 \rceil} + f_0 g_0, \end{aligned}$$

где числа f_1 и g_1 — $\lfloor n/2 \rfloor$ -разрядные, а числа f_0 и g_0 соответственно $\lceil n/2 \rceil$ -разрядные, и заметим, что для вычисления произведений $f_1 g_1$ и $f_0 g_0$ требуется $M(\lfloor n/2 \rfloor) + M(\lceil n/2 \rceil)$ операций, для вычисления разностей и суммы

$$f_0 - f_1, \quad g_0 - g_1, \quad f_1 g_1 + f_0 g_0$$

требуется не более

$$\begin{aligned} & n(1 + \lfloor n/2 \rfloor - \lceil n/2 \rceil) + 2(\lfloor n/2 \rfloor + \lceil n/2 \rceil - 1) + 2(\lfloor n/2 \rfloor + \lceil n/2 \rceil) - 1 = \\ & = 4n - 3 + n(1 + \lfloor n/2 \rfloor - \lceil n/2 \rceil) \end{aligned}$$

операций, так как числа $f_1 g_1$ и $f_0 g_0$ имеют не более чем $2\lfloor n/2 \rfloor$ и $2\lceil n/2 \rceil$ разрядов соответственно, а в случае четного n нужно еще $2\lfloor n/2 \rfloor = n$ операций для предварительного сравнения чисел (чтобы не вычитать из меньшего большее). Заметим далее, что для вычисления произведения $(f_1 - f_0)(g_1 - g_0)$ требуется не более $M(\lfloor n/2 \rfloor) + 1$ операция (одна операция для вычисления знака у произведения), для вычисления разности

$$f_1 g_1 + f_0 g_0 - (f_1 - f_0)(g_1 - g_0) = f_1 g_0 + f_0 g_1$$

требуется не более

$$2\lceil n/2 \rceil + 1 + 2\lfloor n/2 \rfloor - 1 = 4\lfloor n/2 \rfloor$$

операций, сложение чисел $f_0 g_0$ и $f_1 g_1 b^{2\lceil n/2 \rceil}$ осуществляется «бесплатно» (записи этих чисел просто объединяются в одну запись), а для сложения чисел

$$f_1 g_1 b^{2\lceil n/2 \rceil} + f_0 g_0 \quad \text{и} \quad (f_1 g_0 + f_0 g_1) b^{\lceil n/2 \rceil}$$

требуется не более

$$2n - \lfloor n/2 \rfloor + n + 1 - 1 = 2n + \lfloor n/2 \rfloor$$

операций (так как число $f_1g_0 + f_0g_1$ имеет не более $n + 1$ разряда, а младшие $\lfloor n/2 \rfloor$ разрядов числа f_0g_0 не участвуют в операциях). В итоге требуется дополнительно

$$\begin{aligned} 4n - 3 + n(1 + \lfloor n/2 \rfloor - \lceil n/2 \rceil) + 1 + 4\lceil n/2 \rceil + 2n + \lfloor n/2 \rfloor &= \\ = 7n + 3\lfloor n/2 \rfloor + n(1 + \lfloor n/2 \rfloor - \lceil n/2 \rceil) - 2 \end{aligned}$$

операции. □

Отметим, что в среднем сравнение двух n -разрядных b -ичных чисел осуществляется гораздо быстрее, чем за n операций. Действительно, ровно k операций сравнения цифр понадобится для сравнения чисел лишь с вероятностью не выше b^{-k+1} (в предположении равномерной распределенности и независимости цифр в числах), значит, математическое ожидание операций не превосходит

$$\sum_{k=1}^n kb^{-k+1} = \sum_{k=0}^{n-1} \frac{b^{-k} - b^{-n}}{1 - b} < \frac{1 - b^{-1}}{(1 - b)^2} = 1 + O\left(\frac{1}{b}\right).$$

При сложении m -разрядного и n -разрядного чисел при $m > n$ в среднем также требуется не $m+n-1$ операция, а $2n + O(1/b)$ операций. Поэтому в среднем оценку леммы можно заменить на такую

$$M(n) \leq 2M(\lceil n/2 \rceil) + M(\lfloor n/2 \rfloor) + 8n + 2.$$

Остальные детали предоставляем додумать читателю.

Задача 22 (Д. Кнут). Докажите, что K_n нечетно и мультипликативная сложность умножения n -разрядного числа на $(n+1)$ -разрядное равна $(K_n + K_{n+1})/2$.

Схемная реализация метода Карацубы для умножения двоичных чисел

Пусть u и v — два $2n$ -разрядных двоичных натуральных числа:

$$u = (u_{2n-1} \dots u_1 u_0)_2, \quad v = (v_{2n-1} \dots v_1 v_0)_2,$$

$$u_i, v_j \in \{0, 1\}, \quad i, j = 0, 1, \dots, 2n-1;$$

$$u = \sum_{j=0}^{2n-1} u_j 2^j, \quad v = \sum_{i=0}^{2n-1} v_i 2^i.$$

Введем обозначения

$$\begin{aligned} U_1 &= (u_{2n-1} \dots u_n)_2, & V_1 &= (v_{2n-1} \dots v_n)_2, \\ U_0 &= (u_{n-1} \dots u_0)_2, & V_0 &= (v_{n-1} \dots v_0)_2, \\ U_2 &= (U_1 - U_0), & V_2 &= (V_1 - V_0). \end{aligned}$$

Тогда

$$\begin{aligned} u &= 2^n U_1 + U_0, & v &= 2^n V_1 + V_0, \\ uv &= 2^{2n} U_1 V_1 + 2^n (U_1 V_1 + U_2 V_2 + U_0 V_0) + U_0 V_0. \end{aligned} \quad (*)$$

Рассмотрим реализацию метода Карацубы схемами из функциональных элементов (СФЭ) в базисе $\{\neg, \&, \vee, \oplus, 0, 1\}$. Обозначим $M(n)$ число элементов (т. е. сложность) СФЭ, реализующей умножение двух n -разрядных двоичных чисел. Из (*) следует, что

$$M(2n) \leq 3M(n) + c_1 n + c_2.$$

Оценим сверху константы c_i . Для этого понадобятся экономные реализации некоторых вспомогательных схем.

Инвертор двоичному числу $x = (x_n \dots x_1)_2$ ставит в соответствие двоичное число $\bar{x} = (\bar{x}_n \dots \bar{x}_1)_2 = e_n - x$, где $e_n = 2^n - 1 = \underbrace{(1 \dots 1)}_{n \text{ раз}}_2$. Сложность инвертора $L(I_n) = n$.

Схема D_n вычисляет модуль разности $|z - y|$ двух n -разрядных натуральных двоичных чисел z и y и знак разности $(z - y)$

$$\text{sign}(z - y) := \begin{cases} 0, & z \geq y, \\ 1, & z < y. \end{cases}$$

Пусть $z \geq y$, $z = x + y$. Так как

$$z_i = x_i \oplus y_i \oplus q_i,$$

то

$$\begin{aligned} x_i &= z_i \oplus y_i \oplus q_i; \\ q_{i+1} &= x_i y_i \oplus (x_i \oplus y_i) q_i = \\ &= (z_i \oplus y_i \oplus q_i) y_i \oplus (z_i \oplus q_i) q_i = (z_i \oplus q_i)(y_i \oplus q_i) \oplus y_i. \end{aligned}$$

Поэтому,

$$\begin{cases} q_1 = 0, \\ x_i = (z_i \oplus q_i) \oplus y_i, \\ q_{i+1} = (z_i \oplus q_i)(y_i \oplus q_i) \oplus y_i, \end{cases} \quad \text{где } i = 1, \dots, n.$$

Обозначим через R_i схему, изображенную на рис. 1.

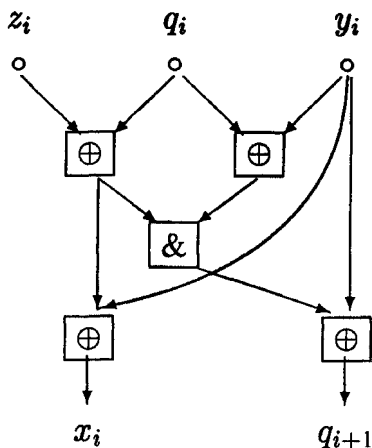


Рис. 1. Блок переносов

Схема S_n , вычисляющая x , $x = z - y$, получается путем последовательного соединения блоков R_i , $i = 1, \dots, n$, как показано на рис. 2.

Блок R_1 осуществляет преобразование $x_1 = z_1 + y_1 \pmod{2}$, $q_2 = z_1 y_1 + y_1 \pmod{2}$. Таким образом, сложность схемы $L(S_n) \leq 5n - 2$. Очевидно, что $s = q_{n+1} = \text{sign}(z - y) = 0$, если $(x_n \dots x_1)_2 = x = |x| \geq 0$.

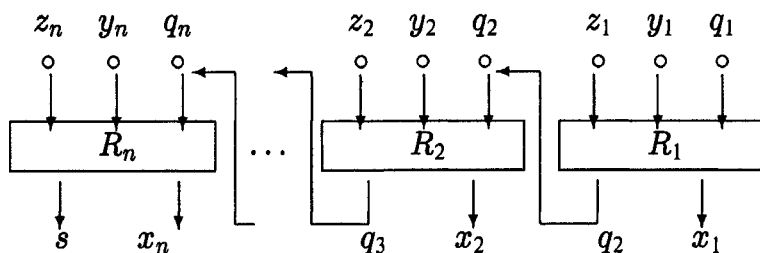


Рис. 2. Схема вычитания с вычислением знака разности

В случае $z < y$

$$\begin{aligned}
 s &= \text{sign}(z - y) = 1, \quad x = (x_n \dots x_1)_2 - 2^n < 0, \\
 |x| &= -x = 2^n - (x_n \dots x_1)_2 = ((2^n - 1) - (x_n \dots x_1)_2) + 1 = \\
 &= (e_n - (x_n \dots x_1)_2) + 1 = (\bar{x}_n \dots \bar{x}_1)_2 + 1 > 0.
 \end{aligned}$$

Таким образом, всегда

$$s = \text{sign}(z - y), \quad |z - y| = (x_n \oplus s, \dots, x_1 \oplus s)_2 + s.$$

Схема D_n выглядит следующим образом (рис. 3).

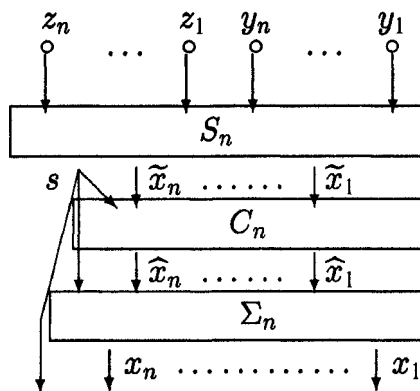


Рис. 3. Схема для вычисления абсолютной величины разности

Блок C_n осуществляет преобразование $\hat{x}_1 = \tilde{x}_1 \oplus s, \dots, \hat{x}_n = \tilde{x}_n \oplus s$, где $s = \text{sign}(z - y)$, со сложностью n .

Блок Σ_n выполняет преобразование

$$(x_n, \dots, x_1)_2 = (\hat{x}_n, \dots, \hat{x}_1)_2 + s$$

со сложностью $2n - 1$, как это следует из предыдущих формул, учитывая, что $q_{n+1} = 0$.

Ясно, что

$$x = (x_n \dots x_1)_2 = |z - y|, \quad \text{sign}(z - y) = s.$$

Сложность построенной схемы

$$L(D_n) \leq L(S_n) + L(C_n) + L(\Sigma_n) \leq (5n - 2) + n + (2n - 1) = 8n - 3.$$

Таким образом, сложность вычисления

$$U_2 = (U_1 - U_0), \quad V_2 = (V_0 - V_1)$$

и знака

$$\text{sign}(U_2 V_2) = \text{sign}(U_2) \oplus \text{sign}(V_2)$$

удовлетворяет неравенству

$$L_0 \leq 2(8n - 3) + 1 = 16n - 5.$$

Пусть $U_1 V_1$, $U_0 V_0$ и $|U_2 \cdot V_2|$ уже вычислены со сложностью

$$L(U_1 V_1, U_0 V_0, |U_2 \cdot V_2|) \leq 3M(n) + 16n - 5$$

и представляют собой $2n$ -разрядные двоичные числа.

Далее требуется вычислить $t = x + y$, где x , $x + y$ — $(2n + 1)$ -разрядные, а y — $2n$ -разрядное двоичные натуральные числа, причем $x \geq 0$, а y может быть как неотрицательным, так и меньше нуля. Это делается с помощью следующей схемы T_n (рис. 4).

Пояснения к схеме:

$$s = \text{sign}(y), \quad x = (x_{2n+1} \dots x_1)_2, \quad y = (y_{2n} \dots y_1)_2.$$

Блок C_n^1 вычисляет

$$\tilde{x}_1 = x_1 \oplus s, \quad \dots, \quad \tilde{x}_{2n+1} = x_{2n+1} \oplus s.$$

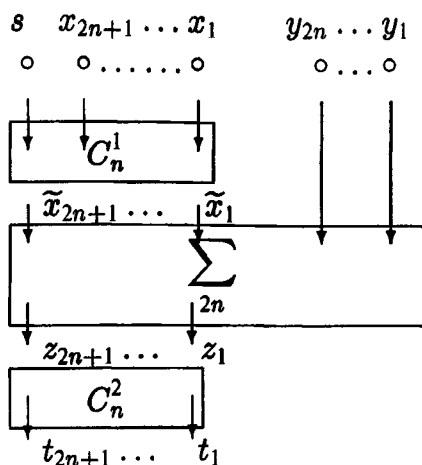


Рис. 4. Схема для сложения—вычитания

Блок Σ_{2n} складывает столбиком $(\tilde{x}_{2n+1}, \dots, \tilde{x}_1)_2$ и $(y_{2n}, \dots, y_1)_2$.
 Блок C_n^2 вычисляет $t = (t_{2n+1} \dots t_1)_2$, где

$$t_1 = z_1 \oplus s, \quad \dots, \quad t_{2n+1} = z_{2n+1} \oplus s.$$

Таким образом, если $s = 0$, то вычисляется $t = x + y$, если $s = 1$, то вычисляется

$$t = \overline{\overline{x} + y} = e_n - ((e_n - x) - y) = x - y.$$

Сложность схемы T_n оценивается как

$$\begin{aligned} L(T_n) &\leq L(C_n^1) + L(\Sigma_{2n}) + L(C_n^2) \leq \\ &\leq (2n + 1) + (5 \cdot 2n - 3 + 1) + (2n + 1) = 14n. \end{aligned}$$

Оценим сложность всей схемы умножения. Вспомним, что

$$\begin{aligned} U_1 V_1 + U_2 V_2 + U_0 V_0 &= U_1 V_1 + (U_1 - U_0)(V_0 - V_1) + U_0 V_0 = \\ &= U_1 V_0 + U_0 V_1 \geq 0. \end{aligned}$$

Сумма $U_1 V_1 + U_0 V_0$ вычисляется со сложностью, не меньшей

$$5 \cdot 2n - 3 = 10n - 3,$$

и получается $(2n + 1)$ -разрядное число. Далее вычисляется

$$(U_1V_1 + U_0V_0) + U_2V_2$$

с помощью схемы T_n .

Таким образом,

$$W_0 = 2^n(U_1V_1 + U_2V_2 + U_0V_0)$$

вычисляется со сложностью

$$L(W_0) \leq (10n - 3) + 14n = 24n - 3$$

в предположении, что соответствующие произведения уже вычислены (очевидно, умножение на 2^n осуществляется просто сдвигом компонент двоичного вектора и без каких-либо вычислений). Остается вычислить $W_0 + W_1$, где

$$W_1 = 2^{2n}U_1V_1 + U_0V_0.$$

Заметим, что двоичные цифры числа W_1 получаются просто объединением двоичных цифр чисел U_1V_1 и U_0V_0 (которые уже вычислены), и эта операция не требует каких-либо вычислений. Число W_1 имеет $4n$ разрядов, а число W_0 — $3n + 1$, причем в W_0 младшие n разрядов — нули. С учетом этого, сложность соответствующего сумматора S^1 будет

$$L(S^1) \leq 5(2n + 1) - 3 + 2(n - 1) - 1 = 12n - 1.$$

Окончательно, имеем следующую рекуррентную оценку сложности:

$$\begin{aligned} M(2n) &\leq L(U_1V_1, U_0V_0, |U_2 \cdot V_2|) + L(W_0) + L(S^1) \leq \\ &\leq 3M(n) + (16n - 5) + (24n - 3) + (12n - 1) = \\ &= 3M(n) + 52n - 9, \end{aligned}$$

т. е.

$$M(2n) \leq 3M(n) + 52n - 9.$$

Покажем, как можно несколько улучшить полученную выше рекуррентную оценку. Рассмотрим число

$$W_1 = 2^{2n}U_1V_1 + U_0V_0.$$

Его младшие $2n$ разрядов в точности совпадают с числом U_0V_0 , а старшие $2n$ разрядов в точности совпадают с числом U_1V_1 . Обозначим число, образованное старшими n разрядами числа U_1V_1 , через A_1 , а число, образованное младшими n разрядами, через A_0 , тогда $U_1V_1 = A_12^n + A_0$. Аналогично определим n -разрядные числа B_1 и B_0 , такие, что $U_0V_0 = B_12^n + B_0$.

Рассмотрим внимательнее сумму

$$\begin{aligned} W_1 + W_0 &= W_1 + (U_1V_1 + U_0V_0 + U_2V_2)2^n = \\ &= 2^{2n}U_1V_1 + U_0V_0 + (U_1V_1 + U_0V_0 + U_2V_2)2^n. \end{aligned}$$

Она равна

$$\begin{aligned} &2^{3n}A_1 + 2^{2n}A_0 + 2^nB_1 + B_0 + (2^nA_1 + A_0 + 2^nB_1 + B_0)2^n + U_2V_22^n = \\ &= 2^{3n}A_1 + 2^{2n}(A_0 + A_1 + B_1) + 2^n(B_1 + A_0 + B_0) + B_0 + U_2V_22^n = \\ &(2^{3n}A_1 + 2^{2n}A_1 + 2^nB_0 + B_0) + (2^n(A_0 + B_1) + (A_0 + B_1) + U_2V_2)2^n. \end{aligned}$$

Первое слагаемое в ней

$$2^{3n}A_1 + 2^{2n}A_1 + 2^nB_0 + B_0$$

составляется, как и число W_1 , без использования каких-либо операций. А второе слагаемое

$$2^n(A_0 + B_1) + (A_0 + B_1) + U_2V_2$$

вычисляется с меньшей сложностью, чем число W_0 . Действительно, $A_0 + B_1$ является, вообще говоря, $(n+1)$ -разрядным числом и вычисляется со сложностью $5n-3$. После этого число $2^n(A_0 + B_1) + (A_0 + B_1)$ вычисляется со сложностью $2n+1$, так как его вычисление сводится к прибавлению к $(n+1)$ -разрядному числу $A_0 + B_0$ одноразрядного числа, а перенос в $(n+2)$ -й разряд не возникает, так как $A_0 + B_1 < 2^{n+1} - 1$, и в результате

получается $(2n+1)$ -разрядное число $2^n(A_0+B_1)+(A_0+B_1)$. К этому числу далее прибавляется или вычитается $2n$ -разрядное число U_2V_2 точно так же, как вычислялось число $(U_1V_1+U_0V_0)+U_2V_2$ с помощью схемы T_n . Но число $U_1V_1+U_0V_0$ ранее вычислялось со сложностью $10n-3$, а используемое здесь вместо него число $2^n(A_0+B_1)+(A_0+B_1)$ было вычислено выше со сложностью $5n-3+2n+1=7n-2$. Поэтому общая сложность в рассматриваемом варианте схемы уменьшается на $3n-1$ и становится равной

$$M(2n) = 3M(n) + 49n - 8.$$

Применяя это неравенство при $n=8$, а для умножения 8-разрядных чисел используя школьный метод, видим, что полученная таким образом схема Карацубы для умножения 16-разрядных чисел имеет чуть меньшую сложность, чем схема, реализующая школьный алгоритм. Но при $n=7$ (т.е. для умножения 14-разрядных чисел) метод Карацубы дает схему большей сложности, чем школьный метод.

Получим в одном частном случае верхнюю оценку сложности схемы Карацубы в явном виде. Пусть n является степенью двойки. Тогда, используя данное выше рекуррентное неравенство, имеем

$$M(n) \leq 3M\left(\frac{n}{2}\right) + 49\frac{n}{2} - 8,$$

$$M\left(\frac{n}{2}\right) \leq 3M\left(\frac{n}{2^2}\right) + 49\frac{n}{2^2} - 8,$$

$$M(n) \leq 3^2 M\left(\frac{n}{2^2}\right) + 49n\left(\frac{1}{2} + \frac{3}{2^2}\right) - 8(1+3),$$

$$\dots\dots\dots M(n) \leq 3^k M\left(\frac{n}{2^k}\right) + 49n\left(\frac{1}{2} + \frac{3}{2^2} + \dots + \frac{3^{k-1}}{2^k}\right) - 8(1+3+\dots+3^{k-1}),$$

$$M(n) \leq 3^k M\left(\frac{n}{2^k}\right) + 49n\left(\left(\frac{3}{2}\right)^k - 1\right) - 4(3^k - 1).$$

Используя оценку

$$M(n/2^k) = M(8) = 6 \cdot 8^2 - 8 \cdot 8 = 320$$

школьного метода, отсюда выводим, что

$$M(n) \leq 26\frac{2}{9} \cdot n^{\log_2 3} - 49n + 4.$$

Например, для умножения 32-битных чисел схема имеет сложность 4808. Школьный метод дает схему сложности 5888. Для умножения 64-битных чисел имеем оценку 15 984. Школьный метод дает схему сложности 24 064.

Задача 23. Аналогичным образом попробуйте улучшить оценку сложности умножения многочленов методом Карацубы.

Задача 24. Попробуйте оценить, насколько быстрее возводить числа в квадрат в сравнении с общим умножением для метода Карацубы.

Глава 8

Деление многозначных чисел

Учитель: Сколько получится, если разделить 28 яблок на семерых?

Ученик: 13!

Учитель: Почему так много?

Ученик: Делим столбиком, 8 делим на 7, получаем 1, от 28 отнимаем 7, получаем 21, опять делим на 7, получаем 3, 1 и 3 составляют 13!

Учитель: А проверка?

Ученик: Умножаем 13 на 7, 1 на 7 равно 7, 3 на 7 равно 21, 21 плюс 7 будет 28!

Школьный анекдот

Деление многозначных чисел — самая трудная из арифметических операций. Это помнят все со школы. Дело в том, что при выполнении деления больших чисел столбиком возникает такая задача: как найти частное q от деления $(n + 1)$ -разрядного числа

$$u = \overline{u_0 \dots u_n}$$

на n -разрядное число

$$v = \overline{v_1 \dots v_n},$$

где $u_0 \leq v_1$. Это число равно $\lfloor u/v \rfloor$ и находится обычно подбором (поэтому школьный алгоритм деления алгоритмом на самом деле не является, так как требует определенного искусства от исполнителя). Следующий прием облегчает его нахождение. В его изложении следуем «Искусству программирования» Д. Кнута. Положим

$$\hat{q} = \min \left(\left\lfloor \frac{10u + u}{v} \right\rfloor, 9 \right).$$

Теорема 13. Справедливы неравенства $\widehat{q} \geq q$, а при $v_1 \geq 5$ к тому же $\widehat{q} - 2 \leq q$. Умножив u и v на $\lfloor 10/v_1 \rfloor$, можно, не изменяя u/v , добиться того, что $v \geq 5$. Неравенство $\widehat{q} - 2 \leq q$, вообще говоря, усилить нельзя.

Доказательство. Можно считать, что $\widehat{q} < 9$, тогда

$$\widehat{q} = \left\lfloor \frac{10u_0 + u_1}{v_1} \right\rfloor, \quad qv_0 \geq (10u_0 + u_1) - v_1 + 1,$$

значит

$$\begin{aligned} u - \widehat{q}v &\leq u - 10^{n-1}v_1\widehat{q} \leq \\ &\leq u_010^n + \dots + u_n - 10^n u_0 - 10^n u_1 + 10^{n-1}v_1 - 10^{n-1} < \\ &< 10^{n-1}v_1 \leq v. \end{aligned}$$

Отсюда следует, что $q \geq \widehat{q}$. Допустим, что $\widehat{q} \geq q + 3$. Тогда

$$\widehat{q} \leq \frac{10u_0 + u_1}{v_1} = \frac{10^n u_0 + 10^{n-1}u_1}{10^{n-1}v_1} \leq \frac{u}{10^{n-1}v_1} < \frac{u}{v - 10^n},$$

значит

$$3 \leq \widehat{q} - q < \frac{u}{v - 10^{n-1}} + 1 - \frac{u}{v} = 1 + \frac{(u/v)10^{n-1}}{v - 10^{n-1}},$$

поэтому

$$\frac{u}{v} \geq \frac{2(v - 10^{n-1})}{10^{n-1}} \geq 2(v_1 - 1),$$

откуда

$$6 \geq q - 3 \geq \widehat{q} = \left\lfloor \frac{u}{v} \right\rfloor \geq 2(v_1 - 1), \quad v_1 \leq 4,$$

противоречие. Для доказательства последнего утверждения заметим, что

$$v \left\lfloor \frac{10}{v+1} \right\rfloor < (v+1) \left\lfloor \frac{10}{v+1} \right\rfloor \leq 10,$$

и при $v \geq 5$ очевидно

$$v \left\lfloor \frac{10}{v+1} \right\rfloor \geq v \geq 5.$$

Если же $1 \leq v \leq 4$, то

$$v \left\lfloor \frac{10}{v+1} \right\rfloor > v \left(\frac{10}{v+1} - 1 \right) \geq 4,$$

так как

$$v \left(\frac{10}{v+1} - 1 \right) - 4 = \frac{v(9-v)}{v+1} \geq 0.$$

Пример, когда неравенство $q - 2 \leq \hat{q}$ обращается в равенство в случае десятичной системы, таков: $u = 4100$, $v = 588$. \square

Теорема 14.

а) Если $u_0 \geq v_1$, то $q = 9$ или 8 .

б) $q \geq \left\lfloor \frac{10u_0 + u_1}{v_1 + 1} \right\rfloor$.

в) Если при $\hat{r} = 10u_0 + u_1 - \hat{q}v_1$ справедливо неравенство $v_2\hat{q} > 10\hat{r} + u_2$, то $q = \hat{q} - 1$ или $q = \hat{q} - 2$, а если $v_1\hat{q} \leq \leq 10\hat{r} + u_2$, то $q = \hat{q}$ или $q = \hat{q} - 1$.

г) Если $v_1 \geq 5$, $v_2\hat{q} \leq 10\hat{r} + u_2$, но $q \neq \hat{q}$, то $u - qv \geq 41v/50$ (т. е. с вероятностью 0,82 все же $q = \hat{q}$).

Доказательство.

а) $\frac{u}{v} > \frac{u_0 10^n}{(v_1 + 1) 10^{n-1}} \geq 10 \left(1 - \frac{1}{v_1 + 1} \right) > 10 \left(1 - \frac{1}{5} \right) = 8.$

б) $\frac{10u_0 + u_1}{v_1 + 1} \leq \frac{u}{10(v_1 + 1)} < \frac{u}{v}.$

в) В первом случае

$$\begin{aligned} u - \hat{q}v &\leq u - \hat{q}v_1 10^{n-1} - \hat{q}v_2 10^{n-2} = \\ &= u_2 10^{n-2} + \dots + u_n + \hat{r} 10 - \hat{q}v_2 10^{n-2} < \\ &< 10^{n-2}(u_2 + 1 + 10r_2 - \hat{q}v_2) \leq 0, \end{aligned}$$

и так как $u - qv \geq 0$, то $\hat{q} > q$.

Во втором случае, предположив, что $\widehat{q} - 2 \geq q$, заметим, что тогда

$$\begin{aligned} u &< (\widehat{q} - 1)v < \widehat{q}(v_1 10^{n-1} + (v_2 + 1)10^{n-2}) - v < \widehat{q}(v_1 10^{n-1} + v_2 10^{n-2}) + \\ &+ 10^{n-1} - v \leq \widehat{q}v 10^{n-1} + (10\widehat{r} + u_2)10^{n-2} + 10^{n-1} - v = u_0 10^n + \\ &+ u_1 10^{n-1} + u_2 10^{n-2} + 10^{n-1} - v \leq u_0 10^n + u_1 10^{n-1} + u_2 10^{n-2} \leq u. \end{aligned}$$

Противоречие.

г) Из неравенства $v_2 \widehat{q} \leq 10\widehat{r} + u_2$ следует

$$\widehat{q} \leq \frac{100u_0 + 10u_1 + u_2}{10v_1 + v_2} \leq \frac{u}{10^{n-2}(10v_1 + v_2)}.$$

Отсюда $u - qv = v(1 - a)$, где

$$\begin{aligned} 0 < a = 1 + q - \frac{u}{v} &\leq \widehat{q} - \frac{u}{v} \leq u \left(\frac{1}{10^{n-2}(10v_1 + v_2)} - \frac{1}{v} \right) = \\ &= \frac{u(10^{n-3}v_3 + \dots + v_n)}{10^{n-2}(10v_1 + v_2)v} < \frac{u}{10v_1 v} \leq \frac{\widehat{q}}{10v_1} \leq \frac{9}{10v_1} \leq \frac{9}{50}. \quad \square \end{aligned}$$

Доказанные теоремы справедливы для позиционной системы по любому натуральному основанию b (при $b = 2$ все они тривиальны). В формулировках и доказательствах надо лишь заменить 10 на b , число 5 — на $\lfloor b/2 \rfloor$, а число 0,82 — на $(1 - 2/b)$.

Прежде чем заняться оценками величины $D(n, m)$ — сложности деления с остатком n -значного числа на m -значное (очевидно, при $n < m$ она равна нулю), добавим к использованным ранее операциям над цифрами новую операцию, осуществляющую деление двузначного числа на однозначное с остатком: $\delta(u_0, u_1, v) = (\delta_3, \delta_2, \delta_1)$, где

$$\begin{aligned} \delta_3 &= \delta_3(u_0, u_1, v), \quad \delta_2 = \delta_2(u_0, u_1, v), \quad q = \delta_3 b + \delta_2 = \left\lfloor \frac{u}{v} \right\rfloor, \\ u &= u_0 b + u_1, \quad \delta_1 = \delta_1(u_0, u_1, v) = u - q \left\lfloor \frac{u}{v} \right\rfloor. \end{aligned}$$

Всегда далее при применении этой операции будет $\delta_3 = 0$.

Лемма 8. Справедливо неравенство $D(n, 1) \leq n$.

Доказательство. Деление n -значного числа $\overline{u_1 \dots u_n}$ на однозначное v осуществляется следующим простым алгоритмом. Полагаем $r = 0$, $j = 1$, вычисляем

$$w_j = \left\lfloor \frac{rb + u_j}{v} \right\rfloor = \delta_2(r, u_j, v),$$

заменяем r на

$$rb + u_j \bmod v = \delta_1(r, u_j, v),$$

увеличиваем j на 1 и повторяем все, пока $j \leq n$. Так как всегда $0 \leq r < v_j$, то

$$0 \leq \left\lfloor \frac{(b + u_j)}{v} \right\rfloor < b,$$

значит $\delta_3(r, u_j, v) = 0$. Частным j будет число $\overline{w_1 \dots w_n}$, а остатком — r . \square

Легко получить оценку $D(n, m) = O(nm)$. Но для получения по возможности меньших мультипликативных констант в этой оценке можно использовать следующие далее леммы. Обозначим далее для краткости $D_0(n, m)$ сложность вычисления только частного (без остатка).

Лемма 9. Пусть

$$u = \overline{u_1 \dots u_m} \geq \overline{v_1 \dots v_m} = v, \quad v_1 \geq \lfloor b/2 \rfloor.$$

Тогда если $u_1 < 2v_1$, то $\lfloor u/v \rfloor = 1$. Равенство $\lfloor u/v \rfloor = 2$ возможно, лишь когда $u_1 \geq 2v_1$, т.е. при нечетном b и $u_1 = b - 1$, $v_1 = (b - 1)/2$. Сложность выполнения деления u на v с остатком, как правило, равна $2m + 1$ и лишь с малой вероятностью равна $6m$.

Доказательство. Частное $w = u/v$ заключено в пределах

$$\frac{u}{v+1} < w < \frac{u+1}{v},$$

значит при $u_1 < 2v_1$ неполное частное $[w] = 1$. Так как $v_1 \geq [b/2]$, то $u < b \leq 3[b/2] \leq 3v$, значит $[w] \leq 2$. Если $u_1 \geq 2v_1$, то $u_1 \geq b$ при четном b , что невозможно, либо $u_1 \geq b-1$, значит $u_1 = b-1 \geq 2v_1 \geq b-1$. Если хотя бы одно из равенств $u = b-1$, $v = (b-1)/2$ неверно, то деление сводится к вычитанию, а если оба верны, то к вычислению $u - 2v$ и прибавлению v , если разность будет отрицательной. \square

Лемма 10. Справедливо неравенство

$$D(n, m) \leq 7nm + 8n + 10m + 7 - 7m^2.$$

Доказательство. Опишем алгоритм деления числа $u = \overline{u_1 \dots u_n}$ на $v = \overline{v_1 \dots v_m}$ и оценим его сложность. Одновременно будем описывать схему (неветвящуюся программу) для деления чисел. Для применения оценок теоремы 13, нужно выполнение неравенства $v_1 \geq [b/2]$. С определенной точки зрения [5] вероятность этого события вовсе не $1/2$, а $\log_b 2$. Согласно той же теореме в случае невыполнения этого неравенства умножаем u и v на $d = [b/(v_1 + 1)]$, на что требуется, согласно лемме 2, не более $3(m+n) - 4$ операций. Вычисление d (вне зависимости от условия $v_1 \geq [b/2]$) можно осуществить со сложностью $S = 6$ с помощью следующей схемы

$$s = v_1 \dot{-} \delta_2(1, 0, 2), \quad d = \max(\delta_2(1, 0, v + s)s, 1),$$

где $x \dot{-} y$ означает знак разности чисел x и y , вычисляемый при их вычитании, т.е. $x \dot{-} y = 1$, если $x < y$, и $x \dot{-} y = 0$ в противном случае. При ручном вычислении сложность здесь равна 4, так как 2 операции очень легкие и их сложностью можно пренебречь.

В случае

$$\overline{u_1 \dots u_m} \geq \overline{v_1 \dots v_m}$$

(это заведомо так при $u_1 > v_1$ и иногда бывает при $u_1 = v_1$) делим $\overline{u_1 \dots u_m}$ на $\overline{v_1 \dots v_m}$ с остатком, записываем неполное частное в качестве первой цифры результата и заменяем запись

$\overline{u_1 \dots u_m}$ на полученную запись остатка. Сложность выполнения этой процедуры, согласно лемме 9, не превосходит $6m$. В случае $\overline{u_1 \dots u_m} < \overline{v_1 \dots v_m}$ выполняем деление $\overline{u_1 \dots u_{m+1}}$ на $\overline{v_1 \dots v_m}$ с остатком, но деление здесь выполняется сложнее. В случае $u_1 = v_1$ полагаем $\hat{q} = b - 1$, а в случае $u_1 < v_1$ полагаем

$$\hat{q} = \left\lfloor \frac{u_1 b + u_2}{v_1} \right\rfloor = \delta_2(u_1, u_2, v_1).$$

Проверяем выполнение неравенства (согласно теореме 14 в случае $u_1 = v_1$ эту проверку можно не делать)

$$v_2 \hat{q} > (u_1 b + u_2 - \hat{q} v_1) b + u_3. \quad (*)$$

Если оно выполнено, то уменьшаем q на 1 и повторяем проверку. Вычисление \hat{q} можно осуществить со сложностью $S = 12$ с помощью следующей схемы

$$s = u_1 \dot{-} v_1, \quad s = (\delta_2(u_1, u_2, v_1) b + u_3) \dot{-} v_2 \delta_1(u_1, u_2, v_1),$$

$$\hat{q} = (\delta_2(u_1, u_2, v_1) - s_1) s + (b - 1)(1 - s).$$

При ручном вычислении сложность здесь равна 7, так как 5 операций тривиальные и их сложностью можно пренебречь. Если при второй проверке неравенство (*) выполняется, то согласно теоремам 13 и 14 истинное значение $q = \lfloor u_1 \dots u_{m+1} / v \rfloor$ равно или \hat{q} , или $\hat{q} - 1$, причем вероятность последнего случая равна примерно $2/b$ при больших b . Далее вычисляем

$$\overline{u_1 \dots u_{m+1}} - v \hat{q}$$

с помощью леммы 8, и если результат только второй проверки будет отрицательным, то к получившемуся числу

$$b^{m+2} + \overline{u_1 \dots u_{m+1}} - v \hat{q}$$

(вместо отрицательного числа $\overline{u_1 \dots u_{m+1}} - v \hat{q}$ в этом случае, согласно лемме 8, получается именно оно) прибавляем «для компенсации» число v и получаем, если не обращать внимание на

$(m+2)$ -й разряд, число

$$\overline{w_1 \dots w_m} = \overline{u_1 \dots u_{m+1}} - v\hat{q}.$$

Схемой это число вычисляется так:

$$s = \overline{u_1 \dots u_{m+1}} - v\hat{q}, \quad x = v_i s, \quad i = 1, \dots, m,$$

$$\overline{w_1 \dots w_m} = (\overline{u_1 \dots u_{m+1}} - v\hat{q}) + \overline{x_1 \dots x_m}.$$

Сложность равна, согласно леммам 8 и 9,

$$3m - 2 + 2m + 1 + 2m - 1 = 7m - 2,$$

так как m операций тривиальны и в ручном счете ими можно пренебречь. Далее заменяем n -значное число $u = \overline{u_1 \dots u_n}$ на $(n-1)$ -значное число

$$\overline{w_1 \dots w_m u_{m+2} \dots u_n}$$

и задача сводится к делению этого числа на v . Отсюда имеем оценку

$$D_0(n, m) \leq D_0(n-1, m) + (7m - 2 + 7).$$

Учитывая сложность умножения на d и вычисления первой цифры частного, а также применяя в дальнейшем индукцию, получаем оценку

$$D_0(n, m) \leq (n - m + 1)(7m + 5) + 3(n + m) + 5m + 2.$$

Полученный остаток от деления еще надо разделить на d (он, кстати, делится нацело), что согласно лемме 8 требует сложности m . В итоге имеем

$$D(n, m) \leq D_0(n, m) + m \leq 7nm + 8n + 10m + 7 - 7m^2.$$

Если учесть, что вычисление d , умножение на d , а потом деление на d в среднем выполняется со сложностью

$$\log_b 2 + (1 - \log_v 2)(3n + 4m),$$

а вычисление $\overline{w_1 \dots w_m}$ — со сложностью, не большей

$$7 + 3m - 2 + 2m + 1 + (2m - 1)\frac{3}{b} = \left(5 + \frac{6}{b}\right)m + 6 - \frac{3}{b},$$

то оценка сложности в среднем чуть уменьшается и имеет вид

$$\begin{aligned} D(n, m) \leqslant (n - m + 1) \left(\left(5 + \frac{6}{b}\right)m + 6 - \frac{3}{b} \right) + \\ + (1 - \log_b 2)(3n + 4m) + 2 \log_b 2 + \log_b 2(5m + 2) < \\ < \left(5 + \frac{6}{b}\right)mn + O(m + n) - 5m^2. \end{aligned}$$

Имеет смысл учесть, что операция деления двухразрядного числа на одноразрядное с вычислением остатка при больших b сложнее, чем операция умножения одноразрядных чисел. Так, при выполнении ее на калькуляторе, она фактически распадается на 3 операции (деление, умножение целой части частного на делитель и вычитание полученного произведения из делимого). Если считать, что сложность этой операции равна 3, то оценка сложности деления принимает вид

$$D(n, m) \leqslant 7nm + 10n + 8m + 13 - 7m^2,$$

а оценка, приведенная в лемме 8, заменяется на оценку

$$D(n, 1) \leqslant 3n.$$

□

Задача 25. При условии использования только операций $\&$, \vee , \oplus , \neg получите следующую оценку сложности умножения в двоичной системе:

$$D(n, m) \leqslant 12(n - m)m + O(n).$$

Глава 9

Как представляются отрицательные числа в компьютере

Когда было введено понятие отрицательных чисел, меньших нуля, математику, как науку, в которой самым важным является ясность и очевидность, накрыло непроницаемым облаком и ввергло в лабиринт парадоксов, один загадочней другого.

Лазарь Карно

Обычный школьный способ записи отрицательных чисел состоит в постановке знака минус перед записью модуля этого числа. Этот способ в компьютерной арифметике называется *прямым кодом*. В случае использования в компьютере у него есть очевидный недостаток, а именно неоднозначность записи нуля в этой системе («проблема минус нуля»). Для компьютера, как правило, удобнее дополнительный код, в котором, например, число $-(123\ 456\ 789)_{10}$ записывается как

$$-10^9 + 1 + (987\ 654\ 321)_{10} = (1\ 987\ 654\ 322)_{10}.$$

Единица в старшем разряде является символом знака и показывает, что данное число отрицательное. Запись $(a_0 a_1 \dots a_n)_{10}$ n -разрядного числа в дополнительном коде во всех случаях означает число $-a_0 10^n + (a_1 \dots a_n)_{10}$, где $(a_1 \dots a_n)_{10}$ — обычная запись положительного n -разрядного числа. Если символ знака $a_0 = 0$, то очевидно $(0 a_1 \dots a_n)_{10} = (a_1 \dots a_n)_{10}$.

Дополнительный код удобен еще и тем, что при вычитании из меньшего числа большего с помощью обычного алгоритма вы-

читания как раз и получается разность, записанная в дополнительном коде. Например, если вычесть из меньшего числа $(89)_{10}$ большее число $(98)_{10}$, то получится запись

$$\begin{array}{r} 89 \\ - 98, \\ \hline 1\ 91 \end{array}$$

так как в старший разряд мы должны поместить «заем», равный единице. Однако такой результат выглядит странно, и при использовании прямого кода в этом случае числа переставляют, опять выполняют вычитание, получают уже положительное число и перед ним ставят знак минус. Но при использовании дополнительного кода эти трудности исчезают, так как полученное число $(191)_{10}$ со знаковым разрядом 1 в дополнительном коде означает $-100 + 91 = -9$, и это верный результат.

На самом деле в компьютере дополнительный код, конечно, используется для двоичной системы, а не для десятичной. Применение дополнительного кода удобно еще и тем, что позволяет выполнять сложение чисел любых знаков тем же алгоритмом, который осуществляет сложение положительных чисел. Например, если компьютер был бы 8-разрядный, то сложение положительного и отрицательного чисел он бы выполнял следующим образом:

$$\begin{array}{r} 00111010 \\ + 10011111. \\ \hline 11011001 \end{array}$$

Первое слагаемое здесь равно

$$(00111010)_2 = (0111010)_2 = 32 + 16 + 8 + 2 = 58,$$

второе равно

$$(10011111)_2 = -128 + (0011111)_2 = -128 + 16 + 8 + 4 + 2 + 1 = -97,$$

сумма равна

$$(11011001)_2 = -128 + (1011001)_2 = -128 + 64 + 16 + 8 + 1 = -39,$$

что очевидно правильно, так как $58 + (-97) = -39$.

Сложение отрицательных чисел выполнялось бы, например, так:

$$\begin{array}{r} 10111010 \\ 11011111. \\ \hline 1\ 10011001 \end{array}$$

Получившийся в результате переноса девятый бит просто отбрасывается, и результат оказывается равным

$$(10011001)_2 = -128 + (0011001)_2 = -128 + 16 + 8 + 1 = -103.$$

Впрочем, если получившийся девятый бит воспринимать как знаковый, то результат будет таким же:

$$\begin{aligned} (110011001)_2 &= -256 + (10011001)_2 = -256 + 128 + (0011001)_2 = \\ &= -128 + (0011001)_2 = -103. \end{aligned}$$

Он очевидно правильный, так как

$$\begin{aligned} (10111010)_2 &= -128 + (0111010)_2 = -128 + 32 + 16 + 8 + 2 = -70, \\ (11011111)_2 &= -128 + (1011111)_2 = -128 + 64 + 16 + 8 + 4 + 2 + 1 = -33, \end{aligned}$$

и $-70 + (-33) = -103$. Однако если абсолютная величина суммы слишком велика (в случае 8-разрядных чисел больше 127), то результат сложения будет неправильным (будет иметь даже неверный знак), как в следующем примере:

$$\begin{array}{r} 10111010 \\ 10011111. \\ \hline 01011001 \end{array}$$

В результате сложения отрицательных чисел $(10111010)_2 = -70$ и $(10011111)_2 = -97$ у нас получилось не -167 , а $(01011001)_2 = 64 + 16 + 8 + 1 = 89$ (правда, возник перенос в девятый разряд, но мы вынуждены его отбросить). На это неприятное явление есть такой ответ. Во-первых, число -167 (и вообще любое число, большее по абсолютной величине 128) невозможно изобразить в нашей системе 8-разрядных знаковых чисел в дополнительном

коде, так как в этой системе записываются только числа из отрезка $\{-128, \dots, 127\}$, поэтому появление числа, большего 128 вызывает *переполнение*. Во-вторых, почти правильный ответ мы все же получили, так как он отличается от верного на 256, другими словами, наш алгоритм выполняет *сложение по модулю* $2^8 = 256$. Причина этого явления конечно в том, что мы отбросили появившийся девятый разряд. В реальных компьютерах все происходит примерно так же, только длина машинного слова сейчас 64 разряда.

Но у дополнительного кода есть и недостатки. Складывать и вычитать в нем действительно очень удобно. А вот умножать и делить — не очень. Для умножения удобнее упоминавшийся выше (и хорошо всем известный) прямой код. И вычислять абсолютную величину числа в дополнительном коде не так тривиально. Тем не менее, в цифровой технике почти повсеместно используется именно дополнительный код.

Задача 26. Докажите, что абсолютная величина числа $(a_8 \dots a_1)_2$, заданного в дополнительном коде, равна

$$(a_7 \oplus a_8, a_6 \oplus a_8, \dots, a_1 \oplus a_8)_2 + a_8.$$

Кроме дополнительного кода изредка используют также обратный код, в котором, например число $-(123\,456\,789)_{10}$ записывается как 876 543 210 (каждая цифра дополняется до 9). В этом коде сложение и вычитание производятся фактически по модулю $10^9 - 1$.

Глава 10

SRT-деление

В мире есть много трудных вещей, но нет ничего труднее, чем четыре действия арифметики.

Бéда Достопочтенный (673–735 гг.)

Здесь будет рассказано об алгоритме деления двоичных чисел, удобном для компьютерных вычислений и реально применяющемся при проектировании логических схем, выполняющих деление¹⁾.

Деление m -битного числа Z на n -битное число D в частном дает число Q , которое может быть $(m - n + 1)$ -битным или $(m - n)$ -битным, и остаток R , $0 \leq R < D$, такие, что $Z = QD + R$. Обычный школьный алгоритм деления столбиком удобно представлять в виде деления дроби $z = 2^{-m-1}Z = 0.0z_1 \dots z_m$ на дробь $d = 2^{-n}D = 0.d_1 \dots d_n$ (здесь и далее в этом разделе предполагается, что $d_1 = 1$, т. е. делитель, как говорят, *нормализован*), тогда частным будет число $Q = q_0 2^{m-n} + \dots + q_{m-n}$, а остатком — число

¹⁾ Любопытно, что при использовании аналогичного алгоритма фирма «Интел» допустила свой знаменитый баг в одной из ранних версий процессора «Пентиум». Баг состоял в том, что процессор неправильно делил. Ошибки, правда, возникали чрезвычайно редко, поэтому и были не замечены при его тестировании. После обнаружения бага специалисты по компьютерной арифметике «Интела» даже написали статью, в которой показали, что вероятность ошибочного деления чрезвычайно мала. Ошибку обнаружил британский специалист по теории чисел во время отлаживания своей программы, проверяющей числа на простоту. Программа иногда выдавала очевидно неверные результаты, и при поиске ошибки в программе он обнаружил, что компьютер неправильно делил, и послал пример ошибочного деления в «Интел».

$r = 2^{-n}R$, $z = dQ + r$, $0 \leq r < 1$. Для выполнения деления вначале полагаем $z_0 = z$, а потом на каждом шаге деления находим очередную цифру частного q_j и остаток $z_{j+1} = 2z_j - dq_j$, при этом после $m+1-n$ шагов имеем равенство

$$z_{m+1-n} = 2^{m+1-n}z_0 - d(2^{m-n}q_0 + 2^{m-n-1}q_1 + \dots + q_{m-n}),$$

в котором легко убедиться по индукции, откуда при $r = z_{m+1-n}$ получаем равенства

$$2^{m+1-n}z = dQ + r, \quad 2^{m+1}z = 2^n dQ + 2^n r,$$

откуда имеем при $D = 2^n d$, $R = 2^n r$

$$Z = 2^{m+1}z = DQ + R.$$

Выполняемые этим алгоритмом действия фактически совпадают с действиями, выполняемыми обычным алгоритмом деления чисел «в столбик», только в обычном алгоритме последние цифры делимого не сносятся вниз, если они реально не участвуют в вычислениях. Тот факт, что дробная часть числа z начинается с нуля, нужен лишь для того, чтобы и на первом шаге алгоритма остаток вычислялся по формуле $z_{j+1} = 2z_j - dq_j$ при $j = 0$. Если бы мы решили использовать при инициализации алгоритма число $0.z_1 \dots z_m$, то в приведенные выше формулы пришлось бы внести небольшие изменения, например на первом шаге остаток вычислялся бы по формуле $z_1 = z_0 - dq_0$. Для вычисления $q_j \in \{0, 1\}$ на каждом шаге этого алгоритма сначала надо сравнить числа $2z_j$ и d , и в случае если первое из них не меньше второго, выбрать $q_j = 1$ и далее выполнить вычитание, в противном случае — взять $q_j = 0$ (и вычитание делать не нужно). При вычислениях с «длинными» числами указанные процедуры довольно трудоемки (требуют времени, пропорционального числу n — «длине» делителя).

В конце 50-х гг. прошлого века американцы Свини, Робертсон и (независимо) Точер предложили модификацию школьного алгоритма деления, свободную от этих недостатков (но существенно более трудную для понимания), которую с тех пор

называют SRT-делением. Первая идея, которую они использовали, заключалась в применении двоичной системы с отрицательными цифрами, точнее с одной такой цифрой — минус единицей, обозначаемой далее $\bar{1}$. При этом на каждом шаге выбирается $q_j \in \{-1, 0, 1\}$ так, чтобы $z_{j+1} = 2z_j - dq_j \in [-d, d)$ (в обычном алгоритме всегда $z_{j+1} \in [0, d)$). Выбирать q_j можно по-разному, в SRT-делении используется следующий эффективный с вычислительной точки зрения способ: если $z_j \geq 0$, то $q_j = 1$ (и тогда $-d \leq z_{j+1} = 2z_j - d < 2d - d = d$), если $0 > z_j \geq -1/4$, то $q_j = 0$ (и тогда при $z_j < 1/4$ в силу предположения о нормализованности делителя имеем $-d \leq -1/2 \leq z_{j+1} = 2z_j < 1/2 \leq d$), и если $z_j < -1/4$, то $q_j = -1$ (и тогда даже при $z_j < 0$ имеем $-d = -2d + d \leq z_{j+1} = 2z_j + d < d$). Формулы для z_j и окончательные формулы для вычисления частного и остатка по сравнению с обычным алгоритмом по существу не меняются, за исключением того, что в формуле для частного q цифры q_j могут равняться и минус единице, т. е. частное может быть представлено в виде двоичной записи с отрицательными цифрами, например

$$1011\bar{1}001111\bar{1}\bar{1}01.$$

Преобразовать такую запись в обычную несложно, для этого надо в ней заменить все минус единицы на нули, получив некоторое число в обычной записи, потом заменить все единицы в исходной записи на нули, а все минус единицы в этой записи на единицы, и, наконец, из первого числа вычесть второе, например

$$1011\bar{1}001111\bar{1}\bar{1}01 = \frac{10110001110001}{00001000001100}.$$

$$10101001100101$$

Обратим внимание на одну особенность алгоритма: остаток в указанном алгоритме может оказаться отрицательным (но не меньшим $-d$). В этом случае для завершения работы алгоритма от полученного частного Q надо отнять единицу, а к остатку прибавить d . Так как остатки z_j могут быть отрицательными, а при вычислении каждого из них делитель d иногда прибавляется, а иногда

отнимается, удобно использовать для записи отрицательных чисел систему записи в дополнительном коде, в которой запись двоичной дроби $sa_0.a_{-1} \dots$ означает при знаковом бите $s = 0$ просто положительную дробь $a_0.a_{-1} \dots = a_0 + 2^{-1}a_{-1} + \dots$ (значит, $s = 0$ указывает на то, что число неотрицательное), а при знаковом бите $s = 1$ запись $sa_0.a_{-1} \dots$ означает число $-2 + a_0 + 2^{-1}a_{-1} + \dots$ (значит, $s = 1$ указывает на то, что число отрицательное), таким образом, в общем случае запись $sa_0.a_{-1} \dots$ означает число $-s + a_0.a_{-1} \dots - 2s + a_0 + 2^{-1}a_{-1} + \dots$. Например, запись 10.1011 означает число

$$-2 + 0.1011 = -2 + \frac{11}{16} = -1\frac{5}{16} = -1.0101$$

в стандартной записи (т. е. записи с помощью абсолютной величины и знака). Запись 11.1011 означает число

$$-2 + 1 + 0.1011 = -1 + 0.1011 = -1 + \frac{11}{16} = -\frac{5}{16} = -0.0101,$$

которое по модулю меньше 1, и поэтому может быть записано также в виде 1.1011. Аналогично запись вида 00.*** эквивалентна записи 0.***. Подобным же образом запись в дополнительном коде можно применять к дробям с любой целой частью, да и к целым числам тоже. Операция смены знака в этой системе записи делается не так просто, как в обычной. Например, $-0.0101 = 1.1011$, как было указано выше. В общем случае алгоритм смены знака такой:

$$-0.a_{-1} \dots a_{-k} = 1.b_{-1} \dots b_{-k},$$

$$(b_{-1} \dots b_{-k})_2 = (\neg a_{-1} \dots \neg a_{-k})_2 + 1,$$

где $\neg 1 = 0$, $\neg 0 = 1$. Зато в этой системе легко складывать числа любых знаков. Для этого, с некоторыми оговорками, годится тот же алгоритм, который складывает обычные положительные числа в двоичной записи. Нужно лишь при появлении переноса в разряд, отсутствующий у слагаемых, иногда именно его рассматривать как знаковый бит, а иногда его игнорировать, а если этого

переноса не происходит, то добавлять в этот разряд нуль и иногда его рассматривать как знаковый бит.

Например, при сложении положительных чисел 0.01011 и 0.011 получается ответ 0.10111, а при сложении чисел 0.01011 и 0.11 тоже получается ответ 01.00011, при сложении чисел 1.011 и 0.011 получается правильный ответ 1.11, при сложении чисел 1.111 и 0.11 получается ответ 10.101, который будет правильным, только если проигнорировать перенос и за результат принять 0.101, при сложении отрицательных чисел 1.001 и 1.010 результат будет 10.011, а при сложении чисел 1.111 и 1.01 результат будет $11.001 = 1.001$. Заметим, что если сумма чисел по модулю меньше 1, то знаковый бит всегда стоит сразу перед дробной частью.

Вычитание числа можно, например, заменить на прибавление этого же числа с противоположным знаком. Поэтому для организации вычисления $z_{j+1} = 2z_j - dq_j$ можно заранее вычислить $d' = -d$, как указано выше, и потом, в зависимости от значения $q_j = -1, 1, 0$, прибавлять d', d или 0 (в случае ручных вычислений последнее означает просто не делать ничего). Сравнение произвольного числа $t = t_0.t_{-1}t_{-2} \dots \in [-1, 1]$ с нулем или $-1/4$ в дополнительном коде делается очень легко. Нужно у t знать лишь 3 старших бита. Очевидно, $t \geq 0 \iff t_0 = 0$,

$$0 > t \geq -\frac{1}{4} \iff t_0.t_{-1}t_{-2} = 1.11 = -\frac{1}{4} \iff t_0t_{-1}t_{-2} = 1.$$

Используя двоичные цифры, можно представить $-1, 0, 1$ как 11, 00, 01. Поэтому цифры q_j можно представить как $(q_{j,1}, q_{j,0})$, а указанный выше алгоритм их вычисления записать в следующем виде:

$$t \geq 0 \iff t_0 = 0 \implies q_j = 1 \iff q_{j,1} = 0 \text{ \& } q_{j,0} = 1,$$

$$0 > t \geq -\frac{1}{4} \iff t_0t_{-1}t_{-2} = 1 \implies q_j = 0 \iff q_{j,1} = 0 \text{ \& } q_{j,0} = 0,$$

$$t < -\frac{1}{4} \iff t_0(\neg(t_{-1}t_{-2})) = 1 \implies q_j = -1 \iff q_{j,1} = 1 \text{ \& } q_{j,0} = 1,$$

откуда следуют формулы

$$q_{j,0} = \neg(t_0t_{-1}t_{-2}), \quad q_{j,1} = t_0(\neg(t_{-1}t_{-2})).$$

Заметим еще, что $q_{j,0} \neg q_{j,1} = \neg t_1$ (так как $(q_{j,0} = 1) \& (q_{j,1} = 0) \iff t_0 = 0$) и $q_{j,0} q_{j,1} = q_{j,1}$ (так как $q_{j,1} = 1 \implies q_{j,0} = 1$). Выразим теперь через $q_{j,0}, q_{j,1}$ биты числа $-dq_j$. Пусть $d' = d'_0.d'_1 \dots d'_n$, $d = 0.d_1 \dots d_n$. Обозначим биты числа $-q_j d = d''_0.d''_1 \dots d''_n$. Тогда для любого i имеем

$$d''_i = d'_i q_{j,0} \neg q_{j,1} \vee d_i q_{j,1} q_{j,0} = d'_i \neg t_0 \vee d_i q_{j,1},$$

где для $\alpha, \beta \in \{0, 1\}$ операция \vee (дизъюнкция) определяется равенствами

$$\alpha \vee \beta = \max(\alpha, \beta) = \alpha + \beta - \alpha\beta = \neg(\neg\alpha \neg\beta),$$

но проще выражается словами: дизъюнкция равна нулю, только если оба операнда нули, в противном случае она равна единице. В компьютерной реализации алгоритма для вычисления булевого вектора d'' можно побитово умножить вектор d' на бит $\neq t_0$, потом побитово умножить вектор d на бит q_1 и вычислить побитовую дизъюнкцию полученных векторов.

Приведем пример применения указанной выше упрощенной версии SRT-алгоритма. Пусть нужно разделить число $(1101110101)_2$ на $(101101)_2$. Переходим к двоичным дробям, т. е. полагаем $z = z_0 = 0.01101110101$, $d = 0.101101$. Вычисляем $d' = -d = 1.010011$ (в виде обычных дробей $d' = -1 + \frac{19}{64} = -\frac{45}{64} = -d$).

На первом шаге вычисляем

$$z_1 = 2z_0 - dq_0 = 2z_0 - d = 0.1101110101 + d' = \begin{array}{r} 0.1101110101 \\ + 1.010011 \\ \hline 0.0010100101 \end{array},$$

так как $z_0 > 0$, а поэтому $q_0 = 1$.

На втором шаге вычисляем

$$z_2 = 2z_1 - dq_1 = 2z_1 - d = 0.0010100101 + d' = \begin{array}{r} 0.010100101 \\ + 1.010011 \\ \hline 1.100111101 \end{array},$$

так как $z_1 > 0$, а поэтому $q_1 = 1$.

На третьем шаге вычисляем

$$z_3 = 2z_2 - dq_2 = 2z_2 + d = 11.00111101 + d = \begin{array}{r} 11.00111101 \\ + 00.101101 \\ \hline 11.11110001 \end{array},$$

так как $z_2 < 1.11 = -1/4$, а поэтому $q_2 = -1$.

На четвертом шаге вычисляем

$$z_4 = 2z_3 - dq_3 = 2z_3 = 111.1110001 = 11.1110001 = 1.1110001,$$

так как $0 > z_3 > 1.11 = -1/4$, а поэтому $q_3 = 0$.

На пятом шаге вычисляем

$$z_5 = 2z_4 - dq_4 = 2z_4 = 11.110001,$$

так как $0 > z_4 > 1.11 = -1/4$, а поэтому $q_4 = 0$.

На этом шаге алгоритм можно закончить с отрицательным остатком

$$r = z_5 = 11.110001 = 1.110001 = -1 + \frac{49}{64} = -\frac{15}{64}$$

и частным $Q = (11\bar{1}00)_2 = 16 + 8 - 4 = 20$, откуда имеем

$$D = 2^6 d = (101101)_2 = 32 + 8 + 4 = 45, \quad R = 2^6 r = -15,$$

$$Z = 2^{11} z_0 = (1101110101)_2 = 885 = 45 \cdot 20 - 15 = DQ + R.$$

Для получения неотрицательного остатка достаточно отнять от частного единицу (получим $(11\bar{1}00)_2 - 1 = (11000)_2 - (100)_2 - 1 = 19$), а к отрицательному остатку прибавить D :

$$R + D = \begin{array}{r} 1110001 \\ + 0101101 \\ \hline 0011110 \end{array} = 30.$$

Заметим еще, что если не остановиться на пятом шаге, а продолжить работу алгоритма до бесконечности, получим разложение дроби

$$\frac{0.01101110101}{0.101101} = \frac{1101110101}{10110100000} = \frac{885}{45} \cdot 2^{-5} = \frac{59}{96}$$

в бесконечную двоичную дробь с отрицательными цифрами

$$\frac{0.01101110101}{0.101101} = 0.11\bar{1}000\bar{1}0\bar{1}0\bar{1}\dots$$

с периодом $(0\bar{1})$. Легко видеть, что

$$\begin{aligned} 0.11\bar{1}000\bar{1}0\bar{1}0\bar{1}\dots &= 0.11 - 0.00100010101\dots = \\ &= \frac{3}{4} - \frac{1}{8} - \frac{1}{32} \left(\frac{1}{4} + \frac{1}{16} + \dots \right) = \frac{5}{8} - \frac{1}{32} \frac{1}{3} = \frac{59}{96}. \end{aligned}$$

В общем случае, конечно, тоже получается бесконечная двоичная периодическая дробь с отрицательными цифрами.

Вторая идея, которая используется в SRT-делении, заключается в том, что сложение для вычисления z_{j+1} по формуле $2z_j - q_j d$ выполнять в полном объеме не надо. Точнее, z_{j+1} будем представлять в виде пары чисел $u_{j+1}, v_{j+1} \in (-1, 1)$, таких, что $u_{j+1} + v_{j+1} = z_{j+1}$, а вычисление этой пары по предыдущей паре u_j, v_j будем выполнять с помощью неполного сложения

$$u_{j+1} + v_{j+1} = 2u_j + 2v_j + (-q_j d).$$

Под неполным сложением будем понимать операцию, которая преобразует три числа

$$a = a_0.a_1\dots a_k, \quad b = b_0.b_1\dots b_k, \quad c = c_0.c_1\dots c_k$$

в два числа

$$\begin{aligned} s &= s_0.s_1\dots s_k, \quad p = p_0.p_1\dots p_k, \quad s_i = a_i + b_i + c_i \bmod 2, \\ p_i &= a_i b_i \vee a_i c_i \vee b_i c_i, \quad i = 0, \dots, k. \end{aligned}$$

Числа $s_i, p_i \in \{0, 1\}$ получаются путем сложения трех чисел a_i, b_i, c_i и связаны с ними формулой $2p_i + s_i = a_i + b_i + c_i$. Ее можно проверить непосредственно, но проверка упрощается, если воспользоваться симметричностью формул для s_i и p_i . Действительно, достаточно выполнить проверку для случаев, когда среди чисел a_i, b_i, c_i имеется соответственно 0, 1, 2 или 3 единицы. Можно заметить также, что число p_i образуется в результате переноса,

который происходит только тогда, когда минимум два из трех чисел a_i, b_i, c_i равны 1. Отсюда следует, что

$$p_i = \max(a_i b_i, a_i c_i, b_i c_i) = a_i b_i \vee a_i c_i \vee b_i c_i.$$

Легко проверить также, что справедливы формулы

$$p_i = a_i b_i + a_i c_i + b_i c_i \bmod 2 = a_i b_i \oplus a_i c_i \oplus b_i c_i = a_i b_i \vee c_i (a_i \oplus b_i),$$

где операция $x \oplus y = x + y \bmod 2$. Из равенств $2p_i + s_i = a_i + b_i + c_i$ следует равенство

$$\begin{aligned} p_0 p_1 \dots p_k 0 + s_0 s_1 \dots s_k &= -(2p_0 + s_0) + \sum_{i=1}^k (2p_i + s_i) 2^{-i} = \\ &= -(a_0 + b_0 + c_0) + \sum_{i=1}^k (a_i + b_i + c_i) 2^{-i} = a + b + c, \end{aligned}$$

т. е. сумма двух чисел, получившихся в результате неполного сложения, равна сумме складываемых трех чисел. Например,

$$\begin{array}{r} 1110001 \\ + 0101101 \\ + 1000100 \\ \hline 0011000 \\ + 11001010 \\ \hline 11100010 \end{array} = -30.$$

В SRT-алгоритме можно на каждом шаге вместо одного числа z_j вычислять два числа u_j, v_j , сумма которых равна z_j , и только в самом конце работы алгоритма для получения остатка $r_n = z_n$ выполнить полное сложение чисел u_n, v_n . На первом шаге алгоритма можно взять $u_1 = 2z_1, v_1 = -dq_1$. Заметим еще, что для вычисления на каждом шаге алгоритма нужных для нахождения q_j разрядов $z_{j,1}, z_{j,0}, z_{j,-1}, z_{j,-2}$ числа $2z_j$ можно просто сложить 4-разрядные числа, полученные из $2u_j, 2v_j$ отбрасыванием всех разрядов, начиная с 3-го. При этом полученное число $2z'_j$ будет меньше, чем $2z_j$ на $1/4 + 1/4 = 1/2$. Но алгоритм выбора q_j был таков, что уменьшение указанного числа на величину, меньшую $1/2$, или вообще не приведет к изменению q_j , или

оно уменьшится на 1 по сравнению со значением, вычисленным по точному значению z'_j . Но при этом, хотя значение q_j изменится, новое значение z_{j+1} , вычисленное по измененному значению q_j , по-прежнему будет удовлетворять нужному нам условию $-d \leq z_{j+1} < d$. Действительно, если $2z'_j \geq 0$, то $q_j = 1$ (и так как $2z_j \geq 0$, то $-d \leq z_{j+1} = 2z_j - d < 2d - d = d$), если $0 > 2z'_j \geq -1/2$, то $q_j = 0$ (и тогда $2z_j < 1/2$, откуда имеем $-d \leq -1/2 \leq z_{j+1} = 2z_j < 1/2 \leq d$), и если $2z'_j < -1/2$, то $q_j = -1$ (и тогда $2z_j < 0$, откуда имеем $-d = -2d + d \leq z_{j+1} = 2z_j + d < d$).

Рассмотрим указанный выше пример работы SRT-алгоритма в его полном варианте. Напоминаем, что во всех числах старший разряд — знаковый! Пусть нужно разделить число $(1101110101)_2$ на $(101101)_2$. Переходим к двоичным дробям, т. е. полагаем $z = z_0 = 0.01101110101$, $d = 0.101101$. Вычисляем $d' = -d = 1.010011$.

На первом шаге вычисляем

$$\begin{aligned} z_1 &= 2z_0 - dq_0 = 2z_0 - d = \\ &= 0.1101110101 + d' = 0.1101110101 + 1.010011, \end{aligned}$$

так как $z_0 > 0$, а поэтому $q_0 = 1$.

На втором шаге вычисляем

$$\begin{aligned} z_2 &= 2z_1 - dq_1 = 2z_1 - d = 01.101110101 + 10.10011 + 1.010011 = \\ &= 01.101110101 + 10.10011 + 11.010011 = \\ &= 00.011011101 + 111.00110 = 0.011011101 + 1.00110, \end{aligned}$$

так как $z_1 > 0$, поэтому $q_1 = 1$, а неполное суммирование дает результат

$$\begin{array}{r} 01.101110101 \\ + 10.10011 \\ + 11.010011 \\ \hline 00.011011101 \\ + 111.00110 \end{array}$$

Заметим, что при выполнении неполного сложения младшие разряды самого длинного (первого) слагаемого, отсутствующие в двух других слагаемых, можно просто перенести в любое из двух чисел

результата, не выполняя фактически над этими разрядами никаких операций. Разумеется, аналогичным образом можно поступать и на следующих шагах алгоритма.

На третьем шаге вычисляем

$$\begin{aligned} z_3 &= 2z_2 - dq_2 = 2z_2 + d = 00.011011101 + 111.00110 + 0.101101 = \\ &= 0.011011101 + 1.00110 + 0.101101 = 10.00001001 + 001.11101 = \\ &= 10.00001001 + 01.11101, \end{aligned}$$

так как $2z'_2 = 0.011 + 1.001 = 1.100 \leq 1.1 = -1/2$, поэтому $q_2 = -1$, а неполное суммирование дает результат

$$\begin{array}{r} 00.11011101 \\ + 10.0110 \\ + 0.101101 \\ \hline 10.00001001 \\ + 001.11101 \end{array}.$$

Заметим, что если бы мы не отбросили два старших бита (числа при этом не изменились), пришлось бы делать неполное сложение с чуть более длинными числами:

$$\begin{array}{r} 000.11011101 \\ + 1110.0110 \\ + 0.101101 \\ \hline 1110.00001001 \\ + 00001.11101 \end{array},$$

разумеется, потом три старших бита тоже можно было бы откинуть.

На четвертом шаге вычисляем

$$z_4 = 2z_3 - dq_3 = 2z_3 + 0 = 100.0001001 + 011.1101 + 0,$$

так как

$$2z'_2 = 100.00 + 011.11 = 111.11 = 1.11 > 1.1 = -\frac{1}{2},$$

поэтому $q_3 = 0$. Так как $10 + 01 = 11 = 1 = 0 + 1$, то в проведенных выше вычислениях можно было бы откинуть старшие разряды

в обоих числах и вычислить z'_2 как $00.00 + 11.11 = 11.11 = 1.11$, а $z_4 = 00.0001001 + 11.1101$. В компьютерных вычислениях все равно нужно преобразовать сумму трех слагаемых, одно из которых нулевое, в сумму двух слагаемых согласно данному алгоритму:

$$\begin{array}{r} 00.0001001 \\ + 11.1101000 \\ + 00.0000000 \\ \hline 11.1100001 \\ + 000.001000 \end{array}$$

Тогда $z_4 = 11.1100001 + 000.001000 = 1.1100001 + 0.001000$.

На пятом шаге вычисляем

$$z_5 = 2z_4 - dq_4 = 2z_4 = 11.110001,$$

так как $2z'_4 = 1.110 + 0.001 = 1.111 > 1.1 = -1/2$, поэтому $q_4 = 0$. В другом варианте приходится вычислять

$$\begin{aligned} 2z'_4 &= 1000.00 + 0111.10 = 000.00 + 111.10 = \\ &= 0.00 + 1.10 = 1.10 \geq 1.1 = -\frac{1}{2}, \end{aligned}$$

но опять получаем, что $q_4 = 0$.

На этом шаге алгоритм можно закончить с отрицательным остатком

$$\begin{aligned} r = z_5 = 2z_4 &= 11.100001 + 00.01000 = 11.110001 = \\ &= 1.110001 = -1 + \frac{49}{64} = -\frac{15}{64} \end{aligned}$$

и частным $Q = (11\bar{1}00)_2 = 16 + 8 - 4 = 20$.

Рассмотрим вопрос о том, не будут ли удлиняться целые части результата при неполном сложении

$$U + V = 2u + 2v + qd$$

на каждом из шагов алгоритма. Очевидно, целая часть числа $d'' = dq$ может быть записана в виде d_0d_0 , так как это число по модулю меньше 1. Если числа u, v тоже по модулю меньше 1,

то применение неполного сложения к старшим двум разрядам чисел $2u, 2v, d''$ выглядит следующим образом (штрихи опускаем):

$$\begin{array}{r}
 + \quad u_0 \ u_{-1} \ u_{-2} \ \dots \\
 + \quad v_0 \ v_{-1} \ v_{-2} \ \dots \\
 + \quad d_0 \ d_{-1} \ d_{-2} \ \dots \\
 \hline
 + \quad s_1 \ s_0 \ s_{-1} \ \dots \\
 + \quad p_2 \ p_1 \ p_0 \ p_{-1} \ \dots
 \end{array}$$

Очевидно, разряд p_2 нельзя игнорировать, только если $p_2 = 1$, $p_1 = 0$ (при $p_2 = 0$ и $p_2 = p_1 = 1$ очевидно $p_2 p_1 p_0.* = p_1 p_0.*$), а тогда $2p < -4 + 1 + 1 = -2$. Так как сумма $2p + s$, равная очередному числу z_j , по модулю меньше 1, то это возможно лишь при $s_1 = 0$, $s_0 = 1$ (тогда $2 > s \geq 1$) и $p_0 = 1$ (потому что при $p_0 = 0$ выполняется неравенство $2p < -4 + 1 = -3$ и $2p + s < -1$ независимо от s). Но тогда

$$p_2 p_1 p_0.* + s_1 s_0.* = 101.* + 01.* = 110.* = 10.* < -1,$$

что невозможно, или

$$p_2 p_1 p_0.* + s_1 s_0.* = 101.* + 01.* = 111.* = 1.* = p_0.* + s_0.*,$$

поэтому в этом случае в числах $2p$ и s можно игнорировать два старших разряда, и полученные числа U, V будут как и исходные иметь только один разряд в целой части (знаковый). Рассмотрим теперь случай, когда $p_2 = 0$. Так как $2p + s = p_1 p_0.* + s_1 s_0.*$ по модулю меньше 1, то если $s, p \geq 0$, имеем $s = 00.*$, $p = 00.*$ (иначе $2p + s > 1$), если $s, p < 0$, имеем $s = 11.*$, $p = 11.*$ (иначе $2p + s < -1$, поэтому в обоих случаях $2p + s = p_1 p_0.* + s_1 s_0.* = p_0.* + s_0.*$, следовательно в обоих числах можно оставить только по одному разряду в целой части. Если же $ps < 0$, то возможны только варианты $p_1 = 1$, $s_1 = 0$ или $p_1 = 0$, $s_1 = 1$. Равенство $p_2 = 0$ возможно лишь при $u_0 + v_0 + d_0 \leq 1$, причем если $s_1 = 0$, тогда $u_0 = v_0 = d_0 = 0$, значит, $u, v, d \geq 0$, откуда $0 \leq 2p + s = 2u + 2v + d < 1$, поэтому $u_{-1} = v_{-1} = d_0 = 0$, следовательно $s_0 = 0$, $p_1 = 0$, $2p + s = p_1 p_0.* + s_1 s_0.* = 0p_0.* + 00.*$ и так

как $2p + s < 1$, то $p_0 = 0$, $2p + s = 00.* + 00.* = 0.* + 0.*$, т. е. и в этом случае в обоих числах можно оставить только по одному разряду в целой части. Если же $s_1 = 1$, то возможны, например, следующие варианты неполного сложения:

$$\begin{array}{r} 0\ 0.1\ \dots \\ +\ 0\ 0.1\ \dots \\ +\ 1\ 1.0\ \dots \\ \hline 1\ 1.0\ \dots \\ + \\ 0\ 0.1.*\ \dots \end{array},$$

$$\begin{array}{r} 0\ 1.*\ \dots \\ +\ 1\ 0.*\ \dots \\ +\ 0\ 0.*\ \dots \\ \hline 1\ 1.*\ \dots \\ + \\ 0\ 0.1.*\ \dots \end{array},$$

в которых оставить в числах s , $2p$ только по одному разряду в целой части без изменения их суммы нельзя. Поэтому в указанном алгоритме возможно появление чисел, в которых целая часть имеет два разряда, что приведет далее к еще большему ее удлинению (хотя этого и не произошло в приведенном выше примере).

Справиться с этим затруднением можно разными способами. Один из них заключается в том, чтобы после выполнения очередного неполного сложения в полученных числах $2p$ и s просто откинуть все старшие разряды в целой части, кроме самого младшего (который после этого становится знаковым разрядом). Сумма полученных чисел $2p$ и s при этом может измениться, но если это произойдет, то, очевидно, она изменится на целое четное число. Если же в этой сумме откинуть старший разряд, то полученный результат по-прежнему будет отличаться от правильного на целое четное число. Но правильный результат равен z_j и поэтому должен быть по модулю меньше единицы. В то же время сумма любых чисел вида $.*\dots$ (у которых имеется только один разряд в целой части, он же знаковый), после того как у нее откинули старший разряд, тоже имеет вид $.*\dots$ и поэтому по модулю меньше еди-

ницы. Так как два числа, по модулю меньшие единицы, разность которых является целым четным числом, очевидно равны, то при неполном сложении трех чисел и отбрасывании лишних разрядов у результата получаем два числа вида $*.***$, такие, что если их сложить и получить результат того же вида, проигнорировав перенос в следующий разряд, то полученный результат в точности совпадет с числом z_j , которое и должно появиться на очередном шаге алгоритма. Так, в приведенном выше примере

$$\begin{array}{r}
 0\ 0.\ 1\ \dots \\
 +\ 0\ 0.\ 1\ \dots \\
 +\ 1\ 1.\ 0\ \dots \\
 \hline
 1\ 1.\ 0\ \dots \\
 +\ \\
 0\ 0.\ 1.\ 0\ \dots
 \end{array}$$

сумма трех чисел равна $00.***$, сумма чисел $11.0***$ и $001.0***$ будет такой же, а после откидывания, как указано выше, старших разрядов из этих двух чисел получатся числа $1.0***$ и $1.0***$, сумма которых будет равна $10.***$, но если в ней откинуть старший (знаковый) разряд, то получим правильный результат $0.***$. Поэтому, если внести в алгоритм указанные изменения, то на каждом его шаге z_j и q_j будут вычисляться правильно, а на последнем шаге алгоритма для вычисления (возможно, отрицательного) остатка нужно будет сложить два полученных на этом шаге числа, но при этом откинуть (если надо) старший разряд.

Оценим сложность и глубину логической схемы (в базисе из всех двухместных булевых операций) для деления m -битного числа на n -битное нормализованное число. Эта схема состоит из подсхемы, вычисляющей

$$d' = -d = (1.d_1 \dots d_n)_2 + (0.0 \dots 01)_2$$

со сложностью $5n$ и глубиной $\log_2 n + 1$, и из $m - n + 1$ выстроенных друг за другом цепью подсхем A_j , каждая из которых выполняет неполное сложение двух m -битных U_j, V_j и одного $(n + 2)$ -битного числа D_j (в числах U_j, V_j откидываются старшие разряды, как было указано выше, а к числу D_j добавляется старший $n + 2$

разряд, равный предшествующему разряду, и он становится знаковым разрядом). Для этого вначале из чисел U_j, V_j составляются два числа, образованные четырьмя старшими разрядами, которые потом складываются, получается 4-разрядное число T_j (пятый разряд не может появиться, так как $|U_j + V_j| < 1$), по трем старшим разрядам которого t_1, t_0, t_{-1} вычисляются два разряда числа q_j . Сложение двух 4-разрядных чисел

$$\begin{array}{r} u_1 \ u_0 \ u_{-1} \ u_{-2} \\ + \ v_1 \ v_0 \ v_{-1} \ v_{-2} \\ \hline t_1 \ t_0 \ t_{-1} \ t_{-2} \end{array}$$

(без вычисления последнего разряда t_{-2}) можно выполнить следующей подсхемой сложности 14 и глубины 6: $a_i = u_i \oplus v_i$, $b_i = u_i \& v_i$, $t_{-1} = a_{-1} \oplus b_{-2}$, $w_{-1} = a_{-1}b_{-2} \vee b_{-1}$, $t_0 = a_0 + w_{-1}$, $w_0 = a_0w_{-1} \vee b_0$, $t_1 = a_1 + w_0$. Если w_0 вычислять по формуле $a_0a_{-1}b_{-2} \vee a_0b_{-1} \vee b_0$, то получится схема глубины 5 и сложности 16. Поэтому вычисление обоих разрядов числа q_j по формулам

$$q_{j,0} = \neg(t_1 t_0 t_{-1}) = \neg t_1 \vee \neg(t_0 t_{-1}), \quad q_{j,1} = t_1 (\neg(t_0 t_{-1}))$$

одновременно с t_1 можно выполнить подсхемой сложности 17 и глубины 7 или подсхемой сложности 19 и глубины 6. Далее подсхема, одновременно вычисляющая D_j и q_j , имеет сложность $3n + 2 + 19$ и глубину 7, так как младшие n разрядов можно вычислить по формуле

$$D_i = d'_i \neg t_1 \vee d_i q_{j,1} = d'_i \neg t_1 \vee (d_i \neg(t_0 t_{-1})) t_1,$$

а

$$\neg(t_0 t_{-1}) = \neg((a_{-1} \oplus b_{-2})(a_0 \oplus b_{-1}))$$

вычисляется с глубиной 4. Подсхема A_j теперь может быть получена из построенной подсхемы добавлением $n + 2$ параллельно расположенных подсхем сложности 5 и глубины 3, осуществляющих неполное сложение чисел U_j, V_j, D_j . Каждому разряду числа

D_j соответствует одна такая подсхема. Заметим, что в подсхемах, соответствующих четырем старшим разрядам, можно выбросить 7 элементов, вычисляющих попарные суммы и произведения старших разрядов чисел U_j, V_j , так как они уже есть в подсхеме, выполняющей сложение старших четырех разрядов этих чисел. Заметим еще, что у первой подсхемы A_j глубина будет равна $2 \log_2 n + 1 + 2 + 3$, так как глубина подсхемы, вычисляющей d' равна $2 \log_2 n + 1$. У каждой следующей подсхемы A_j глубина увеличивается на 9, потому что глубина входов D_j равна 7, глубина входов U_j, V_j равна 0 (в каждой подсхеме, вычисляющей $p_i = u_i v_i \vee (u_i \oplus v_i) d_i$, $s_i = (u_i \oplus v_i) \oplus d_i$), глубина входов d_i увеличивается на 2. Сложность каждой подсхемы A_j равна $3n + 21 + 5n + 3 = 8n + 24$. Выходы последней подсхемы A_{m-n+1} подаются на входы сумматора $(n+2)$ -разрядных чисел (на его выходе получается остаток $z_{m+1-n} \in [-d, d]$, являющийся $(n+2)$ -разрядным числом). Сложность этой схемы равна $9(n+2)$, а глубина $2 \log_2(n+2) + 1$. Частное получается в виде числа

$$(q_{0,1} q_{0,0}) \cdot (q_{1,1} q_{1,0}) \dots (q_{m-n,1} q_{m-n,0}),$$

состоящего из цифр $-1, 0, 1$, записанных в виде пар двоичных разрядов. Если в этом числе заменить отрицательные единицы на обычные нули, а положительные единицы и нули на обычные единицы и нули, то получится число с обычной двоичной записью

$$((\neg q_{0,1}) \& q_{0,0}, (\neg q_{1,1}) \& q_{1,0}, \dots, (\neg q_{m-n,1}) \& q_{m-n,0})_2,$$

а если заменить положительные единицы на нули, а отрицательные единицы на обычные единицы, то получится число с обычной двоичной записью

$$(q_{0,1} \& q_{0,0}, q_{1,1} \& q_{1,0}, \dots, q_{m-n,1} \& q_{m-n,0})_2.$$

Для перевода числа Q в обычную двоичную запись достаточно из первого числа вычесть второе. Для этого используем схему для вычитания $(m-n+1)$ -разрядных чисел. С учетом необходимости предварительного вычисления $2(m-n)$ конъюнкций сложность

этой схемы равна $11(m - n + 1)$ (переноса в $(m - n + 2)$ -й разряд не возникает), а глубина цепи схем A_j после присоединения соответствующих их выходов q_j к входам указанной схемы вычитания увеличится на $2 \log_2(m - n + 1)$. Если остаток z_{m+1-n} отрицательный (т. е. его знаковый разряд $z_{m+1-n,1} = 1$), то его превращаем в неотрицательный, прибавляя $dz_{m+1-n,1}$, для чего используем схему сложения $(n + 2)$ -разрядных чисел. Ее сложность равна $10(n + 2)$ (переноса в $(n + 3)$ -й разряд не возникает), а глубина этой схемы не больше $2 \log_2(n + 2)$. Для исправления частного к нему надо прибавить число $z_{m+1-n,1}$ (фактически эта операция выполняется только в случае отрицательного остатка z_{m+1-n}), что еще увеличивает сложность схемы на $5(n + 2)$, а глубину на $1 + \log_2(n + 2)$. Заметим, что глубина выходов, на которых реализуется частное Q , меньше, чем глубина выходов, на которых реализуется остаток. Поэтому вся схема деления имеет сложность $(m + 1 - n)(8n + 35) + 24(n + 2)$ и глубину не больше $9(m + 1 - n) + 5 \log_2(n + 2)$.

Глубину схемы деления можно уменьшить до $8(m + 1 - n) + 5 \log_2(n + 2)$, если схемы A_j построить следующим образом. Вместо вычисления $z_{j+1} = 2z_j - dq_j = 2(U_j + V_j) - dq_j$ параллельно вычисляем $2S_j + P_j = 2(U_j + V_j) - d$, $2S'_j + P'_j = 2(U_j + V_j) + d$, $2S''_j + P''_j = 2(U_j + V_j)$ с помощью двух схем неполного сложения со сложностью $8(n + 2)$ и глубиной 3 (в схемах неполного сложения имеется $2(n + 2)$ общих элементов, вычисляющих попарные суммы и произведения одноименных разрядов чисел U_j, V_j , а вычисление $S''_j = U_j$, $P''_j = 2V_j$ делается бесплатно). Потом выбираем из полученных трех пар чисел правильную пару по формулам

$$S_j \neg t_0 \vee S'_j q_{j,1} \vee S''_j \neg q_{j,0} = S_j \neg t_0 \vee t_0 (S'_j \neg (t_{-1} t_{-2}) \vee S''_j (t_{-1} t_{-2})),$$

$$P_j \neg t_0 \vee t_0 (P'_j \neg (t_{-1} t_{-2}) \vee P''_j (t_{-1} t_{-2})).$$

Сложность этих вычислений равна $6m + 19$ (одновременно вычисляются $q_{j,0}, q_{j,1}$), а глубина 8 (так как глубина $t_{-1} t_{-2}$ равна 4, а глубина $P_j, P'_j, P''_j, S_j, S'_j, S''_j$ не больше трех). Окончательно имеем, что глубина указанной реализации схемы A_j равна 8, а сложность $8(n + 2) + 6m + 19$. Поэтому сложность всей схемы деления

равна $(m + 1 - n)(8n + 6m + 46) + 24(n + 2)$. Например, для деления 16-разрядного числа на 8-разрядное получаем схему сложности не более 2094 и глубины не более 92. Если же делить 64-разрядное число на 32-разрядное, то сложность схемы будет 23454.

Задача 27. Докажите, что если не стремиться минимизировать глубину и воспользоваться упрощенным алгоритмом SRT-деления, то для его реализации можно построить схему сложности $7(m - n)n + O(n)$. Глубина при этом будет равна $O((m - n) \log_2 n)$.

Так как глубина схемы деления оказалась, как видим, очень большой, то обычно такие схемы делают многотактовыми, т. е. работающими как конечные автоматы, а не как чисто логические схемы. Для того, чтобы уменьшить число тактов применяют SRT-алгоритмы, использующие представление чисел не в двоичной, а четверичной или восьмеричной системе. Алгоритмы при этом становятся более изощренными, так как усложняется процедура вычисления очередной цифры частного. Именно при использовании четверичного SRT-алгоритма фирма «Интел» допустила свой знаменитый баг в одной из ранних версий процессора «Пентиум».

Глава 11

Быстрое деление многочленов

*Все следует сделать настолько простым,
насколько это возможно, но не проще.*

Альберт Эйнштейн
(цитируется по книге Джона Бентли
«Жемчужины программирования»)

Здесь будет показано, что деление можно выполнять с той же по порядку сложностью, что и умножение. Впервые это было сделано немецким математиком Фолкером Штрассеном в 70-е гг. XX века. Мы приведем другое доказательство, которое далее почти дословно будет перенесено и на случай деления многозначных чисел.

Обозначим $D_0(m, n)$ наименьшее количество арифметических операций над числами, требующихся для нахождения неполного частного при делении многочлена степени не выше m на многочлен степени n , а через $D(m, n)$ — сложность деления этих многочленов, в которую включается также сложность нахождения остатка от деления и произведения неполного частного на делитель. Очевидно, что при $m \leq n$ обе эти величины равны нулю. Далее для удобства изменим определение $M(n, m)$, заменив его на $M(n + 1, m + 1)$, т. е. обозначим $M(m, n)$ сложность умножения двух многочленов степеней m и n соответственно.

Для любых двух многочленов $p(x)$ и $q(x)$ обозначим $\lfloor p(x)/q(x) \rfloor$ целую часть рациональной функции $p(x)/q(x)$, т. е. неполное частное при делении $p(x)$ на $q(x)$. Дробную часть этой функции, т. е. разность $p(x)/q(x) - \lfloor p(x)/q(x) \rfloor$, обозначим $\{p(x)/q(x)\}$. Тогда остаток от деления $p(x)$ на $q(x)$ равен числителю последней дроби, т. е. $q(x)\{p(x)/q(x)\}$.

Нашей целью является доказательство следующих теорем. Пусть $M(n)$ обозначает любую функцию, такую, что

$$M(n) \leq M(n) \leq \frac{M(2n)}{2}.$$

Теорема 15. При $n \geq 1$ справедливы неравенства

$$D(2n, n) < D(2n, n) < 5M(n) + 3n + 2 \log_2 n.$$

Теорема 16. При $m \geq 2n$ справедлива оценка

$$D(m, n) < \frac{2mM(n)}{n} + 3M(n) + m + 4n,$$

а при $n \rightarrow \infty$ и предположении, что $M(n)/n \rightarrow \infty$, справедливо асимптотическое неравенство

$$D(m, n) \lesssim 2mM(n)/n.$$

При $n \leq m < 2n$ справедлива оценка¹⁾

$$D(m, n) \leq \left\lceil \frac{n}{m-n} \right\rceil M(m-n) + \\ + 4M(m-n) + 3n + (m-n) + o(m-n),$$

а при условиях $m-n \rightarrow \infty$, $(m-n)/n \rightarrow 0$ и предположении, что $M(n)/n \rightarrow \infty$ и $M(n)/n = o(n)$, справедливо асимптотическое неравенство

$$D(m, n) \lesssim nM(m-n)/(m-n).$$

Доказательства теорем занимают несколько строк и будут приведены в конце раздела. Но предварительно придется дока-

¹⁾ Далее иногда будем пользоваться стандартным обозначением: $f = o(g)$, если f/g стремится к нулю (при выполнении указанных в контексте условий, например при стремлении некоторого параметра к бесконечности).

зять несколько лемм. Заметим, что для получения оценок вида $D(m, n) = O(mM(n)/n)$ леммы 17, 22–24 и 26 не нужны, не нужно последнее равенство в лемме 20, а леммы 21 и 25 нужны в упрощенном и ослабленном виде.

Задача 28. Докажите, что если определить $D(m, n)$, не требуя вычисления произведения неполного частного на делитель, то сложность уменьшается не более чем на n .

Указание. Для нахождения произведения неполного частного на делитель достаточно вычесть из делимого остаток.

Лемма 11. При $m \geq n$ справедливо неравенство

$$D(m, n) \leq D_0(m, n) + M(m - n, n) + n.$$

Доказательство. Для нахождения остатка воспользуемся равенством $r = f - g[f/g]$ и тем фактом, что при вычитании все члены со степенями не ниже n -й сокращаются, поэтому для вычисления остатка достаточно найти разности n младших коэффициентов. \square

Лемма 12. При $m \geq n$ справедливо неравенство

$$D(m, n) \leq D(2n, n) \left(\left\lceil \frac{m}{n} \right\rceil - 1 \right) < \frac{mD(2n, n)}{n}.$$

Доказательство. Представим многочлен степени m в виде

$$f_0 + f_1 x^n + \dots + f_k x^{nk},$$

где $k = \lceil m/n \rceil - 1$ и степени многочленов f_i меньше n (у f_k степень может быть равна n), и, пользуясь тождеством

$$\begin{aligned} f &= f_0 + f_1 x^n + \dots + f_k x^{nk} = f' + h_k g x^{n(k-1)} = \\ &= (f_0 + f_1 x^n + \dots + f_{k-2} x^{n(k-2)} + r_k x^{n(k-1)}) + h_k g x^{n(k-1)}, \end{aligned}$$

где $h_k = \lfloor (f_k x^n + f_{k-1})/g \rfloor$ — частное, а $r_k = f_k x^n + f_{k-1} - gh_k$ — остаток от деления $f_k x^n + f_{k-1}$ на g , сведем деление f на g к де-

лению f' на g и получим неравенство

$$D(m, n) \leq D(nk + n, n) \leq D(nk, n) + D(2n, n),$$

из которого индукцией получаем неравенство

$$\begin{aligned} D(m, n) &\leq D(nk + n, n) \leq kD(2n, n) = \\ &= D(2n, n) \left(\left\lceil \frac{m}{n} \right\rceil - 1 \right) < \frac{m}{n} D(2n, n). \end{aligned}$$

Для обоснования последнего неравенства применяем следующую далее лемму 19. Лемма доказана. \square

Обозначим $R(m, n)$ наименьшее количество арифметических операций над числами, требующихся для нахождения неполного частного при делении многочлена x^m на произвольный многочлен степени n . Очевидно, что

$$R(m, n) \leq D(m, n).$$

Правильной дробью назовем дробь, у которой степень числителя меньше степени знаменателя, а полуправильной — дробь, у которой степень числителя не больше степени знаменателя. Очевидно, полуправильная дробь есть сумма правильной дроби и константы.

Лемма 13. Сумма правильных дробей — правильная дробь, а сумма полуправильных — полуправильная. Произведение полуправильных дробей — полуправильная дробь, а произведение правильной на полуправильную — правильная дробь. Любая дробь однозначным образом представляется в виде суммы целого многочлена и правильной дроби.

Доказательство. Доказательство первых четырех утверждений легко получается с помощью того факта, что степень произведения многочленов равна сумме их степеней. Последнее утверждение следует из того, что многочлен может совпадать с правильной дробью, лишь когда он равен нулю. \square

Лемма 14. Сумма дробных частей нескольких рациональных функций равна дробной части их суммы, и то же самое верно для суммы их целых частей. В частности, целый многочлен можно выносить за знак целой части при суммировании.

Доказательство обоих утверждений легко получается с помощью первого и последнего утверждений предыдущей леммы.

Лемма 15. Для любой рациональной функции R и любой полуправильной дроби R справедливо равенство

$$\lfloor R_1 \lfloor R_2 \rfloor \rfloor = \lfloor R_1 R_2 \rfloor.$$

Доказательство. Согласно леммам 13 и 14 справедливо равенство

$$\lfloor R_1 R_2 \rfloor = \lfloor R_1 (\lfloor R_2 \rfloor + \{R_2\}) \rfloor = \lfloor R_1 \lfloor R_2 \rfloor + R_1 \{R_2\} \rfloor = \lfloor R_1 \lfloor R_2 \rfloor \rfloor. \quad \square$$

Лемма 16. Для любого многочлена f степени $m \geq n$ деление на многочлен x^n осуществляется бесплатно, т.е. не требует выполнения арифметических операций.

Доказательство. Действительно, если $f = a_m x^m + \dots + a_0$, то

$$\left\lfloor \frac{f}{x^n} \right\rfloor = a_m x^{m-n} + \dots + a_n, \quad x^n \left\{ \frac{f}{x^n} \right\} = a_{n-1} x + \dots + a_0. \quad \square$$

Лемма 17. При $m \geq k$ сложность вычисления $k+1$ старшего коэффициента в произведении многочленов степеней m и n не превосходит величины $M(k, n)$.

Доказательство. Воспользуемся вытекающим из леммы 15 тождеством

$$\left\lfloor \frac{fg}{x^{m-k+n}} \right\rfloor = \left\lfloor \left\lfloor \frac{f}{x^{m-k}} \right\rfloor \frac{g}{x^n} \right\rfloor,$$

где степени многочленов f и g равны m и n соответственно. Согласно предыдущей лемме для нахождения $(k+1)$ -го старшего коэффициента в произведении многочленов f и g достаточно вычислить левую часть этого тождества. В силу той же леммы многочлен $\lfloor f/x^{m-k} \rfloor$ вычисляется бесплатно, и для вычисления правой части тождества достаточно выполнить умножение многочленов степеней n и k , на что требуется не более $M(k, n)$ арифметических операций. \square

Лемма 18. При $m \geq n$ справедливо неравенство

$$D_0(m, n) \leq R(m, n) + M(m - n).$$

Доказательство. Воспользуемся вытекающим из леммы 15 тождеством

$$\left\lfloor \frac{f}{g} \right\rfloor = \left\lfloor \frac{f}{x^m} \left\lfloor \frac{x^m}{g} \right\rfloor \right\rfloor = \left\lfloor f \frac{\lfloor x^m/g \rfloor}{x^m} \right\rfloor,$$

где степени многочленов f и g равны m и n соответственно.

Далее применяем лемму 17 при $k=m-n$ для оценки сложности вычисления k старших коэффициентов произведения $\lfloor f \lfloor x^m/g \rfloor \rfloor$, т. е. $\lfloor f/x^m \lfloor x^m/g \rfloor \rfloor$. \square

Лемма 19. Функция $M(n, m)$ монотонна по обоим переменным, а $D(m, n)$ и $R(m, n)$ — только по первой.

Доказательство. Первые два утверждения следуют из того, что при формальном добавлении к сомножителям старших членов с нулевыми коэффициентами произведение не меняется, и при добавлении к делимому старших членов с нулевыми коэффициентами частное и остаток тоже не меняются, так как коэффициенты частного и остатка выражаются в виде рациональных функций от коэффициентов делимого и делителя со знаменателями, являющимися степенями старшего коэффициента делителя. Последнее

утверждение следует из тождества

$$\left\lfloor \frac{x^m}{g} \right\rfloor = \left\lfloor \frac{\lfloor x^{n+m}/g \rfloor}{x^n} \right\rfloor,$$

вытекающего из леммы 15 и леммы 16. Немонотонность по второму аргументу следует из того, что при $m < n$ очевидно

$$D(m, n) = 0.$$

□

Следующая лемма является основой алгоритма быстрого деления.

Лемма 20. Пусть многочлен $P_k = \lfloor x^{k+n}/g \rfloor$, где g — заданный многочлен степени n со старшим коэффициентом a , а следующим коэффициентом — b . Тогда $P_1(x) = a^{-1}x - a^{-2}b$,

$$P_{2k+1} = 2 \cdot x^{k+1} P_k - \left\lfloor \frac{P_k^2 g}{x^{n-1}} \right\rfloor, \quad P_{2k} = 2 \cdot x^k P_k - \left\lfloor \frac{P_k^2 g}{x^n} \right\rfloor.$$

Доказательство. Первое равенство проверяется непосредственно, так как многочлен

$$x^{n+1} - P_1(x)g = x^{n+1} - (a^{-1}x - a^{-2}b)(a^n x + b^{n-1}x + \dots + c)$$

имеет степень меньше n , что становится очевидным после раскрытия скобок, и значит, он является остатком при делении x^{n+1} на g . Для доказательства второго равенства заметим, что

$$P_k g = x^{k+n} - gr,$$

где

$$r = \left\{ \frac{x^{k+n}}{g} \right\},$$

так как

$$P_k + r = \left\lfloor \frac{x^{k+n}}{g} \right\rfloor + \left\{ \frac{x^{k+n}}{g} \right\} = \frac{x^{k+n}}{g},$$

учтем, что степень многочлена gr не выше $n-1$, и выведем с помощью лемм 14 и 15 равенства

$$\begin{aligned}
\left\lfloor \frac{P_k^2 g}{x^{n-1}} \right\rfloor &= \left\lfloor \frac{P_k(x^{k+n} - gr)}{x^{n-1}} \right\rfloor = \\
&= \left\lfloor P_k x^{k+1} - \frac{P_k gr}{x^{n-1}} \right\rfloor = P_k x^{k+1} - \left\lfloor \frac{P_k gr}{x^{n-1}} \right\rfloor = \\
&= P_k x^{k+1} - \left\lfloor \frac{x^{k+n}}{g} \frac{gr}{x^{n-1}} \right\rfloor = P_k x^{k+1} - \lfloor x^{k+1} r \rfloor,
\end{aligned}$$

из которых, согласно лемме 14, немедленно следует равенство

$$\begin{aligned}
2 \cdot x^{k+1} P_k - \left\lfloor \frac{P_k^2 g}{x^{n-1}} \right\rfloor &= 2 \cdot x^{k+1} P_k - P_k x^{k+1} + \lfloor x^{k+1} r \rfloor = \\
&= P_k x^{k+1} + \lfloor x^{k+1} r \rfloor = \lfloor P_k x^{k+1} + x^{k+1} r \rfloor = \\
&= \lfloor x^{k+1} (P_k + r) \rfloor = \\
&= \left\lfloor x^{k+1} \left(\left\lfloor \frac{x^{k+n}}{g} \right\rfloor + \left\{ \frac{x^{k+n}}{g} \right\} \right) \right\rfloor = \\
&= \left\lfloor x^{k+1} \left(\frac{x^{k+n}}{g} \right) \right\rfloor = \left\lfloor \frac{x^{2k+1+n}}{g} \right\rfloor = P_{2k+1}.
\end{aligned}$$

Последнее равенство леммы доказывается совершенно аналогично. \square

Лемма 21. Положим для краткости $R_k = R(n+k, n)$. Тогда при любом k справедлива оценка

$$R_k \leq R_{\lfloor k/2 \rfloor} + M\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + M(k) + 2\left\lfloor \frac{k}{2} \right\rfloor + 2.$$

Доказательство. Рассмотрим случай нечетного k . Применяя леммы 20 и 16, получаем неравенство

$$R_k \leq R_{\lfloor k/2 \rfloor} + M\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + M\left(2\left\lfloor \frac{k}{2} \right\rfloor, n\right) + 2\left\lfloor \frac{k}{2} \right\rfloor + 2.$$

Далее при $2\lfloor k/2 \rfloor < n$ воспользуемся леммами 17 и 16 и заметим, что для вычисления $\lfloor P_{\lfloor k/2 \rfloor}^2 g / x^{n-1} \rfloor$ достаточно найти $2\lfloor k/2 \rfloor + 2$

старших коэффициента произведения $P_{\lfloor k/2 \rfloor}^2 g$, что можно сделать со сложностью

$$M\left(2\left\lfloor \frac{k}{2} \right\rfloor + 1, 2\left\lfloor \frac{k}{2} \right\rfloor\right) \leq M(k).$$

В случае четного k последняя оценка заменяется на

$$M\left(2\left\lfloor \frac{k}{2} \right\rfloor\right) = M(k).$$

Заметим, что оценка

$$R_k \leq R_{\lfloor k/2 \rfloor} + O(M(k))$$

доказывается совсем просто, при этом не требуется лемма 17 и в силу леммы 19 достаточно рассмотреть случай нечетного k . \square

Лемма 22. При любом действительном x и натуральном n

$$\left\lfloor \frac{x}{n} \right\rfloor = \left\lfloor \frac{\lfloor x \rfloor}{n} \right\rfloor.$$

Доказательство. Действительно, $\lfloor x/n \rfloor$ — это количество чисел на отрезке от 0 до x , кратных n . Но на отрезке от 0 до $\lfloor x \rfloor$ лежит такое же количество этих чисел. Случай отрицательного x сводится к случаю положительного с помощью тождества $\lfloor -x \rfloor = -\lfloor x \rfloor - 1$. \square

Лемма 23. Положим

$$\lfloor k \rfloor_i = \lfloor \lfloor k \rfloor_{i-1} \rfloor, \quad \lfloor k \rfloor_1 = \left\lfloor \frac{k}{2} \right\rfloor, \quad m = \lfloor \log_2 k \rfloor,$$

и обозначим $\nu(k)$ сумму всех цифр двоичной позиционной записи числа k . Тогда

$$\sum_{i=1}^m \lfloor k \rfloor_i = \sum_{i=1}^m \left\lfloor \frac{k}{2^i} \right\rfloor = k - \nu(k).$$

Доказательство. Сначала индукцией по i с помощью предыдущей леммы проверяем, что

$$[k]_i = [[k]_{i-1}]_1 = \left\lfloor \frac{[k/2^{i-1}]}{2} \right\rfloor = \left\lfloor \frac{k/2^{i-1}}{2} \right\rfloor = \left\lfloor \frac{k}{2^i} \right\rfloor.$$

Потом применяем индукцию по m и проверяем, что в случае четного k имеем $\nu(k) = \nu(k/2)$ и

$$\begin{aligned} \sum_{i=1}^m [k]_i &= \sum_{i=1}^m \left\lfloor \frac{k}{2^i} \right\rfloor = \sum_{i=1}^m \left\lfloor \frac{k/2}{2^{i-1}} \right\rfloor = \frac{k}{2} + \sum_{i=1}^{m-1} \left\lfloor \frac{k/2}{2^i} \right\rfloor = \\ &= \frac{k}{2} + \frac{k}{2} - \nu\left(\frac{k}{2}\right) = k - \nu(k), \end{aligned}$$

а в случае нечетного k имеем

$$k = 2 \left\lfloor \frac{k}{2} \right\rfloor + 1, \quad \nu(k) = \nu\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + 1,$$

$$\begin{aligned} \sum_{i=1}^m [k]_i &= \sum_{i=1}^m \left\lfloor \frac{k}{2^i} \right\rfloor = \left\lfloor \frac{k}{2} \right\rfloor + \sum_{i=2}^m \left\lfloor \frac{[k/2]}{2^{i-1}} + 2^{-i} \right\rfloor = \\ &= \left\lfloor \frac{k}{2} \right\rfloor + \sum_{i=2}^m \left\lfloor \frac{[k/2]}{2^{i-1}} \right\rfloor = \left\lfloor \frac{k}{2} \right\rfloor + \sum_{i=1}^{m-1} \left\lfloor \frac{[k/2]}{2^i} \right\rfloor = \\ &= 2 \left\lfloor \frac{k}{2} \right\rfloor - \nu\left(\left\lfloor \frac{k}{2} \right\rfloor\right) = 2 \left\lfloor \frac{k}{2} \right\rfloor - \nu(k) + 1 = k = k - \nu(k). \quad \square \end{aligned}$$

Лемма 24. Справедливо неравенство $M(2n, n) \leq 2M(n) + n$.

Доказательство. Доказательство основано на тождестве

$$(f_1 x^n + f_0)g = f_1 g x^n + f_0 g,$$

где многочлены f_1 и g имеют степени n , а f_0 — степень $n-1$. \square

Лемма 25. Справедливо неравенство

$$R_k \leq 3M(k) + 2k + 2\lfloor \log_2 k \rfloor - 2\nu(k),$$

и, в частности, при $k = 2^s - 1$ справедливо неравенство

$$R_k \leq 3M(k) + 2k - 2.$$

При любых m и n , таких, что $m > n$, справедливо неравенство

$$R(m, n) < 3M(m - n) + 2(m - n) + 2\log_2(m - n).$$

Доказательство. Из лемм 21 и 23 вытекают неравенства

$$R_{\lfloor k \rfloor_{i-1}} - R_{\lfloor k \rfloor_i} \leq M(\lfloor k \rfloor_i) + M(2\lfloor k \rfloor_i) + 2\lfloor k \rfloor_i + 2,$$

где $\lfloor k \rfloor_i = \lfloor k/2^i \rfloor$, $0 \leq i \leq m$. Складывая их и пользуясь неравенствами

$$M(\lfloor k \rfloor_i) \leq M(\lfloor k \rfloor_i) \leq 2^{-i}M(2^i \lfloor k \rfloor_i) \leq 2^{-i}M(k), \quad M(1) \geq R(1)$$

и соотношениями леммы 23, получаем, что

$$R_k \leq 3M(k) + 2k - 2\nu(k) + 2\lfloor \log_2 k \rfloor.$$

Последнее неравенство леммы немедленно следует из первого.

Заметим, что равенство $R_k = O(M(k))$ доказывается при $k = 2^s - 1$ совсем просто, без использования леммы 23 и с помощью упрощенного варианта леммы 21. Применяя лемму 19, получаем это равенство без всяких ограничений на k . \square

Лемма 26. При $m \geq 2n$ справедлива оценка

$$D(m, n) < \frac{2mM(n)}{n} + 3M(n) + m + 4n,$$

а при $n \rightarrow \infty$ и предположении, что $M(n)/n \rightarrow \infty$, справедливо асимптотическое неравенство

$$D(m, n) \lesssim \frac{2mM(n)}{n}.$$

Доказательство. Повторяя доказательство леммы 12, заметим, что неравенство

$$D(m, n) \leq D(nk + n, n) \leq D(nk, n) + D(2n, n)$$

можно, применяя леммы 11 и 18, заменить на неравенство

$$D(m, n) \leq D(nk + n, n) \leq D(nk, n) + R(2n, n) + 2M(n) + n,$$

из которого по индукции можно вывести неравенство

$$D(m, n) \leq D(nk + n, n) \leq R(2n, n) + k(2M(n) + n),$$

если заметить, что процедура вычисления $[x^{2n}/g]$, имеющая сложность $R(2n, n)$, выполняется только один раз. Подставляя $\lceil m/n \rceil - 1$ вместо k и оценивая $R(2n, n)$ с помощью леммы 25 имеем

$$\begin{aligned} D(m, n) &\leq 3M(n) + 2 \left(\left\lceil \frac{m}{n} \right\rceil - 1 \right) M(n) + \left(\left\lceil \frac{m}{n} \right\rceil + 1 \right) n + 2 \log_2 n \leq \\ &\leq 3M(n) + \frac{2mM(n)}{n} + m + 2n + 2 \log_2 n, \end{aligned}$$

откуда и следует первое неравенство леммы. Второе неравенство очевидно следует из первого. Лемма доказана.

Заметим, что при больших m неравенство леммы точнее, чем оценка, которую можно получить из теоремы 15 и леммы 12. Например, применяя оценку Карацубы, из леммы 26 можно вывести асимптотическое неравенство

$$D(m, n) \lesssim \frac{70}{3} mn^{\log_2(3/2)}.$$

Очевидно, эта оценка при больших n точнее, чем оценка сложности школьного алгоритма, и всего лишь асимптотически в два раза хуже, чем аналогичная оценка сложности умножения многочленов. \square

Доказательство теоремы 15. Применяя леммы 11, 18, 24 и 25, имеем

$$\begin{aligned} D(2n, n) &< D_0(2n, n) + M(n) + n \leq M(n) + n + R(2n, n) + M(2n, n) \leq \\ &\leq 3M(n) + 3M(n) + 4n + 2 \log_2 n. \end{aligned}$$

Остается заметить, что $\log_2 n < n$, так как при $n=1$ и $n=2$ это верно, а производная $\log_2 x$ равна $1/(x \ln 2) < 2/x \leq 1$ при $x \geq 2$. \square

Заметим, что с помощью упрощенного варианта леммы 25 получается равенство $D(2n, n) = O(M(n))$.

Доказательство теоремы 16. Применяя леммы 11, 12, 18, 25, 26 и теорему 12, имеем при $m \geq 2n$ неравенство

$$D(m, n) < \frac{2mM(n)}{n} + 3M(n) + m + 4n,$$

а при $n \leq m < 2n$ имеем неравенство

$$D(m, n) \leq \left\lceil \frac{n}{m-n} \right\rceil M(m-n) + 4M(m-n) + 3n + (m-n) + o(m-n).$$

Последнее утверждение теоремы 16 немедленно следует из доказанного.

Равенство $D(m, n) = O(mM(n)/n)$ вытекает из упрощенного варианта теоремы 15. \square

Если $m > n$ и $m - n = o(n)$, то из теоремы 16 и теоремы Карацубы вытекает асимптотическая оценка

$$D(m, n) \lesssim \frac{nM}{m-n} m - n = O(n(m-n)^{\log_2(3/2)}),$$

которая очевидно точнее, чем оценка сложности школьного алгоритма деления многочленов.

Глава 12

Быстрое деление чисел

*Натуральные числа придумал Господь Бог.
Все остальное — дело рук человеческих.*

Леопольд Кронекер ¹⁾

Результаты этого раздела во многом аналогичны результатам предыдущего. Для простоты рассматриваем алгоритм деления только в двоичной системе счисления. Главная идея, лежащая в основе этого алгоритма, заключается в применении метода Ньютона, который обычно используется для приближенного решения уравнений. Хотя придумать алгоритм быстрого деления мог и Ньютон, но в реальности он был открыт только в 60-е гг. XX века (вероятно, впервые он был изложен в диссертации американского математика Стивена Кука в 60-е гг. XX века).

Теорема 17. При $m \geq 2n$ справедлива оценка

$$D(m, n) < \frac{2mM(n)}{n} + 3M(n) + O(m),$$

а при $n \rightarrow \infty$ и предположении, что $M(n)/n \rightarrow \infty$, справедливо асимптотическое неравенство

$$D(m, n) \lesssim \frac{2mM(n)}{n}.$$

При $m = 2n$ справедлива оценка

$$D(2n, n) < 5M(n) + O(n).$$

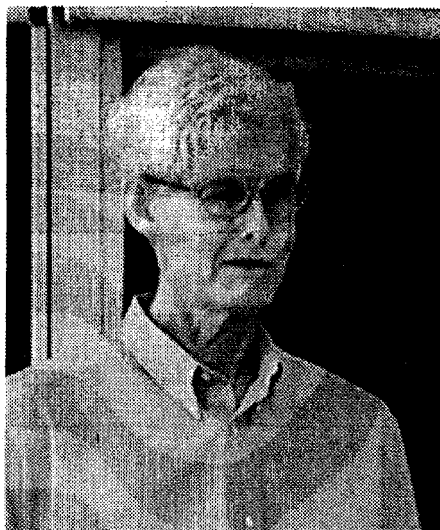
¹⁾ Леопольд Кронекер (1823–1891) — выдающийся немецкий математик, специалист по алгебре и теории чисел.

Теорема 18. При $n < m < 2n$ справедлива оценка

$$D(m, n) \leq \frac{nM(m-n)}{m-n} + 4M(m-n) + O(n),$$

а при условиях $m-n \rightarrow \infty$, $(m-n)/n \rightarrow 0$ и предположении, что $M(n)/n \rightarrow \infty$ и $M(n)/n = o(n)$, справедливо асимптотическое неравенство

$$D(m, n) \lesssim \frac{nM(m-n)}{m-n}.$$



Стивен Кук

Доказательства теорем будут приведены в конце раздела. Для доказательства понадобится ряд лемм. Отметим, что лемма 33, последнее неравенство леммы 28 и неравенства леммы 31 в нем не используются. Однако они нужны для более простого доказательства менее точной оценки

$$D(m, n) = O\left(\frac{mM(n)}{n}\right),$$

в котором в свою очередь не используются первые неравенства

леммы 28 и применяются упрощенные варианты лемм 29–32. Далее обозначаем $\{x\}$ ближайшее целое к числу x , причем при целом n полагаем

$$\left\{n + \frac{1}{2}\right\} = n + 1,$$

а вместо $\lfloor 2^n x \rfloor 2^{-n}$ пишем x_n . Легко проверить, что

$$|x - \{x\}| \leq \frac{1}{2} \quad \text{и} \quad x - 2^{-n} < x_n \leq x,$$

а x_n записывается двоичной дробью не более чем с n знаками после запятой.

Лемма 27. Пусть q — произвольное заданное число, $1/2 \leq q < 1$, а $z_k = z_{k,-1}z_{k,0}, z_{k,1} \dots z_{k,k}$ — двоичная k -разрядная дробь, такая, что

$$\left| z_k - \frac{1}{q} \right| \leq 2^{-k},$$

где

$$\frac{1}{2} \leq q < 1 \text{ и } z_{2k-1} = \{(2z_k - q_{2k+2}z_k^2)2^{2k-1}\}2^{1-2k}.$$

Докажите, что

$$\left| z_{2k-1} - \frac{1}{q} \right| \leq 2^{-2k+1}, \quad 1 \leq z_{2k-1} \leq 2$$

и z_{2k-1} — двоичная $(2k-1)$ -разрядная дробь.

Доказательство. Положим $z'_{2k-1} = 2z_k - q_{2k+2}z_k^2$, тогда

$$0 \leq z'_{2k-1} - 2z_k + qz_k^2 = (q - q_{2k+2})z_k^2 \leq 2^{-2k-2} \cdot 4 = 4^{-k}.$$

Так как

$$|1 - qz_k| \leq 2^{-k}q,$$

то

$$0 \leq \frac{1}{q} - 2z_k + qz_k^2 = \frac{(1 - qz_k)^2}{q} \leq 4^{-k}q \leq 4^{-k},$$

поэтому

$$\left| z'_{2k-1} - \frac{1}{q} \right| \leq 4^{-k}.$$

В силу неравенства $q_{2k+2} \geq 1/2$ имеем

$$z'_{2k-1} = 2z_k - q_{2k+2}z_k^2 \leq 2z_k - \frac{z_k^2}{2} = 2 - \frac{(2 - z_k)^2}{2} \leq 2.$$

Отсюда и из соотношения

$$|z'_{2k-1} - z_{2k-1}| = |z'_{2k-1} - \{z'_{2k-1}2^{2k-1}\}2^{1-2k}| \leq 4^{-k}$$

следует, что

$$z_{2k-1} = \{z'_{2k-1} 2^{2k-1}\} 2^{1-2k} \leq 2,$$

$$\left| z_{2k-1} - \frac{1}{q} \right| \leq \left| z'_{2k-1} - \frac{1}{q} \right| + 4^{-k} \leq 2^{-2k+1}.$$

Неравенство $z_{2k-1} \geq 1$ вытекает из предыдущего неравенства. Лемма доказана. \square

Обозначим R_k наименьшее количество операций над числами 0 и 1, требующихся для нахождения двоичной k -разрядной дроби

$$z_k = z_{k,-1} z_{k,0}, z_{k,1} \dots z_{k,k}, \quad 1 \leq z_k \leq 2,$$

такой, что

$$\left| z_k - \frac{1}{q} \right| \leq 2^{-k},$$

где q — заданная двоичная дробь, $1/2 \leq q < 1$, а $M(n)$ — любая монотонно неубывающая функция, удовлетворяющая соотношениям

$$M(n) \leq M(n) \leq \frac{M(2n)}{2}.$$

Лемма 28. Справедливы неравенства

$$M(n+1) \leq M(n) + 7n - 1,$$

$$M(2n, n) \leq 2M(n) + 3n - 1.$$

Доказательство. Для доказательства первого неравенства воспользуемся тождеством

$$(2u_1 + u_0)(2v_1 + v_0) = 4u_1v_1 + 2(u_1v_0 + v_1u_0) + u_0v_0,$$

где u_1 и v_1 — n -разрядные числа, а u_0 и v_0 — просто нули или единицы, и заметим, что для вычисления $2(u_1v_0 + v_1u_0) + u_0v_0$ достаточно $n + n + 2n - 1 + 1 = 4n$ операций, а для вычисления $4u_1v_1 + 2(u_1v_0 + v_1u_0) + u_0v_0$ — еще $M(n) + 3n - 1$ операция. Для

доказательства второго неравенства воспользуемся тождеством

$$(2^n u_1 + u_0)v_0 = 2^n u_1 v_0 + u_0 v_0,$$

где u_1, u_0 и v_0 — n -разрядные числа, и заметим, что вычисление суммы требует всего лишь $3n - 1$ операцию, если произведения уже готовы. \square

Лемма 29. При $k \geq 3$ справедливы неравенства

$$\begin{aligned} R_k &\leq R_{\lfloor k/2 \rfloor + 1} + M\left(\left\lfloor \frac{k}{2} \right\rfloor + 2\right) + M\left(2\left\lfloor \frac{k}{2} \right\rfloor + 4\right) + 7\left\lfloor \frac{k}{2} \right\rfloor + 12 \leq \\ &\leq R_{\lfloor k/2 \rfloor + 1} + M\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right) + M(k) + 17k + 22. \end{aligned}$$

Доказательство. Пусть $k = 2s - 1$ или $2s - 2$, тогда $s = \lfloor k/2 \rfloor + 1$. Обозначим z_s такую s -разрядную дробь, что для ее вычисления требуется не более R_s операций и

$$\left| z_s - \frac{1}{q} \right| \leq 2^{-s}.$$

Пользуясь леммой 27, получаем, что число

$$z_{2s-1} = (2z_s - q_{2s+2}z_s^2)2^{2s-1}2^{1-2s}$$

удовлетворяет неравенству

$$\left| z_{2s-1} - \frac{1}{q} \right| \leq 2^{-2s+1} \leq 2^{-k},$$

и для его вычисления достаточно $M(s+1) + M(2s+2) + 7s + 5$ операций. Действительно, можно считать, что $z_s \neq 2$ (иначе вообще не требуется умножений), поэтому для возведения в квадрат нужно $M(s+1)$ операций, для умножения q_{2s+2} на z_s достаточно $M(2s+2)$ операций, для вычисления $(2z_s - q_{2s+2}z_s^2)2^{2s-1}$ достаточно $5s + 4$ операции, и для нахождения z_{2s-1} нужна еще $2s + 1$

операция. Остается применить предыдущую лемму. Заметим, что неравенство

$$R_k \leq R_{\lfloor k/2 \rfloor + 1} + O(M(k))$$

доказывается проще и без использования леммы 28. \square

Лемма 30. Справедливо неравенство $R_k \leq 3M(k) + 46k$.

Доказательство. Положим

$$\lfloor k \rfloor_i = \lfloor \lfloor k \rfloor_{i-1} \rfloor_1, \quad \lfloor k \rfloor_1 = \left\lfloor \frac{k}{2} \right\rfloor_1 + 1, \quad m = \lfloor \log_2 k \rfloor.$$

С помощью леммы 22 докажем по индукции равенство

$$\lfloor k \rfloor_i = \left\lfloor \frac{k-2}{2^i} \right\rfloor + 2.$$

Из предыдущей леммы следует, что

$$R_{\lfloor k \rfloor_{i-1}} - R_{\lfloor k \rfloor_i} \leq M(\lfloor k \rfloor_i) + M(\lfloor k \rfloor_{i-1}) + 17\lfloor k \rfloor_i + 22, \quad 0 \leq i \leq m.$$

Сложим их и воспользуемся неравенствами

$$\begin{aligned} M(\lfloor k \rfloor) &\leq M\left(\left\lfloor \frac{k-2}{2^i} \right\rfloor\right) + 6\lfloor k \rfloor_i - 11 \leq M\left(\left\lfloor \frac{k-2}{2^i} \right\rfloor\right) + 6\lfloor k \rfloor_i - 11 \leq \\ &\leq 2^{-i}M(k) + 6\lfloor k \rfloor_i - 11, \quad 3 \geq R(2) = R(\lfloor k \rfloor_m), \end{aligned}$$

первое из которых вытекает из леммы 28, а последнее — из того, что

$$\left| 1, z_1 z_2 - \frac{1}{0, 1q_2 q_3 + \delta/8} \right| \leq \frac{1}{4}$$

при любых $q_i = 0$ или 1 , $0 \leq \delta \leq 1$ и

$$z_1 = 1 - q_2, \quad z_2 = \max(q_2, (1 - q_3)).$$

Отсюда, используя лемму 23, получаем, что

$$R_k < 3M(k) + 46k. \quad \square$$

Заметим, что при $k = 2^s + 1$ равенство $R_k = O(M(k))$ доказывается проще, без использования лемм 22 и 23 и при помощи ослабленного варианта предыдущей леммы и оценки $R(2) = O(1)$. Обозначим $R(m, n)$ наименьшее количество операций над числами 0 и 1, требующихся для нахождения $\lfloor 2^{m-1}/Q \rfloor$, где Q — произвольное двоичное n -разрядное число,

$$Q = \sum_{i=0}^{n-1} 2^i \alpha_{n-i},$$

$\alpha_j = 0$ или 1, $\alpha_{n-1} = 1$.

Лемма 31. При $k \geq 1$

$$R(k+n, n) \leq R_k + M(k, n) + 2k,$$

а при $m > n$

$$R(m, n) \leq 3M(m-n) + M(m-n, n) + 48(m-n).$$

Доказательство. Действительно, если при $q = 2^{-n}Q$

$$\left| z_{k-1} - \frac{1}{q} \right| \leq 2^{-k+1},$$

то

$$\left| 2^{k-1} z_{k-1} - \frac{2^{n+k-1}}{Q} \right| \leq 1,$$

а так как Q не степень двойки, то $2^{n+k-1}/Q$ не является конечной двоичной дробью, поэтому целое число $\lfloor 2^{n+k-1}/Q \rfloor$ равно $2^{k-1} z_{k-1}$ либо $2^{k-1} z_{k-1} - 1$. Число z_{k-1} находится со сложностью R_{k-1} . Сравнивая $2^{k-1} z_{k-1} Q$ с 2^{n+k-1} , находим $\lfloor 2^{n+k-1}/Q \rfloor$ точно. Для этого нужны дополнительные $M(k, n) + 2k$ операции. Последнее неравенство леммы вытекает из первого и предыдущей леммы. Заметим, что менее точное равенство $R(m, n) = O(M(m))$ легко вытекает из упрощенного варианта леммы 30 и следующей

далее леммы 33. Обозначим $D(m, n)$ сложность деления с остатком m -разрядного числа на n -разрядное число и одновременного вычисления произведения неполного частного на делитель. \square

Лемма 32. При $m > n$ справедливы неравенства

$$D(m, n) \leq R(m, n) + M(m, m-n) + M(n, m-n) + O(m),$$

$$D(m, n) \leq 4M(m-n) + M(n, m-n) + 47m - 44n + 7.$$

Доказательство. Применим леммы 30 и 31 и воспользуемся равенством

$$\left\lfloor \frac{f}{g} \right\rfloor = \left\lfloor f \frac{2^m/g}{2^m} \right\rfloor + \alpha,$$

где f — m -разрядное, g — n -разрядное двоичные числа, а $\alpha = 0$ или 1, вытекающим из неравенства

$$0 \leq \frac{f}{g} - \frac{f}{2^m} \left\lfloor \frac{2^m}{g} \right\rfloor \leq \frac{f}{2^m} < 1.$$

Для точного вычисления $\lfloor f/g \rfloor$ остается найти разность

$$f - g \left\lfloor f \frac{2^m/g}{2^m} \right\rfloor,$$

сравнить ее с числом g и в зависимости от результата, возможно, прибавить к числу

$$\left\lfloor f \frac{2^m/g}{2^m} \right\rfloor$$

единицу. Но таким образом получается лишь оценка

$$D(m, n) \leq R(m, n) + M(m, m-n) + M(n, m-n) + O(m).$$

Чтобы получить лучшую оценку, достаточно вместо числа $\lfloor 2^m/g \rfloor \leq 2^{m-n+1}$ взять целое число $a = 2^k z_k$ (где $k = m-n$, на вычисление которого требуется R_k операций), такое, что $\lfloor 2^m/g \rfloor$ равно a или $a-1$. Тогда

$$\left| f \frac{2^m/g}{2^m} - \frac{fa}{2^m} \right| \leq \frac{f}{2^m} < 1,$$

$$\left| \frac{fa}{2^m} - \left\lfloor \frac{f}{2^{m-k}} \right\rfloor \frac{a}{2^k} \right| < \frac{a}{2^k} \leq 1,$$

значит,

$$\left| \left\lfloor \frac{f \lfloor 2^m/g \rfloor}{2^m} \right\rfloor - \left\lfloor \left\lfloor \frac{f}{2^{m-k}} \right\rfloor \frac{a}{2^k} \right\rfloor \right| \leq 2.$$

На вычисление произведения $\lfloor f/2^{m-k} \rfloor a$ достаточно $M(k)$ операций, так как числа $\lfloor f/2^{m-k} \rfloor$ и a являются k -разрядными, а вычисление чисел $\lfloor f/2^{m-k} \rfloor$ и $b = \lfloor \lfloor f/2^{m-k} \rfloor a/2^k \rfloor$ делается бесплатно. Из доказанных неравенств следует, что $\lfloor f/g \rfloor = b + \beta$, где $\beta = -2, \dots, 3$ и произведение gb равно числу $g \lfloor f/g \rfloor$ или отличается от него на βg . Для точного вычисления $\lfloor f/g \rfloor$ и $g \lfloor f/g \rfloor$ достаточно сравнить $gb - f$ с числами ig , где $i = -1, 0, 1, 2$, и закончить доказательство так же, как и выше. Сложность выполнения указанных действий не больше

$$M(k) + M(n, k) + 3(n + 2) + k + 1$$

(так как числа $\pm g, 2g$ вычисляются бесплатно, а сравнение $gb - f$ с нулем не нужно, ибо уже сравнили f с gb). Остается применить лемму 29. \square

Лемма 33. Функция $M(n, m)$ монотонна по обоим переменным, а $D(m, n)$ и $R(m, n)$ — только по первой. При любых неотрицательных целых m, n, k, s

$$D(m + k + s, n + k) \geq D(m, n),$$

$$R(m + k + s, n + k) \geq R(m, n)$$

и, в частности, последовательность $D(n)$, равная по определению $D(2n, n)$, монотонна.

Доказательство. Первое утверждение следует из того, что при формальном добавлении к сомножителям старших разрядов с нулевыми цифрами произведение не меняется. Монотонность $D(m, n)$

по первому аргументу следует из того, что при добавлении к делимому старших разрядов с нулевыми цифрами частное и остаток тоже не меняются, так как цифры частного и остатка выражаются в виде функций алгебры логики от цифр делимого и делителя.

Монотонность $R(m, n)$ по первому аргументу выводится с помощью вытекающего из леммы 22 тождества

$$\left\lfloor \frac{\lfloor 2f/g \rfloor}{2} \right\rfloor = \left\lfloor \frac{f}{g} \right\rfloor.$$

Немонотонность по второму аргументу следует из того, что при $m < n$ очевидно $D(m, n) = 0$. Доказательство неравенства в случае $s = 0$ основано на применении очевидных тождеств

$$\left\lfloor \frac{2^k f}{2^k g} \right\rfloor = \left\lfloor \frac{f}{g} \right\rfloor, \quad 2^k g \left\{ \frac{2^k f}{2^k g} \right\} = 2^k g \left\{ \frac{f}{g} \right\}.$$

Случай $s > 0$ сводится к рассмотренному с помощью монотонности $D(m, n)$ по первому аргументу. \square

Задача 29. Докажите неравенства

$$M(n, m) \geq n + m - 1, \quad D(n, m) \geq n + m - 1.$$

Указание. Докажите, что произведение и частное двух чисел существенно зависят от всех их цифр.

Лемма 34. При $m \geq 2n$ справедливо неравенство

$$D(m, n) \leq D(n) \left\lfloor \frac{m-1}{n} \right\rfloor < \frac{(m-1)D(n)}{n}.$$

Доказательство. Лемма доказывается аналогично лемме 12. Представим m -разрядное число в виде

$$f_0 + f_1 2^n + \dots + f_k 2^{nk},$$

где $k = \lfloor (m-1)/n \rfloor$ и f_i суть n -разрядные числа и, пользуясь тождеством

$$\begin{aligned} f &= f_0 + f_1 2^n + \dots + f_k 2^{nk} = f' + h_k g 2^{n(k-1)} = \\ &= (f_0 + f_1 2^n + \dots + f_{k-2} 2^{n(k-2)} + r_k 2^{n(k-1)}) + h_k g 2^{n(k-1)}, \end{aligned}$$

где $h_k = \lfloor (f_k 2^n + f_{k-1})/g \rfloor$ — частное, а $r_k = f_k 2^n + f_{k-1} - gh_k$ — остаток от деления $f_k 2^n + f_{k-1}$ на g , сведем деление f на g к делению f' на g и получим неравенство

$$D(m, n) \leq D(nk + n, n) \leq D(nk, n) + D(2n, n),$$

из которого индукцией получаем неравенство

$$\begin{aligned} D(m, n) &\leq D(nk + n, n) \leq kD(2n, n) = \\ &= D(2n, n) \left\lfloor \frac{m-1}{n} \right\rfloor < \frac{m-1}{n} D(n). \end{aligned}$$

Остается применить лемму 33.

Аналогично лемме 26 доказывается следующая лемма. \square

Лемма 35. При $m \geq 2n$ справедлива оценка

$$D(m, n) < \frac{2mM(n)}{n} + 3M(n) + m + 50n + 7,$$

а при $n \rightarrow \infty$ и предположении, что $M(n)/n \rightarrow \infty$, справедливо асимптотическое неравенство

$$D(m, n) \lesssim \frac{2mM(n)}{n}.$$

Доказательство. Повторяя доказательство леммы 34, заметим, что неравенство

$$D(m, n) \leq D(nk + n, n) \leq D(nk, n) + D(2n, n)$$

можно, применяя лемму 32, заменить на неравенство

$$D(m, n) \leq D(nk + n, n) \leq D(nk, n) + R_n + 2M(n) + 4n + 7,$$

из которого по индукции можно вывести неравенство

$$D(m, n) \leq D(nk + n, n) \leq R_n + k(2M(n) + n),$$

если заметить, что процедура вычисления числа $a = 2^n z_n$, имеющая сложность R_n , выполняется только один раз. Подставляя $\lfloor (m-1)/n \rfloor$ вместо k и оценивая R_n с помощью леммы 30, имеем

$$\begin{aligned} D(m, n) &\leq 3M(n) + 2 \left\lfloor \frac{m}{n} \right\rfloor M(n) + \left(\left\lfloor \frac{m}{n} \right\rfloor + 50 \right) n + 7 \leq \\ &\leq 3M(n) + \frac{2mM(n)}{n} + m + 50n, \end{aligned}$$

откуда и следует первое неравенство леммы. Второе неравенство очевидно следует из первого. Лемма доказана.

Заметим, что при больших m, n неравенство леммы асимптотически в 2,5 раза лучше, чем оценка, вытекающая из лемм 32 и 34:

$$D(m, n) < \frac{mD(2n, n)}{n} < \frac{5mM(n)}{n} + 50m + \frac{7m}{n}.$$

Применяя оценку Карацубы, можно вывести равенство

$$D(m, n) = O(mn^{\log_2(3/2)}).$$

Очевидно, эта оценка при больших m, n лучше, чем оценка сложности школьного алгоритма. В то же время, если в указанном алгоритме деления $2n$ -разрядного числа на n -разрядное в качестве алгоритма умножения применить школьный алгоритм, то указанный алгоритм быстрым не будет, так как оценка его сложности асимптотически в 5 раз выше, чем у школьного алгоритма умножения, в то время как стандартные алгоритмы деления имеют, как было показано выше, почти такую же сложность, как и стандартный алгоритм умножения. \square

Доказательство теоремы 17. Применяя лемму 32, имеем

$$D(2n, n) \leq 5M(n) + 50n + 7,$$

а из леммы 35 при $m \geq 2n$ следует, что

$$D(m, n) < \frac{2mM(n)}{n} + 3M(n) + m + 50n + 7. \quad \square$$

Доказательство теоремы 18. Пользуясь леммой 32 и теоремой 12, имеем

$$\begin{aligned} D(m, n) &< 4M(m - n) + M(n, m - n) + 47m - 37n < \\ &< \frac{nM(m - n)}{m - n} + 4M(m - n) + O(m). \end{aligned} \quad \square$$

Если $m > n$ и $m - n = o(n)$, то из теоремы 18 и теоремы Карацубы вытекает асимптотическая оценка

$$D(m, n) \lesssim \frac{nM(m - n)}{m - n} = O(n(m - n)^{\log_2(3/2)}),$$

которая очевидно лучше, чем оценка сложности школьного алгоритма деления чисел.

Оценим сложность полученной схемы для деления 64-разрядного числа на 32-разрядное:

$$D(64, 32) \leq 5 \cdot 4924 + 50 \cdot 32 + 7 = 26\,227.$$

Если использовать для умножения 32-разрядных чисел не метод Карацубы, а школьный метод, то оценка станет равной 31 047. Обе схемы имеют большую сложность, чем схема SRT-деления, но глубина у второй из них может быть сделана несколько меньше, чем у схемы SRT. Для деления 128-разрядного числа на 64-разрядное схема быстрого деления уже будет иметь меньшую сложность, чем схема SRT-деления, так при удвоении числа разрядов сложность этой схемы приблизительно утраивается, а сложность SRT-схемы возрастает в 4 раза.

Для деления чисел близкой друг к другу разрядности указанный быстрый алгоритм более эффективен, например, схема для деления 40-разрядного числа на 32-разрядное имеет сложность 3328. Схема SRT-деления имеет сложность 5694.

Глава 13

Сравнение сложности умножения, деления, возведения в квадрат и извлечения квадратного корня

Будь силен в расчетах, и часу не проводи без повторения их, ибо математика — наука свирепая.

Кей-Кавус.
Кабус-Намэ (XI век)

Сначала докажем ряд лемм о сложности операций с многочленами.

Лемма 36. Пусть для рациональных функций R_1 и R_2 удвоенная степень многочлена $[R_2]$ не меньше степени многочлена $[R_1]$. Тогда справедливо равенство

$$\left\lfloor \frac{R_1}{[R_2]} \right\rfloor = \left\lfloor \frac{R_1}{R_2} \right\rfloor.$$

Доказательство. У рациональной функции $R_1/([R_2]R_2)$ степень числителя не больше степени знаменателя согласно условию леммы. Действительно, если у рациональной функции R_i степень числителя равна a_i , а степень знаменателя равна b_i , то у рациональной функции $R_1/([R_2]R_2)$ степень числителя равна $a = a_1 + b_2$, а степень знаменателя равна $b = b_1 + a_2 + a_2 - b_2$, а так как $2(a_2 - b_2) \geq a_1 - b_1$, то $a \leq b$. Отсюда, применяя лемму 13, выводим, что $R_1\{R_2\}/([R_2]R_2)$ — правильная дробь. Так как

$$\frac{R_1}{[R_2]} - \frac{R_1}{R_2} = \frac{R_1\{R_2\}}{[R_2]R_2},$$

то согласно лемме 14

$$\left\lfloor \frac{R_1}{\lfloor R_2 \rfloor} \right\rfloor = \left\lfloor \frac{R_1}{R_2} \right\rfloor.$$

что и доказывает лемму. \square

Лемма 37. Для любых многочленов f и g степени n справедливо тождество

$$fg = \left\lfloor \frac{x^{3n}f}{\lfloor x^{3n}/g \rfloor} \right\rfloor.$$

Доказательство. Достаточно применить лемму 36 при $R_1 = x^{3n}f$, $R_2 = \lfloor x^{3n}/g \rfloor$.

Положим для краткости $D(n) = D(2n, n)$. \square

Лемма 38. Справедливо неравенство

$$M(n) \leq 2D(n) + D(2n).$$

Доказательство. Применим лемму 37 для вычисления fg . Вычисление $x^{3n}f$ делается бесплатно, деление $\lfloor x^{3n}/g \rfloor$ делается со сложностью $2D(n)$ согласно лемме 12, деление $\lfloor x^{3n}f / \lfloor x^{3n}/g \rfloor \rfloor$ выполняется со сложностью $D(2n)$. \square

Лемма 39. При любых неотрицательных целых m, n, k, s справедливо неравенство

$$D(m + k + s, n + k) \geq D(m, n),$$

и, в частности, последовательность $D(2n, n)$ монотонна. Аналогичное утверждение справедливо для функции R .

Доказательство. В случае $s = 0$ доказательство основано на применении очевидных тождеств

$$\left\lfloor \frac{x^k f}{g x^k} \right\rfloor = \left\lfloor \frac{f}{g} \right\rfloor, \quad g x^k \left\{ \frac{x^k f}{g x^k} \right\} = x^k \left(g \left\{ \frac{f}{g} \right\} \right).$$

Случай $s > 0$ сводится к рассмотренному с помощью леммы 19. Лемма доказана.

Пусть $\Delta(n)$ обозначает любую функцию, удовлетворяющую соотношениям

$$D(n) \leq \Delta(n), \quad \frac{\Delta(n)}{\Delta(2n)} \rightarrow \frac{1}{2}. \quad \square$$

Лемма 40. Справедливо неравенство $\Delta(n) \gtrsim M(n)/4$.

Доказательство. Применим лемму 38. Тогда

$$M(n) \leq 2D(n) + D(2n) \lesssim 4\Delta(n).$$

Из леммы 39 вытекает более слабое неравенство $M(n) \leq 3D(2n)$.

Далее докажем ряд лемм о сложности вычислений с двоичными числами. \square

Лемма 41. Пусть A и B суть n -разрядные двоичные числа. Тогда

$$\left\lfloor \frac{2^{3n} A}{\lfloor 2^{3n}/B \rfloor} \right\rfloor = AB, \quad \left\lfloor \frac{2^{3n-1} A}{\lfloor 2^{3n-1}/B \rfloor} \right\rfloor = AB + a,$$

$$a = 0 \quad \text{или} \quad 1.$$

Доказательство. Заметим, что

$$AB + 1 > \frac{2^{3n} A}{\lfloor 2^{3n}/B \rfloor} \geq AB,$$

так как

$$(AB + 1) \left(\frac{2^{3n}}{B} - 1 \right) = A \cdot 2^{3n} + \frac{2^{3n}}{B} - 1 - AB > A \cdot 2^{3n}$$

в силу неравенств

$$AB \leq 2^{2n} - 1 \leq \frac{2^{3n}}{B} - 1.$$

Второе неравенство леммы доказывается аналогично. \square

Лемма 42. Справедливы неравенства

$$D(n+1) < D(n) + O(n), \quad M(n+1) < M(n) + O(n).$$

Доказательство. Второе неравенство доказано в лемме 28. Докажем первое неравенство. Пусть f — $(2n+2)$ -разрядное, а g — $(n+1)$ -разрядное числа. Положим

$$q = \left\lfloor \frac{f}{g} \right\rfloor, \quad r = f - qg, \quad q_1 = \left\lfloor \frac{\lfloor f/4 \rfloor}{\lfloor g/2 \rfloor} \right\rfloor, \quad r_1 = \left\lfloor \frac{f}{4} \right\rfloor - q_1 \left\lfloor \frac{g}{2} \right\rfloor,$$

тогда числа q_1 и $q_1 \lfloor g/2 \rfloor$ вычисляются со сложностью $D(n) + O(n)$, а значит, и число $2q_1 g$, равное $4q_1 \lfloor g/2 \rfloor$ при четном g и $4q_1 \lfloor g/2 \rfloor + 2q_1$ при нечетном g , вычисляется со сложностью $D(n) + O(n)$. Так как, согласно лемме 22,

$$q_1 = \left\lfloor \frac{\lfloor f/4 \rfloor}{\lfloor g/2 \rfloor} \right\rfloor = \left\lfloor \frac{f}{4} \left\lfloor \frac{g}{2} \right\rfloor \right\rfloor,$$

то

$$-2 < 2q_1 - q < \frac{f}{2\lfloor g/2 \rfloor} - \frac{f}{g} + 1 \leq \frac{f}{2g\lfloor g/2 \rfloor} + 1 < 5,$$

значит,

$$-2g < 2q_1 g - qg < 5g, \quad -5g < f - 2q_1 g < 3g.$$

Сравнивая число $f - 2q_1 g$ с числами $-4g, -3g, \dots, 2g$, представляем его в виде $sg + r$, где $s = -5, \dots, 2$, и со сложностью $O(n)$ находим остаток r и частное $q = 2q_1 + s$. \square

Лемма 43. Справедливо неравенство

$$M(n) \leq 2D(n) + D(2n) + O(n).$$

Доказательство. Доказательство немедленно получается, если применить лемму 41, лемму 34 и лемму 42.

Пусть $\Delta(n)$ обозначает любую функцию, удовлетворяющую соотношениям

$$D(n) \leq \Delta(n), \quad \frac{\Delta(n)}{\Delta(2n)} \rightarrow \frac{1}{2}, \quad \frac{\Delta(n)}{n} \rightarrow \infty. \quad \square$$

Лемма 44. Справедливы неравенства $\Delta(n) \geq M(n)/4$.

Доказательство. Из леммы 43 следует, что

$$M(n) \leq 2D(n) + D(2n) + O(n) \leq 4\Delta(n).$$

Из леммы 33 вытекает более слабое неравенство $M(n) \leq 3D(2n)$.

Обозначим $Q(n)$ сложность возведения n -разрядного числа в квадрат. Очевидно, что $Q(n) \leq M(n)$. Такое же обозначение введем и для сложности возведения в квадрат многочлена n -й степени. \square

Лемма 45. Для случая операций с числами справедливо неравенство

$$M(n) \leq 2Q(n) + 13n + O(1),$$

а для случая операций с многочленами справедливо неравенство

$$M(n) \leq 2Q(n) + 6n + 4.$$

Доказательство. Доказательство можно получить, используя тождество

$$ab = \frac{(a+b)^2 - (a-b)^2}{4}.$$

\square

Лемма 46. Для любого n -разрядного числа x число

$$\left\lfloor \frac{2^{4n-1}}{[2^{4n-1}/x] - [2^{4n-1}/(x+1)]} \right\rfloor - x$$

отличается от x^2 не более чем на 3.

Доказательство. Заметим, что

$$\frac{2^{4n-1}}{x(x+1)} - 1 = \frac{2^{4n-1}}{x} - \frac{2^{4n-1}}{x+1} - 1 < \left\lfloor \frac{2^{4n-1}}{x} \right\rfloor - \left\lfloor \frac{2^{4n-1}}{x+1} \right\rfloor <$$

$$< \frac{2^{4n-1}}{x} - \frac{2^{4n-1}}{x+1} + 1 = \frac{2^{4n-1}}{x(x+1)} + 1,$$

$$2^{2n-1} \leq \left\lfloor \frac{2^{4n-1}}{x} \right\rfloor - \left\lfloor \frac{2^{4n-1}}{x+1} \right\rfloor < 2^{2n+1},$$

откуда имеем

$$\frac{\frac{2^{4n-1}}{x(x+1)} + 1}{\frac{2^{4n-1}}{x(x+1)} + 1} < \frac{2^{4n-1}}{\left\lfloor \frac{2^{4n-1}}{x} \right\rfloor - \left\lfloor \frac{2^{4n-1}}{x+1} \right\rfloor} < \frac{2^{4n-1}}{\frac{2^{4n-1}}{x(x+1)} - 1},$$

значит,

$$\begin{aligned} \frac{\frac{2^{4n-1}}{x(x+1)} + 1}{\frac{2^{4n-1}}{x(x+1)} - 1} &= x(x+1) \frac{1}{1 - \epsilon_n} < x(x+1)(1 + \epsilon_n + 2\epsilon_n^2) < \\ &< x(x+1) + (x(x+1))^2 2^{1-4n} + (x(x+1))^3 2^{3-8n} < \\ &< x(x+1) + 3, \end{aligned}$$

и аналогично

$$\begin{aligned} \frac{\frac{2^{4n-1}}{x(x+1)} + 1}{\frac{2^{4n-1}}{x(x+1)} + 1} &= x(x+1) \frac{1}{1 + \epsilon_n} > x(x+1)(1 - \epsilon_n) > \\ &> x(x+1) - (x(x+1))^2 2^{1-4n} > x(x+1) - 2. \quad \square \end{aligned}$$

Лемма 47. Для любого многочлена n -й степени $f(x)$ справедливо тождество

$$f(x) = \left\lfloor \frac{x^{2n}}{\left\lfloor \frac{x^{2n}}{f} \right\rfloor} \right\rfloor.$$

Доказательство. Если $fg = x^{2n} + r$, где f, g — многочлены степени n , а r — многочлен степени меньше n , то

$$\left\lfloor \frac{x^{2n}}{f} \right\rfloor = g, \quad \left\lfloor \frac{x^{2n}}{g} \right\rfloor = f. \quad \square$$

Лемма 48. Для любого многочлена n -й степени $f(x)$ справедливо тождество

$$f^2 = \left\lfloor \frac{x^{4n}}{\lfloor x^{4n}/f \rfloor - \lfloor x^{4n}/(f+1) \rfloor} \right\rfloor - f.$$

Доказательство. Так как

$$\left\lfloor \frac{x^{4n}}{f} \right\rfloor - \left\lfloor \frac{x^{4n}}{f+1} \right\rfloor = \left\lfloor \frac{x^{4n}}{f} - \frac{x^{4n}}{f+1} \right\rfloor = \frac{x^{4n}}{f(f+1)}$$

есть многочлен степени $2n$, то согласно лемме 47

$$f^2 + f = \left\lfloor \frac{x^{4n}}{\lfloor x^{4n}/f \rfloor - \lfloor x^{4n}/(f+1) \rfloor} \right\rfloor.$$

Обозначим для краткости $R(2n, n)$ через $R(n)$. Следующее утверждение справедливо и для случая чисел, и для случая многочленов. \square

Лемма 49. Справедливо неравенство

$$Q(n) < R(4n, 2n) + 2R(4n, n) + O(n) < 3R(3n) + O(n).$$

Доказательство. Рассмотрим случай многочленов. Применим лемму 48. Вычисление $\lfloor x^{4n}/f \rfloor$ и $\lfloor x^{4n}/(f+1) \rfloor$ выполняется со сложностью $2R(4n, n) + 1$. Разность этих многочленов вычисляется со сложностью $O(n)$ и является многочленом $x^{4n}/[f(f+1)]$ степени $2n$. Вычисление $f^2 + f$ выполняется со сложностью $R(4n, 2n)$. Потом применяется лемма 39.

Числовой вариант леммы доказывается аналогично, только вместо леммы 48 используется лемма 46, а вместо леммы 30 — лемма 33. \square

Теорема 19. Функции $Q(n)$, $M(n)$, $D(n)$, $R(n)$ имеют одинаковый порядок роста как для чисел, так и для многочленов.

Доказательство. Доказательство немедленно следует из лемм 40, 44, 45, 49 и первых теорем разделов 11, 12. Без лемм 40, 44 можно обойтись, но они дают более точные оценки.

Определим теперь функцию $SQR(n)$ как сложность вычисления для любого n -разрядного числа x целой части его арифметического квадратного корня. \square

Лемма 50. Для любого целого x , $0 \leq x < 2^n$,

$$\lfloor \sqrt{2^{12n+6} + x \cdot 2^{9n+6}} \rfloor = 2^{6n+3} + x \cdot 2^{3n+2} - x^2.$$

Доказательство. Для доказательства достаточно заметить, что

$$(2^{6n+3} + x \cdot 2^{3n+2} - x^2)^2 = 2^{12n+6} + x \cdot 2^{9n+6} + x^4 - x^3 \cdot 2^{3n+3} < < 2^{12n+6} + x \cdot 2^{9n+6},$$

а также, что

$$\begin{aligned} & \left(2^{6n+3} + x \cdot 2^{3n+2} - x^2 + \frac{1}{2} \right)^2 = \\ & = 2^{12n+6} + x \cdot 2^{9n+6} + x^4 - x^3 \cdot 2^{3n+3} + 2^{6n+3} + x \cdot 2^{3n+2} - x^2 + \frac{1}{4} > \\ & > 2^{12n+6} + x \cdot 2^{9n+6} + 2^{6n+3} - x^3 \cdot 2^{3n+3} > 2^{12n+6} + x \cdot 2^{9n+6}. \end{aligned} \quad \square$$

Лемма 51. Справедливо неравенство

$$Q(n) < SQR(12n+7) + 4n.$$

Доказательство основывается на лемме 50. \square

Лемма 52. Справедливо неравенство

$$SQR(n) < 3M(2n) + O(n).$$

Доказательство. Для вычисления $\lfloor \sqrt{X} \rfloor$ представим X в виде $4^n x$, $1/4 \leq x \leq 1$, и заметим, что $\lfloor \sqrt{X} \rfloor = 2^n X_n$, где X_n — двоично-рациональное приближение \sqrt{x} с погрешностью не более 2^{-n} .

Для его вычисления применим рекуррентную последовательность

$$z_{k+1} = \frac{z_k(3 - xz_k^2)}{2}$$

и в качестве искомого приближения возьмем xz_k , такое, что

$$|z_k - x^{-1/2}| \leq 2^{-k}.$$

Если представить z_k в виде $(1 + \epsilon_k)/\sqrt{x}$, где ϵ_k — относительная погрешность приближения, то

$$\frac{1 + \epsilon_{k+1}}{\sqrt{x}} = \frac{(1 + \epsilon_k)(3 - (1 + \epsilon_k)^2)}{2\sqrt{x}},$$

откуда

$$\epsilon_{k+1} = -\frac{3\epsilon_k^2}{2} - \frac{\epsilon_k^3}{2}, \quad |\epsilon_{k+1}| < 2\epsilon_k^2$$

при достаточно малом ϵ_k . Докажем по индукции, что существует двоично-рациональная последовательность $u_k = q_k/2^K$, $K = 2^k + 4$, такая, что

$$|u_k - x^{-1/2}| \leq 2^{-K},$$

и оценим сложность ее вычисления. Очевидно, сложность вычисления z_0 не превосходит некоторой константы C . Для построения u_{k+1} сначала найдем $w_{k+1} = u_k(3 - xu_k^2)/2$ и заметим, что относительная погрешность $\epsilon_k \leq \sqrt{x} \cdot 2^{-K} = 2^{-K}$, поэтому $\epsilon_{k+1} \leq 2 \cdot \epsilon_k \leq 2^{-2K+1}$, откуда следует, что

$$|w_k - x^{-1/2}| \leq \frac{\epsilon_{k+1}}{\sqrt{x}} \leq 2^{-2K+2}.$$

Оставив в x только $m = 2^{k+1} + 5$ знаков после запятой, заменим x на x_m , тогда $|x - x_m| \leq 2^{-m}$, и для $t_{k+1} = u_k(3 - x_mu_k^2)/2$ имеем оценку погрешности

$$\begin{aligned} |t_{k+1} - x^{-1/2}| &\leq |t_{k+1} - w_{k+1}| + |w_{k+1} - x^{-1/2}| \leq \\ &\leq 2^{-m-1} + 2^{-2K+2} = 2^{-2K+3}. \end{aligned}$$

Вычисление $x_m u_k^2$ требует не более $M(K) + M(m, 2K)$ операций. Оставив в числе $3 - x_m u_k^2$ только $m - 1$ разрядов из $m + 2K$, получим число s_k , такое, что

$$|3 - x_m u_k^2 - s_k| \leq 2^{-m+1}.$$

Тогда $f_{k+1} = u_k s_k / 2$ удовлетворяет неравенству

$$\begin{aligned} |f_{k+1} - x^{-1/2}| &\leq |f_{k+1} - t_{k+1}| + |t_{k+1} - x^{-1/2}| \leq \\ &\leq 2^{-m} + 2^{-2K+3} \leq 2^{-2K+4}. \end{aligned}$$

Для вычисления f_{k+1} достаточно $M(K, m - 1) = 2M(2^k) + O(2^k)$ операций. Округлив f_{k+1} до $m - 1 = 2^{k+1} + 4$ разрядов с сохранением неравенства $|f_{k+1} - x^{-1/2}| \leq 2^{-2^{k+1}-4}$, получаем очередной член последовательности u_{k+1} . Сложность его вычисления при условии предварительного вычисления предыдущего члена последовательности u_k оценивается сверху, согласно лемме 28, как

$$M(K) + M(m, 2K) + M(K, m - 1) = 3M(2^k) + M(2^{k+1}) + O(2^k).$$

Поэтому полная сложность вычисления u_{k+1} оценивается (как мы уже не раз делали)

$$\sum_{i=0}^k 3M(2^i) + M(2^{i+1}) + O(2^i) \leq 3M(2^{k+1}) + M(2^{k+2}) + O(2^{k+1}).$$

Осталось выбрать k так, чтобы $2^k + 2 \geq n > 2^{k-1} + 2$, тогда по $K = (2^k + 4)$ -разрядной двоичной дроби u_k , такой, что

$$|u_k - x^{-1/2}| \leq 2^{-n-2},$$

находим $(2n + K)$ -разрядную дробь xu_k , такую, что

$$|\sqrt{x} - xu_k| \leq x|x^{-1/2} - u_k| \leq 2^{-n-2}.$$

Оставляя в дроби x только старшие $n + 2$ разряда, получим $(n + 2)$ -разрядную дробь x_n , такую, что $|x - x_n| \leq 2^{-n-2}$, откуда для $(n + 2 + K)$ -разрядной дроби $x_n u_k$ имеем

$$|x_n u_k - \sqrt{x}| \leq |x_n u_k - xu_k| + |xu_k - \sqrt{x}| \leq 2^{-n-2} + 2^{-n-2} = 2^{-n-1},$$

причем произведение $x_n u_k$ вычисляется со сложностью не более

$$M(n+2, K) \leq M(n+2, 2n-2) \leq 2M(n) + O(n)$$

и в то же время не более

$$M(n+2, K) \leq M(K).$$

Округляя дробь $x_n u_k$ до n разрядов, получаем нужное нам число X_n — двоично-рациональное приближение \sqrt{x} с погрешностью не более 2^{-n} . Заметим, что округление делается «бесплатно», так как нам известно, u_k больше, чем $x^{-1/2}$, или меньше (это зависит от четности или нечетности k). Поэтому полная сложность вычисления X_n оценивается как

$$4M(K) + M(2K) + O(K)$$

или как

$$2M(n) + 3M(K) + M(2K) + O(n),$$

где $K = 2^k + 2 \leq n < 2^{k+1} + 2$, а также как

$$2M(n) + 3M(2n) + M(4n) + O(n) \leq 3M(4n) + O(n).$$

Вспоминая, что в выбранном выше двоичном представлении числа X было $2n$ или $2n-1$ разряд, получаем, наконец, что для вычисления $\lfloor \sqrt{X} \rfloor$ из m -разрядного числа достаточно $3M(2m) + O(m)$ двоичных операций. \square

Теорема 20. Функция $SQR(n)$ имеет тот же порядок роста, что и функции $Q(n)$, $M(n)$, $D(n)$, $R(n)$.

Доказательство. Для доказательства достаточно применить леммы 51, 52 и теорему 19. \square

Глава 14

О сложности преобразования чисел из одной системы в другую

Сообщение на доске объявлений гласило, что он находится в комнате 423, но при взгляде на план системы нумерации, с виду логичной, складывалось впечатление, будто ее разрабатывал либо лунатик, либо математик.

Роберт Бэрнанд, 1983

(цитируется по книге Д. Кнута
«Искусство программирования»)

Обозначим $K_{b,B}(n)$ сложность преобразования n -значного числа из b -ичной системы в B -ичную. Операции выполняем в B -ичной арифметике.

Теорема 21. Если число $\beta = \log_B b$ рационально, то

$$K_{b,B}(n) = O(n).$$

Если β иррационально, то при $b < B$

$$K_{b,B}(n) \leq (2n + 1)(n - 1)\beta + o(n),$$

а при $b > B$

$$K_{b,B}(n) \leq \frac{n^2(5\beta[\beta] - \beta)}{2} + O(\beta^2 n).$$

Доказательство. Для того, чтобы перевести число $u = (u_n \dots u_1)_b$ в B -ичную систему, применим схему Горнера вычисления значения многочлена $u_n b^{b^{-1}} + \dots + u_1$, проводя все операции в B -ичной

системе. Оценка сложности $O(n^2)$ для этого алгоритма очевидна. Получим более точные оценки. Пусть β рационально, тогда для некоторых натуральных c и C имеем $B^C = b^c$. Перевод из b -ичной системы в B -ичную представляется в виде композиции перевода из b -ичной системы в b^c -ичную и перевода из B -ичной системы в B^C -ичную, которые выполняются со сложностью $O(n)$ каждый.

Пусть β иррационально и $b < B$. Положим $x_k = (u_n \dots u_{n-k+1})_b$, тогда

$$x_{k+1} = x_k b + u_{n-k}, \quad x_n = u,$$

и в силу неравенства

$$b^k - 1 \leq B^K - 1, \quad K = \lceil k\beta \rceil,$$

длина числа x в B -ичной системе не больше K . Предполагаем, что множество b -ичных цифр «бесплатно» разыскивается во множестве B -ичных цифр. Тогда вычисление x в B -ичной арифметике выполняется со сложностью не больше

$$3K - 2 + \lceil (k+1)\beta \rceil = (4k+1)\beta + 2 - 3\{k\beta\} - \{(k+1)\beta\},$$

где знак $\{x\} = x - \lfloor x \rfloor$ — символ дробной части числа x . Значит, сложность вычисления $u = x_n$ не выше

$$\begin{aligned} & \sum_{k=1}^{n-1} ((4k+1)\beta + 2 - 3\{k\beta\} - \{(k+1)\beta\}) = \\ & = (n-1)(2n+1)\beta + 2(n-1) - 4 \sum_{k=1}^{n-1} \{k\beta\} - \{n\beta\} + \{\beta\}. \end{aligned}$$

Так как β иррационально, то последовательность $\{n\beta\}$ равномерно распределена на отрезке $[0, 1]$ согласно теореме Вейля—Серпинского, значит,

$$\sum_{k=1}^{n-1} \{k\beta\} = \frac{n-1}{2} + o(n),$$

откуда и следует оценка теоремы. □

Пусть $b > B$. Тогда для нахождения B -ичных записей чисел u_i требуется сложность $O(n)$. Ее можно найти с помощью такого алгоритма:

$$u = (U_k \dots U_1)_B,$$

где

$$U_1 = u \bmod B, \quad U_2 = \left\lfloor \frac{U_1}{B} \right\rfloor \bmod B, \quad \dots, \quad U_k = \left\lfloor \frac{U_{k-1}}{B} \right\rfloor \bmod B,$$

$x \bmod B$ означает $x - B[x/B]$ — остаток от деления x на B . Этот алгоритм, кстати, можно использовать для перевода не только одnorазрядных, но и произвольных чисел из одной системы в другую. Можно оценить его сложность более точно, если учесть то, как заданы числа u_i . Если, например, они заданы в двоичном виде, число B тоже, и операции деления проводятся в двоичной арифметике, то получается оценка $O(\beta^2 n)$. Далее, для вычисления $u = x_n$ с помощью указанной выше схемы Горнера требуется, согласно результатам раздела 2, сложность не выше

$$\begin{aligned} & \sum_{k=1}^{n-1} (M(\lceil k\beta \rceil, \lceil \beta \rceil) + \lceil (k+1)\beta \rceil + \lfloor \beta \rfloor) \leq \\ & \leq \sum_{k=1}^{n-1} (5\lceil k\beta \rceil \lceil \beta \rceil - 2\lceil k\beta \rceil + \lceil (k+1)\beta \rceil - 3), \end{aligned}$$

так как неравенство $b \leq B^\beta - 1$ равносильно неравенству $b \leq B^\beta$. Так как

$$\begin{aligned} & 5\lceil k\beta \rceil \lceil \beta \rceil - 2\lceil k\beta \rceil + \lceil (k+1)\beta \rceil - 3 = \\ & = 5(\beta \lceil \beta \rceil k + \lceil \beta \rceil - \{k\beta\} \lceil \beta \rceil) - (k-1)\beta - \{(k+1)\beta\} + 2\{k\beta\} - 4, \end{aligned}$$

то, используя равенство

$$\sum_{k=1}^{n-1} k\beta = \frac{n-1}{2} + o(n),$$

получаем оценку

$$5\beta \lceil \beta \rceil \frac{n(n-1)}{2} - \frac{\beta(n-2)(n-1)}{2} + \left(\frac{5\lceil \beta \rceil}{2} - 4 + \frac{1}{2} \right) (n-1).$$

Полученные оценки сложности несколько завышены, так как в них учтены, например, все переносы, реально в ручных вычислениях выполняющиеся «в уме». При $b = B + 1$ оценку теоремы 21 можно улучшить. А именно, тогда можно считать, что перевод чисел u_i в B -ичную систему осуществляется «бесплатно», а вычисление

$$x_{k+1} = x_k b + u_{n-k} = x_k(B + 1) + u_{n-k} = x_k B + u_{n-k} + x_k$$

в B -ичной арифметике осуществляется со сложностью не выше

$$2\lceil k\beta \rceil + \lceil (k+1)\beta \rceil - 1.$$

Поэтому общая оценка сложности имеет вид

$$\sum_{k=1}^{n-1} (2\lceil k\beta \rceil + \lceil (k+1)\beta \rceil - 1) = \frac{3\beta n(n-1)}{2} + \frac{n}{2} + o(n).$$

В случае $b = B - 1$ вычисление

$$x_{k+1} = x_k b + u_{n-k} = x_k(B - 1) + u_{n-k} = x_k B + u_{n-k} - x_k$$

в B -ичной арифметике осуществляется со сложностью не выше $2\lceil k\beta \rceil + 1$. Поэтому общая оценка сложности имеет вид

$$\sum_{k=1}^{n-1} (2\lceil k\beta \rceil + 1) = \beta n(n-1) + 2n + o(n).$$

Так как оценка сложности перевода при $b > B$ получилась выше, то в случае $b = b_0^c$, где $b < B$, а c — натуральное число, лучше переводить из b -ичной в b_0 -ичную систему со сложностью $O(n)$, а потом из b_0 -ичной в B -ичную систему, и пользоваться оценкой

$$K_{b,B}(n) \leq O(n) + K_{b_0,B}(n).$$

Этой оценкой можно пользоваться и при $b_0 = B \pm 1$. Так, например для перевода из троичной в десятичную системы, скажем,

числа $(1220102)_3$, сначала переводим его в девятеричную систему и получаем число $(1812)_9$, а потом, проводя вычисления в десятичной арифметике, на k -м шаге из десятичного числа, записанного первыми $k + 1$ цифрами, отнимаем десятичное число, записанное первыми k цифрами, и записываем цифры разности вместо первых $k + 1$ цифр, и проводя эти операции для $k = 1, 2, \dots, n - 1$, получаем в итоге десятичную запись исходного числа:

$$\begin{array}{r} 1.812 \\ - 1 \\ \hline 17.12 \\ - 17 \\ \hline 154.2 \\ - 154 \\ \hline 1388 \end{array}$$

Оценка сложности имеет вид

$$K_{3,10}(n) \leq n + K_{9,10}(n) \leq (\log_{10} 9)n(n - 1) + 3n + o(n).$$

Для проверки выполним обратный перевод. Чтобы перевести десятичное число в троичное, сначала переведем его в девятеричное. Для этого применяем подобный же алгоритм, но на k -м шаге будем не отнимать, а прибавлять, учитывая формулу

$$x_{k+1} = x_k b + u_{n-k} = x_k B + u_{n-k} + x_k,$$

и операции будем выполнять на сей раз в девятеричной арифметике.

Вычисления выглядят так (точки, как и ранее, ставятся с целью избежать ошибок):

$$\begin{array}{r} 1.388 \\ + 1 \\ \hline 14.88 \\ + 14 \\ \hline 163.8 \\ + 163 \\ \hline 1812 \end{array}$$

Для перевода в троичную запись достаточно каждую девятеричную цифру заменить на ее троичную запись, которая получается путем деления на 3 с остатком. В итоге оценка сложности перевода имеет вид

$$K_{10,3}(n) \leq \left\lceil \frac{n}{\log_{10} 9} \right\rceil + K_{10,9}(n) \leq \frac{3n(n-1)}{2 \log_{10} 9} + \left(\frac{1}{2} + \frac{1}{\log_{10} 9} \right) n + o(n).$$

Рассмотрим теперь более практически важный случай перевода из двоичной системы в десятичную. Напомним уже рассматривавшийся для этого алгоритм (см. «Схема Горнера и перевод из одной позиционной системы в другую»¹⁾). Сначала переведем число из двоичной системы в восьмеричную. Для этого разбиваем справа налево его цифры на «тройки» (последняя «тройка» на самом деле может быть «двойкой» или одной цифрой) и переводим их в восьмеричную систему схемой Горнера (выполняемой устно). Например,

$$(1111110000)_2 = (1.111.110.000)_2 = (1760)_8.$$

Сложность этого перевода не выше $4n/3$ (при выполнении на компьютере по существу она нулевая). Выполним перевод из восьмеричной системы в десятичную. Пусть

$$u = (u_n \dots u_1)_8.$$

На k -м шаге выполняем над полученной на предыдущем шаге записью в десятичной арифметике действия

$$\overline{u_n \dots u_{n-k-1}} - 2\overline{u_n \dots u_{n-k}} = \overline{v_n \dots v_{n-k-1}}$$

и получаем запись

$$v_n \dots v_{n-k-1}.u_{n-k-2} \dots u_1$$

(старшие разряды могут оказаться нулевыми и в ручных вычислениях участвовать не будут). На $(n-1)$ -шаге получаем десятичную

¹⁾ Гашков С. Б. Занимательная компьютерная арифметика. М.: URSS, 2012.

запись числа u . Например,

$$\begin{array}{r} 1.760 \\ - 2 \\ \hline 15.60 \\ - 30 \\ \hline 126.0 \\ - 252 \\ \hline 1008 \end{array}$$

Указанный алгоритм удобен в применении, однако вариант алгоритма, использованный в доказательстве теоремы 21, дает лучшую оценку сложности, именно

$$K_{2,10}(n) \leq (\log_{10} 8)(n(n-1)) + \frac{n}{2} + o(n).$$

Действительно, в случае $b = 2$ вычисление

$$x_{k+1} = 2x_k + u_{n-k}$$

в B -ичной арифметике осуществляется со сложностью не выше

$$\lceil (k+1)\beta \rceil + 2\lceil k\beta \rceil - 1.$$

Поэтому общая оценка сложности имеет вид

$$\sum_{k=1}^{n-1} (2\lceil k\beta \rceil + \lceil (k+1)\beta \rceil - 1) = \frac{3\beta n(n-1)}{2} + \frac{n}{2} + o(n).$$

Алгоритм перевода из десятичной системы в двоичную почти такой же. Сначала переводим в восьмеричную запись. Для этого, пользуясь восьмеричной арифметикой, на k -м шаге выполняем над полученной на предыдущем шаге записью действия:

$$\overline{u_n \dots u_{n-k-1}} + 2\overline{u_n \dots u_{n-k}} = \overline{v_{n+1} \dots v_{n-k-1}}$$

и получаем запись

$$\overline{v_{n+1} \dots v_{n-k-1} . u_{n-k-2} \dots u_1}$$

(поначалу $(n + 1)$ -е разряды окажутся нулевыми и в ручных вычислениях участвовать не будут). На $(n - 1)$ -м шаге получаем восьмеричную запись числа u . Например,

$$\begin{array}{r} + 1.945 \\ + \underline{2} \\ 23.45 \\ + 46 \\ \underline{302.5} \\ + 604 \\ \underline{3631} \end{array}$$

Далее со сложностью не больше $2n$ переводим восьмеричное n -значное число в двоичное (вычисляя для каждой восьмеричной цифры двумя делениями на 2 с остатком ее двоичную запись). Подобно теореме 21 получаем (несколько более точную чем вытекающая из нее) оценку сложности перевода

$$K_{10,2}(n) \leq 2 \left\lceil \frac{n}{\log_{10} 8} \right\rceil + K_{10,8}(n) \leq \frac{5n(n-1)}{2\log_{10} 8} + \left(\frac{5}{2} + \frac{2}{\log_{10} 8} \right) n + o(n).$$

Действительно, в случае $b = B + 2$ вычисление

$$x_{k+1} = x_k b + u_{n-k} = x_k (B + 2) + u_{n-k} = x_k B + u_{n-k} + 2x_k$$

в B -ичной арифметике осуществляется со сложностью не выше $4\lceil k\beta \rceil + \lceil (k+1)\beta \rceil$ (требуется не более $\lceil k\beta \rceil$ переносов при вычислении $x_k B + u_{n-k}$, со сложностью не более $\lceil k\beta \rceil - 1$ вычисляется $2x_k$ путем сложения и получается не более чем $(\lceil k\beta \rceil + 1)$ -разрядное число, со сложностью не более $\lceil k\beta \rceil + \lceil (k+1)\beta \rceil$ выполняется сложение $x_k B + u_{n-k} + 2x_k$, с единичной сложностью вычисляется B -ичная запись числа u_{n-k}), поэтому общая оценка сложности имеет вид

$$\sum_{k=1}^{n-1} (4\lceil k\beta \rceil + \lceil (k+1)\beta \rceil) = 5\beta \frac{n(n-1)}{2} + \frac{5n}{2} + o(n).$$

Алгоритмы перевода, приведенные выше, хорошо работают для небольших чисел. Однако известен асимптотически более быстрый алгоритм. Пусть $M(n)$ — любая верхняя оценка сложности умножения n -разрядных чисел, удовлетворяющая условию

$$M(n + m) \geq M(n) + M(m).$$



Арнольд Шёнхаге

Например, в качестве такой функции можно взять cn^a , если $a \geq 1$.

Теорема 22 (теорема Шёнхаге). Если

$$\beta = \log_B b$$

иррационально, то

$$K_{b,B}(n) \leq 2M\left(\left\lfloor \frac{n\beta}{2} \right\rfloor\right) \log_2 n + C_0 n \log_2 n + C_1 n,$$

где C_0 и C_1 — константы.

Доказательство. Сначала докажем несколько лемм.

Лемма 53. Справедливо неравенство

$$M(n + 1) \leq M(n) + 12n + 1.$$

Предлагаем эту лемму читателю доказать самостоятельно.

Лемма 54. Справедливы неравенства

$$\begin{aligned} M(n, n + k) &\leq M(n) + M(n, k) + 3n - 1 \leq \\ &\leq M(n) + \frac{nM(k)}{k} + M(k) + 6n, \\ M(n, n + 1) &\leq M(n) + 6n - 3. \end{aligned}$$

Доказательство. Второе неравенство следует из первого и теоремы 12. Последнее неравенство следует из первого и леммы из раздела 2.

Обозначим $n_0 = \lfloor n/2 \rfloor$, $n_1 = \lceil n/2 \rceil$,

$$n = (n_{\alpha_1, \dots, \alpha_{k-1}})_{\alpha_k}, \quad \alpha_i = 0, 1. \quad \square$$

Лемма 55. Справедливы равенство и неравенства

$$\sum_{\alpha_1=0,1, \dots, \alpha_k=0,1} n_{\alpha_1, \dots, \alpha_k} = n, \quad \left\lfloor \frac{n}{2^k} \right\rfloor \leq n_{\alpha_1, \dots, \alpha_k} \leq \left\lceil \frac{n}{2^k} \right\rceil.$$

Доказательство. Применим индукцию по k и проверим, что

$$\begin{aligned} n_{0, \dots, 0} &= \left\lfloor \frac{\lfloor \dots \lfloor \lfloor n/2 \rfloor / 2 \rfloor \dots \rfloor}{2} \right\rfloor = \left\lfloor \frac{n}{2^k} \right\rfloor, \\ n_{1, \dots, 1} &= \left\lceil \frac{\lceil \dots \lceil \lceil n/2 \rceil / 2 \rceil \dots \rceil}{2} \right\rceil = \left\lceil \frac{n}{2^k} \right\rceil, \\ n_{0, \dots, 0} &\leq n_{\alpha_1, \dots, \alpha_k} \leq n_{1, \dots, 1}, \\ \sum_{\alpha_1=0,1, \dots, \alpha_k=0,1} n_{\alpha_1, \dots, \alpha_k} &= \sum_{\alpha_1=0,1, \dots, \alpha_{k-1}=0,1} \sum_{\alpha_k=0}^1 (n_{\alpha_1, \dots, \alpha_{k-1}})_{\alpha_k} = \\ &= \sum_{\alpha_1=0,1, \dots, \alpha_{k-1}=0,1} n_{\alpha_1, \dots, \alpha_{k-1}} = \dots = \sum_{\alpha_1=0}^1 n_{\alpha_1} = n_0 + n_1 = n. \quad \square \end{aligned}$$

Следующая лемма является основной в доказательстве теоремы.

Лемма 56. Справедливо неравенство

$$K_{b,B}(n) \leq K_{b,B}(n) + K_{b,B}(n) + 2M\left(\left\lceil \frac{n\beta}{2} \right\rceil\right) + K_0 n + K_1,$$

где при $b < B$ константы $K_0 = 25,5\beta$, $K_1 = 10$, а при $b > B$

$$K_0 = 13,5\beta + \frac{2M(\lceil \beta \rceil)}{\lceil \beta \rceil} + 12, \quad K_1 = \frac{2M(\lceil \beta \rceil)}{\lceil \beta \rceil} + 12 + 2M(\lceil \beta \rceil).$$

Доказательство. Обозначим $K(n)$ сложность перевода n -значного b -ичного числа в B -ичную систему и одновременного вычисления в этой системе числа b^n . Пусть $u = (u_n \dots u_1)_b$. Представим его в виде

$$u = U_1 b^{n_0} + U_0,$$

где

$$U_1 = (u_n \dots u_{n_0+1})_b, \quad U_0 = (u_{n_0} \dots u_1)_b.$$

Со сложностью $K(n_0) + K(n_1)$ можно перевести числа U_i в B -ичную систему и вычислить в ней числа b^{n_0} и b^{n_1} . После этого вычисление $b^n = b^{n_0} b^{n_1}$ и $u = U_1 b^{n_0} + U_0$ выполняется со сложностью

$$2M(\lceil n_0 \beta \rceil, \lceil n_1 \beta \rceil) + \lceil n \beta \rceil + \lceil n_0 \beta \rceil - 1,$$

так как при иррациональном β число B -ичных разрядов в числах U_0 , b^{n_0} не превосходит $\lceil n_1 \beta \rceil$, в числах U_1 , b^{n_1} не превосходит $\lceil n_1 \beta \rceil$ и в числе u не превосходит $\lceil n \beta \rceil$. Так как $\lceil n_1 \beta \rceil - \lceil n_0 \beta \rceil \leq \lceil \beta \rceil$, то согласно леммам 53, 54

$$M(\lceil n_0 \beta \rceil, \lceil n_1 \beta \rceil) \leq M(\lceil n_0 \beta \rceil) + \left(\frac{M(\lceil \beta \rceil)}{\lceil \beta \rceil} + 6 \right) (\lceil n_0 \beta \rceil + 1) + M(\lceil \beta \rceil),$$

$$M(\lceil n_0 \beta \rceil) \leq M(\lfloor n_0 \beta \rfloor) + 12 \lfloor n_0 \beta \rfloor + 1,$$

а при $\beta < 1$

$$M(\lceil n_0 \beta \rceil, \lceil n_1 \beta \rceil) \leq M(\lceil n_0 \beta \rceil) + 18 \lfloor n_0 \beta \rfloor + 4. \quad \square$$

Лемма 57. Справедливы неравенства

$$M(x_1 + \dots + x_n) \geq M(x_1) + \dots + M(x_n),$$

$$\lfloor x_1 + \dots + x_n \rfloor \geq \lfloor x_1 \rfloor + \dots + \lfloor x_n \rfloor.$$

Доказательство. Применим индукцию по n и проверим, что

$$\begin{aligned} M(x_1 + \dots + x_n) &\geq M(x_1 + \dots + x_{n-1}) + M(x_n) \geq \\ &\geq M(x_1) + M(x_2) + \dots + M(x_n). \end{aligned}$$

Аналогично доказываем второе неравенство, предварительно проверив, что $\lfloor x_1 + x_2 \rfloor \geq \lfloor x_1 \rfloor + \lfloor x_2 \rfloor$. Лемма доказана. \square

Докажем теперь теорему. Применяя индукцию и леммы 55, 56, 57, получаем

$$\begin{aligned} K_{b,B}(n) &\leq 2^{k+1} K_{b,B} \left(\left\lceil \frac{n}{2^{k+1}} \right\rceil \right) + 2M \left(\left\lfloor \frac{n\beta}{2} \right\rfloor \right) + \dots + \\ &+ \sum_{\alpha_1=0,1,\dots,\alpha_k=0,1} 2M \left(\left\lfloor \frac{n_{\alpha_1,\dots,\alpha_k}\beta}{2} \right\rfloor \right) + \\ &+ K_0 n(k+1) + K_1(1+2+\dots+2^k) \leq 2^{k+1} K_{b,B} \left(\left\lfloor \frac{n}{2^{k+1}} \right\rfloor \right) + \\ &+ 2M \left(\left\lfloor \frac{n\beta}{2} \right\rfloor \right) (k+1) + K_0 n(k+1) + K_1 2^{k+1}. \end{aligned}$$

Полагая $k+1 = \lfloor \log_2 n \rfloor$, отсюда выводим, что

$$K_{b,B}(n) \leq 2M \left(\left\lfloor \frac{n\beta}{2} \right\rfloor \right) \log_2 n + K_0 n \log_2 n + (K_1 + K_{b,B}(2))n.$$

Можно проверить, что при $b > B$

$$K_{b,B}(2) \leq 2K_{b,B}(1) + 2M(\lceil \beta \rceil) + 3\lceil \beta \rceil - 1, \quad K_{b,B}(1) \leq O(\log_2 b)^2,$$

а при $b < B$ $K_{b,B}(2) = 2$. Теорема доказана. \square

Глава 15

Модулярная арифметика и китайская теорема об остатках

Частный случай этой теоремы был сформулирован китайским математиком Сунь Цю, который предложил правило «тай йен» («большое обобщение»). Дата написания его работы точно не установлена; предположительно между 280 и 473 гг. н. э.

Д. Кнут. Искусство программирования

Будем здесь и далее обозначать остаток от деления числа a на m через $a \bmod m$. Остатки от деления на m суммы и произведения двух чисел удобно вычислять по следующим формулам:

$$a + b \bmod m = a \bmod m \oplus b \bmod m,$$

$$ab \bmod m = a \bmod m \circ b \bmod m,$$

где через $x \oplus y$ обозначается операция $x + y \bmod m$, а через $x \circ y$ обозначается операция $xy \bmod m$. Частные случаи этих формул при $m = 2$ известны любому младшекласснику: сумма четных чисел четна так же, как и сумма двух нечетных чисел, сумма чисел разной четности нечетна, произведение нечетных чисел нечетно, а произведение четного числа на любое число всегда четно. Вычисления по модулю 7 позволяют легко решать задачи о календаре. Для разминки предлагаем читателю пару таких задач.

Задача 30. В какой день недели начнется XXII век?

Указание. Учтите, что XXII век начинается 1 января 2101 года, а не 2100.

Задача 31. Вы решили не выбрасывать старый настенный календарь. Через сколько лет он опять сможет Вам пригодиться?

Указание. Не забудьте, что календарь мог быть для високосного года.

В качестве более сложного примера приведем формулу Гаусса для Пасхалии. В n -й год от Рождества Христова западные церкви празднуют Пасху $(22 + d + e)$ марта или $(d + e - 9)$ апреля, где $d = (19a + M) \bmod 30$, $e = (2b + 4c + 6d + N) \bmod 7$, $a = n \bmod 19$, $b = n \bmod 4$, $c = n \bmod 7$, $M = 23$, $N = 3$ при $n = 17**$, $M = 23$, $N = 4$ при $n = 18**$, $M = 24$, $N = 5$ при $n = 19**$, $M = 24$, $N = 6$ при $n = 20**$, $M = 24$, $N = 7$ при $n = 21**$.

Задача 32. Вычислите, когда будет Пасха в текущем году.

О том, как быстро вычислять остатки от больших чисел по малым модулям, фактически шла речь в разделе, посвященном признакам делимости. По остатку от деления произвольного числа на mn можно однозначно определить остатки от его деления на m и n . Китайская теорема об остатках утверждает, что если $(m, n) = 1$, то по остаткам от деления на m и n можно однозначно восстановить остаток от деления на mn , причем всегда найдется число, имеющее заданные остатки от деления на числа m и n . Сунь Цю, живший около 2000 лет назад, сформулировал частный случай этой теоремы. Прежде чем приступить к ее изучению, решите следующую задачу.

Задача 33. Найдите наименьшее натуральное число, которое при делении на n дает остаток $n - 1$, а при делении на $n + 1$ — остаток n . Мы рассмотрим несколько доказательств китайской теоремы.

Указание. Прибавьте 1.

Начнем с неконструктивного доказательства. Заметим, что если два числа имеют разные остатки от деления на mn , то они имеют либо разные остатки от деления на m , либо разные остатки от деления на n . Действительно, в противном случае разность

этих чисел делится на m и на n , а значит, и на mn (ведь m и n взаимно просты), но это невозможно, так как они имеют разные остатки при делении на mn . Но разных упорядоченных пар остатков от деления на m и n имеется ровно mn штук, и столько же имеется различных остатков от деления на mn . Значит, по остаткам от деления на m и n можно однозначно восстановить остаток от деления на mn , причем всегда найдется число, имеющее заданные остатки от деления на числа m и n .

Теперь приведем конструктивное доказательство, причем в более общем случае. А именно, докажем, что для любых попарно взаимно простых чисел m_i , $1 \leq i \leq r$, и любых чисел n_i , $0 \leq n_i < m_i$, найдется единственное число N , $0 \leq N < M = m_1 \dots m_r$, дающее при делении на m_i остаток n_i .

Действительно, пусть M_i — произведение всех этих чисел, кроме m_i . Найдется такое натуральное число a_i , что $a_i M_i$ при делении на m_i дает в остатке 1. Это число быстрее всего можно найти с помощью алгоритма Евклида, но можно также довольно быстро вычислить по явной формуле

$$a_i = M_i^{\phi(m_i)-1} \bmod m_i,$$

где $\phi(m)$ — функция Эйлера. Действительно, согласно теореме Эйлера

$$a_i M_i \bmod m_i = M_i^{\phi(m_i)} \bmod m_i = 1.$$

Заметим теперь, что число

$$\sum_{i=1}^r n_i a_i M_i \bmod M$$

при делении на m_i дает остаток n_i . Действительно, слагаемое $n_i a_i M_i$ делится на M_i , а значит, и на любое m_j при $j \neq i$, а при делении на m_i дает остаток n_i . Поэтому сумма при делении на m_i дает остаток n_i , что и доказывает существование числа, имеющего заданные остатки по модулям m_i . Единственность такого числа по модулю $M = m_1 \dots m_r$ (т. е. однозначность определения его остатка при делении на M) вытекает из следующего.

Различных упорядоченных наборов остатков при делении на m_i имеется ровно M штук, и каждому из них соответствует хотя бы один из остатков по модулю M (как только что было доказано), а значит, и ровно один из этих остатков. Действительно, в противном случае общее число разных наборов остатков при делении на m_i , соответствующих разным остаткам по модулю M , было бы меньше M , и для какого-то набора остатков по модулю m_i не нашлось бы числа, соответствующего этому набору, что невозможно, как было уже доказано.

Сделаем замечание об организации вычислений. При вычислении любой из указанных сумм не следует сначала производить все арифметические действия и только в конце вычислять остаток по модулю M , так как это приведет к громоздким операциям с большими числами. Надо каждую из арифметических операций заменять на соответствующую операцию по модулю M , как было указано в начале раздела, тогда получится тот же результат, но гораздо быстрее, причем при вычислениях никогда не будут появляться числа, большие M . При использовании для вычислений суммы

$$\sum_{i=1}^r n_i M_i^{\phi(m_i)} \bmod M = \sum_{i=1}^r n_i a_i M_i \bmod M$$

следует возведение в степень проводить при помощи быстрого (например, бинарного) алгоритма. Чтобы им воспользоваться, однако, надо заранее знать показатели степеней, а для этого нужно уметь вычислять значение функции Эйлера. К сожалению, быстро ее можно вычислить, только если заранее известно разложение ее аргумента на простые множители. Приведем эту формулу Эйлера: если $p_1^{\alpha_1} \dots p_n^{\alpha_n}$ — разложение числа m на простые множители, то

$$\begin{aligned} \phi(m) &= (p_1^{\alpha_1} - p_1^{\alpha_1-1}) \dots (p_n^{\alpha_n} - p_n^{\alpha_n-1}) = \\ &= m \left(1 - \frac{1}{p_1}\right) \dots \left(1 - \frac{1}{p_n}\right). \end{aligned}$$

Для ее доказательства достаточно применить следующее, так называемое мультипликативное свойство функции Эйлера: при вза-

имно простых m и n справедливо равенство

$$\phi(mn) = \phi(m)\phi(n),$$

и воспользоваться частным случаем указанной формулы, который предлагается в виде задачи.

Задача 34. Докажите, что при простом p

$$\phi(p^n) = p^n - p^{n-1}.$$

Указание. Число взаимно просто с p^n тогда и только тогда, когда оно не кратно p .

Для доказательства мультипликативного свойства заметим, что при взаимно простых m и n остаток от деления на mn взаимно прост с mn тогда и только тогда, когда соответствующие ему (согласно китайской теореме) остатки от деления на m и n взаимно просты с m и n соответственно, и поэтому $\phi(mn)$, равное по определению количеству взаимно простых с mn остатков по модулю mn , равно количеству всех различных упорядоченных пар остатков по модулям m и n , взаимно простых со своими модулями, т. е. равно $\phi(m)\phi(n)$. Остается доказать сформулированное утверждение «тогда и только тогда». Действительно, если число взаимно просто с mn , то его остаток, например, по модулю m , взаимно прост с модулем, так как иначе их общий делитель, скажем d , будет общим делителем нашего числа и mn , что невозможно. Если же это число не взаимно просто с mn , то оно имеет нетривиальный общий делитель, например, с числом n , и значит, его остаток по модулю n будет иметь с ним тот же общий делитель, поэтому, если остатки нашего числа по модулям m и n взаимно просты со своими модулями, то наше число взаимно просто с mn .

Задача 35. Докажите равенство

$$\phi(m^k) = m^{k-1}\phi(m).$$

Указание. Разложите m на простые множители и примените формулу Эйлера.

Недостаток приведенного алгоритма вычисления по заданным остаткам n_i по $\text{mod } m_i$ числа $\langle n_1, \dots, n_r \rangle \text{ mod } M$ в том, что числа M_i могут быть очень велики по сравнению с m_i . Этого недостатка лишен следующий способ вычисления функции $\langle n_1, \dots, n_r \rangle \text{ mod } M$ (принадлежащий Х. Гарнеру). Он основан на использовании чисел c_{ij} , таких, что $c_{ij}m_i \text{ mod } m_j = 1$. Эти числа, причем не превосходящие по величине m_j , можно заранее вычислить с помощью алгоритма Евклида (или теоремы Эйлера). Далее рекурсивно вычисляем

$$u_1 = n_1,$$

$$u_2 = (n_2 - u_1)c_{1,2} \text{ mod } m_2,$$

$$u_3 = ((n_3 - u_1)c_{1,3} - u_2)c_{2,3} \text{ mod } m_3,$$

.....

$$u_r = (\dots ((n_r - u_1)c_{1,r} - u_2)c_{2,r} - \dots - u_{r-1})c_{r-1,r} \text{ mod } m_r.$$

Операции выполняем модулярные, поэтому все возникающие в процессе вычислений числа не превосходят

$$m = \max m_i.$$

Тогда справедливо равенство

$$\langle n_1, \dots, n_r \rangle \text{ mod } M = u_r m_{r-1} \dots m_1 + \dots + u_3 m_2 m_1 + u_2 m_1 + u_1.$$

Действительно, сумма неотрицательна и меньше M , так как

$$u_2 m_1 + u_1 < (m_2 - 1)m_1 + m_1 = m_2 m_1,$$

$$u_3 m_2 m_1 + u_2 m_1 + u_1 \leq (m_3 - 1)m_2 m_1 + m_2 m_1 = m_3 m_2 m_1,$$

.....

$$u_r m_{r-1} \dots m_1 + \dots + u_3 m_2 m_1 + u_2 m_1 + u_1 <$$

$$< (m_r - 1)m_{r-1} \dots m_1 + m_{r-1} \dots m_1 = m_r m_{r-1} \dots m_1 = M.$$

Остаток от ее деления на m_i равен остатку от деления на m_i числа

$$u_i m_{i-1} \dots m_1 + \dots + u_3 m_2 m_1 + u_2 m_1 + u_1,$$

а он равен n_i , в чем можно убедиться по индукции, так как

$$\begin{aligned}
 u_i m_{i-1} \dots m_1 \bmod m_i &= \\
 &= (\dots ((n_i - u_1) c_{1,i} - u_2) c_{2,i} - \dots - u_{i-1}) c_{i-1,i} m_{i-1} \dots m_1 \bmod m_i = \\
 &= (\dots ((n_i - u_1) c_{1,i} - u_2) c_{2,i} - \dots - u_{i-1}) m_{i-2} \dots m_1 \bmod m_i = \\
 &= (\dots ((n_i - u_1) c_{1,i} - u_2) c_{2,i} - \dots - u_{i-2}) c_{i-2,i} m_{i-2} \dots m_1 - \\
 &\quad - u_{i-1} m_{i-2} \dots m_1 \bmod m_i = \\
 &= n_i - u_1 - u_2 m_1 - u_3 m_2 m_1 - \dots - u_{i-1} m_{i-2} \dots m_1 \bmod m_i.
 \end{aligned}$$

В последнее время китайская теорема об остатках нашла интересные применения в криптологии — науке об организации хранения и передачи секретной информации. Расскажем об одном из простейших таких применений. Допустим, что Вам поручили организовать доступ к секретной информации на компьютере для трех сотрудников банка так, чтобы они, только собравшись втроем, смогли получить ее. Каждому Вы сообщаете ключ — некоторое большое (например, стозначное) число, которое он хранит в секрете от остальных. Паролем, который открывает секретную информацию, является неизвестное никому из них число, которое однако можно быстро определить (разумеется, с помощью компьютера) по трем упомянутым ключам. В случае же отсутствия одного из них для определения пароля придется, даже зная алгоритм определения пароля по ключам, перебрать $9 \cdot 10^{99}$ вариантов возможного значения неизвестного ключа, а это пока не доступно даже суперкомпьютерам. Кроме того, для надежности вы можете периодически менять ключи. Это смогут сделать даже сами работники банка, договорившись только о размерах ключей, чтобы гарантировать их отличие друг от друга и сохранить пароль в тайне даже от Вас.

Задача 36. Предложите алгоритм определения пароля по ключам, основанный на применении китайской теоремы об остатках.

Указание. Выберем 3 больших попарно взаимно простых числа, например числа Ферма

$$a_1 = 2^{2^n} + 1, \quad a_2 = 2^{2^{n+1}} + 1, \quad a_3 = 2^{2^{n+2}} + 1.$$

Пусть i -й «компаньон» в качестве ключа выбирает произвольное число k_i в пределах от 1 до $a_i - 1$. Тогда, собравшись вместе, они могут выработать пароль k , взяв в качестве него такое число $a = a_1 a_2 a_3$ в пределах от 1 до $a - 1$, которое при делении на a_i дает в остатке k_i . Как было показано выше, число k можно вычислить по формуле

$$k = k' - \left\lfloor \frac{k'}{a} \right\rfloor a,$$

где

$$k' = k_1 n_1 + k_2 n_2 + k_3 n_3, \quad n_i m_i - a_i q_i = 1, \quad m_i = a_j a_k, \quad j \neq k \neq i \neq j.$$

Для вычисления n можно применить алгоритм Евклида. Нетрудно написать программу для компьютера, которая будет вычислять пароль k после того, как «компаньоны», таясь друг от друга, введут свои ключи. Положим $N = 2^n$. Если двое в тайне от третьего захотят найти ключ k , то им придется перебрать все возможные варианты выбора третьего ключа (потому что разным способам выбора ключей соответствуют разные значения k), т. е. перебрать не менее 2^N вариантов. С помощью результатов раздела 12 и оценки числа шагов алгоритма Евклида (указанной в разделе «Схема Горнера и перевод из одной позиционной системы в другую»¹⁾) можно оценить сложность алгоритма вычисления общего ключа по порядку как $N \cdot M(N)$, где $M(N)$ — сложность умножения N -разрядных чисел. Применение метода умножения Карацубы дает оценку $O(N^{\log_2 3})$. Поэтому даже при больших N вычисление общего ключа можно провести достаточно быстро.

Для применения модулярной арифметики в криптографических алгоритмах нужно иметь быстрые алгоритмы для перехода от системы остатков r_i по взаимно простым модулям m_i , $1 \leq i \leq n$, к остатку r по модулю $m = m_1 \dots m_n$ и обратно для очень больших чисел m_i . Обратный переход требует выполнения деления числа r на числа m_i с остатком.

Если предположить, что числа m_i являются N -разрядными, то сложность стандартного алгоритма для этого равна по порядку $n^2 M(N)$, где $M(N)$ — сложность умножения N -разрядных чисел, которую можно оценить, согласно результатам предыдущих

¹⁾ Гашков С. Б. Занимательная компьютерная арифметика. М.: URSS, 2012.

параграфов, по порядку как $N^{\log_2 3}$. Более быстрый алгоритм вычисления системы остатков при большом количестве модулей основан на методе «деления пополам». Для простоты предположим, что n является степенью двойки, и вычислим вначале попарные произведения $m_1 m_2, \dots, m_{n-1} m_n$, потом «четверные» произведения и т. д., и наконец произведения $M_1 = m_1 \dots m_{n/2}$ и $M_2 = m_{n/2+1} \dots m_n$. Если учесть, что функция $M(N)$ удовлетворяет неравенству

$$M(N_1 + N_2) \leq M(N_1) + M(N_2),$$

то сложность всех операций умножения оценивается как

$$M(Nn) \log_2 n.$$

Потом вычисляем $R_1 = m \bmod M_1$ и $R_2 = m \bmod M_2$. Применяя лемму 32, можно оценить сложность двух этих операций деления как

$$5M(Nn) + 50Nn + 14.$$

Теперь заметим, что для нахождения остатков $r_i = r \bmod m_i$ достаточно при $i \leq n/2$ найти $R_1 \bmod m_i$, а при $i > n/2$ найти $R_2 \bmod m_i$, т. е. исходная задача нахождения n остатков свелась к двум таким же задачам, но вдвое меньшего размера. Поэтому, применяя указанные выше неравенства, получаем полную верхнюю оценку сложности вычисления системы остатков r_i , равную

$$6M(Nn) \log_2 n + 50Nn \log_2 n + 14 \log_2 n.$$

При больших n эта оценка лучше оценки $n^2 M(N)$, если, конечно, для умножения использовать, например, алгоритм с оценкой по порядку $N^{\log_2 3}$.

Рассмотрим теперь вопрос о вычислении по системе остатков $n_i \bmod m_i$ остатка по $\bmod m_1 \dots m_n$. Для этого модифицируем алгоритм, приведенный в начале раздела. Для вычисления системы чисел $M \bmod m_i$ применяем изложенный выше прием «деления пополам». Непосредственное вычисление системы чисел M_i имеет оценку сложности $O(nM(Nn))$. Это слишком много. Но если

заметить, что

$$M_i \bmod m_i = \frac{M_i m_i \bmod m_i^2}{m_i} = \frac{M \bmod m_i^2}{m_i},$$

где $M = m_1 \dots m_n$, и вычислить вначале квадраты m_i , потом применить указанный выше алгоритм для вычисления $M \bmod m_i^2$ и выполнить деления на m_i , то получим оценку сложности, равную по порядку $M(Nn) \log_2 n$.

Далее вычисляем числа a_i с помощью алгоритма Евклида по уже найденным остаткам $M_i \bmod m_i$. Остается вычислить сумму

$$\sum_{i=1}^n n_i a_i M_i \bmod M.$$

Для этого опять применяем «деление пополам», используя тождество

$$\begin{aligned} \sum_{i=1}^n n_i a_i M_i \bmod M &= m_1 \dots m_{n/2} \sum_{i=\frac{n}{2}+1}^n n_i a_i M'_i \bmod m_{\frac{n}{2}+1} \dots m_n + \\ &+ m_{\frac{n}{2}+1} \dots m_n \sum_{i=1}^{n/2} n_i a_i M''_i \bmod m_1 \dots m_{n/2}, \end{aligned}$$

которое сводит рассматриваемую задачу к двум таким же задачам, но вдвое меньшего размера. Как и раньше, получаем оценку сложности, по порядку равную $M(Nn) \log_2 n$.

Глава 16

Сложность операций модулярной арифметики

Однако в общем случае эта теорема¹⁾ впервые была сформулирована и доказана Чин Чжу-Шао в работе Shu Shu Chiu Chang в 1247 г.

Д. Кнут.

Искусство программирования

Операцию умножения двух n -разрядных чисел по модулю n -разрядного числа можно выполнить со сложностью $6M(n) + O(n)$, что очевидно следует из результатов раздела 12. Однако если модуль фиксирован, то часть вычислений можно выполнить заранее, и эту оценку можно улучшить.

Теорема 23. Сложность операции сложения—вычитания n -разрядных чисел по данному n -разрядному модулю p равна $O(n)$, а сложность умножения не больше $3M(n) + O(n)$.

Доказательство. Для выполнения сложения чисел x, y по модулю p достаточно вычислить $x + y - p$, и если это число будет отрицательным, то заменить его на $x + y$. Для выполнения умножения чисел x, y по модулю p достаточно вычислить $f = xy$, а потом $f \bmod p$. Для выполнения последней операции надо найти $q = \lfloor f/p \rfloor$ и вычислить $f - qp$. Для вычисления q можно применить метод доказательства леммы 32, а именно воспользоваться

¹⁾ Китайская теорема об остатках.

тем, что $a = \lfloor 2^{2n}/p \rfloor$ можно вычислить заранее, тогда

$$\left| \left\lfloor \frac{f \lfloor 2^{2n}/p \rfloor}{2^{2n}} \right\rfloor - \left\lfloor \left\lfloor \frac{f}{2^n} \right\rfloor \frac{a}{2^n} \right\rfloor \right| \leq 2,$$

на вычисление произведения $\lfloor f/2^n \rfloor a$ достаточно $M(n)$ операций, так как числа $\lfloor f/2^n \rfloor$ и a являются n -разрядными, а вычисление чисел $\lfloor f/2^n \rfloor$ и $b = \lfloor \lfloor f/2^n \rfloor a/2^n \rfloor$ делается бесплатно.

Указанный метод модулярного умножения по существу совпадает с методом Баррета. \square

Для модулей p специального вида оценки теоремы, разумеется, можно улучшить. Еще большего улучшения можно достичь в случае выполнения операций с числами специального вида. Приведем несколько примеров, представляющих практический интерес.

Пусть $p = 2^n - 1$. Тогда прибавление—вычитание единицы по модулю p имеет сложность $4n + O(1)$, так как требуется приведение по модулю p , сложение—вычитание по модулю p имеет сложность $A(p) = 7n + O(1)$, умножение по модулю p школьным методом имеет сложность $6n^2 - n + O(1)$.

Умножение на элементы 2^k при любом целом k сводится к циклическому сдвигу и схемно делается бесплатно.

Если натуральное число C может быть представлено в виде алгебраической суммы $l(C)$ степеней двойки по модулю p , то сложность $M(C, p)$ умножения на константу C по модулю p не больше $(l(C) - 1)A(p)$.

Если число $C = C_1 C_2 \bmod p$, то очевидно

$$M(C, p) \leq M(C_1, p) + M(C_2, p).$$

Поэтому, например при $C = 1 + 2^3 + 2^5 + 2^8 = (1 + 2^3)(1 + 2^5)$

$$M(C, p) \leq 2A(p).$$

Можно комбинировать эти два приема.

Умножение на $1/C$ по модулю p можно выполнить по формуле

$$x \cdot \frac{1}{C} \bmod p = \frac{x - (x \bmod C)}{C} + \left(x \bmod C \cdot \frac{1}{C} \bmod p \right) \bmod p.$$

Пусть $p = 2^n + 1$. Представление чисел по этому модулю требует не n , а $n + 1$ разряд. Прибавление—вычитание единицы по модулю p имеет сложность $4n + O(1)$, сложение—вычитание по модулю p имеет сложность $A(p) = 7n + O(1)$, умножение по модулю p имеет сложность $6n^2 + 11n + O(1)$.

Умножение на элементы 2^k при $1 \leq k \leq n-1$ уже не сводится к циклическому сдвигу и схемно делается со сложностью $4n + O(1)$. При $k = n$ умножение на 2^k — это смена знака, а при $2n > k > n$ оно сводится к умножению на 2^{k-n} и смене знака.

Если натуральное число C может быть представлено в виде алгебраической суммы $l(C)$ степеней двойки по модулю p , то сложность умножения на константу C по модулю p не больше

$$(l(C) - 1)A(p) + (4n + C)l(C).$$

Например, $M(3, p) \leq 11n + O(1)$.

Умножение на константы $1/C$ делается аналогично модулю $2^n + 1$, но тоже чуть сложнее. Так, умножение 2-битового числа $x \bmod 3$ на число

$$\begin{aligned} \frac{1}{3} &= \frac{-(2^n - 1)}{6} = \frac{1 + 4 + \dots + 4^k}{2} = 2 + 8 + \dots + 2^{2k-1} + \frac{1}{2} = \\ &= 2 + 8 + \dots + 2^{2k-1} - 2^{n-1} = 2 + 8 + \dots + 2^{n-3} - 2^{n-1} = \\ &= 3 + 8 + \dots + 2^{n-3} + 2^{n-1} \bmod p \end{aligned}$$

делается со сложностью $2n + O(1)$. В итоге имеем

$$M\left(\frac{1}{3}, p\right) = 21n + O(1).$$

Пусть $p = 2^n + c$, $c = \pm 3, \pm 5$. Тогда прибавление—вычитание единицы по модулю p имеет сложность $4n + O(1)$, сложение—вычитание по модулю p имеет сложность $A(p) = 7n + O(1)$, умножение по модулю p имеет сложность $6n^2 + O(n)$. Умножение на элементы 2^k при $1 \leq k \leq n$ делается со сложностью $7n + O(1)$.

Если натуральное число C может быть представлено в виде алгебраической суммы $l(C)$ степеней двойки по модулю p , то

сложность умножения на константу C по модулю p не больше $(2l(C) - 1)A(p)$.

Умножение на константы $1/C$ делается аналогично случаю $p = 2^n - 1$. Так, при $p = 2^n - 3$ умножение 2-битного числа $x \bmod 3$ на число

$$\begin{aligned} \frac{1}{3} &= \frac{2^n - 1}{6} = \frac{1 + 4 + \dots + 4^k}{2} = 2 + 8 + \dots + 2^{2k-1} + \frac{1}{2} = \\ &= 2 + 8 + \dots + 2^{2k-1} + 2^{n-1} - 1 = 1 + 8 + \dots + 2^{n-1} \end{aligned}$$

делается бесплатно. В итоге имеем

$$M\left(\frac{1}{3}, p\right) = 19n + O(1).$$

Но еще проще выполнять операции по модулю 2^n . Очевидно, для этого вообще не нужно приводить результат по модулю. Более того, для выполнения умножения по этому модулю достаточно в каждом из сомножителей оставить только младшие n разрядов, перемножить полученные числа и в произведении из $2n$ разрядов оставить только младшие n разрядов. Обозначим сложность такого усеченного умножения через $M_0(n)$.

Задача 37. Модифицируйте школьный алгоритм так, чтобы получить оценку $M_0(n) \leq M(n)/2$.

Можно модифицировать и метод Карацубы для ускоренного вычисления усеченного произведения.

Задача 38. Докажите, что $M_0(2n) \leq M(n) + 2M_0(n) + 10n - 12$.

Указание. Складывать надо n -разрядные числа и только два раза, причем переносы в старший разряд вычислять не нужно.

Задача 39. Оцените скорость работы указанного выше алгоритма.

Подобные же задачи можно сформулировать и в случае умножения многочленов.

Как найти остаток от деления, не вычисляя частное

Пусть надо найти остаток от деления произвольного числа

$$x = (x_n \dots x_1)_2$$

на фиксированное число

$$p = (p_m \dots p_1)_2.$$

Естественно попытаться сделать это, не находя частное. По существу речь идет об алгоритмической реализации признака делимости на p .

Один из возможных способов таков. Представим x в виде

$$X2^g + Y, \quad X = (x_n \dots x_g)_2, \quad Y = (x_{g-1} \dots x_1)_2.$$

Остаток $a = X2^g \bmod p$ находится по таблице размера $2^{n-g+1} \times m$, а остаток $b = Y \bmod p$ находится по таблице размера $2^{g-1} \times m$. Выбирая $g-1 = \lceil n/2 \rceil$, можно обе таблицы слить в одну, имеющую длину $\lceil n/2 \rceil$ и ширину $2m$. Эту таблицу можно заранее вычислить и поместить в память компьютера.

Далее надо сложить a, b по модулю p . Для этого можно просто их сложить, из полученной суммы вычесть p и выбрать из этих двух чисел второе, если оно неотрицательно, или первое — в противном случае.

Этот метод можно обобщить, представляя x в виде суммы k чисел X_i , у каждого из которых число ненулевых битов не превосходит $\lceil n/k \rceil$, для вычисления всех остатков $a_i = X_i \bmod p$ используется таблица с длиной $\lceil n/k \rceil$ и шириной km .

Далее надо сложить остатки a_i по модулю p . Для этого можно просто их сложить, при этом число разрядов у суммы не превосходит $m + \lceil \log_2 k \rceil$. У полученного результата надо найти остаток по модулю p . Это можно сделать тремя способами: можно рекурсивно применить описанный выше алгоритм (это эффективно, если $m + \lceil \log_2 k \rceil$ достаточно велико); выполнить деление на p с остатком методом SRT (это эффективно, если $\lceil \log_2 k \rceil$ достаточно мало, во всяком случае не больше m); применить еще одну таблицу с длиной $m + \lceil \log_2 k \rceil$ и шириной m .

Второй метод вычисления $x \bmod p$, где

$$x = (x_n \dots x_1)_2,$$

является частным случаем предыдущего метода, в котором x представляется в виде суммы $k = n$ чисел X_i , у каждого из которых число ненулевых разрядов равно 1, т. е. каждое X_i является по существу степенью двойки,

$$X_i = x_i 2^{i-1}.$$

Поэтому $a_i = X_i \bmod p$ равно или нулю, или $2^{i-1} \bmod p$, а эти числа можно вычислить заранее. Далее надо сложить числа a_i по модулю p . Для этого можно просто их сложить, при этом число разрядов у суммы не превосходит $m + \lceil \log_2 n \rceil$, а у полученного результата надо найти остаток по модулю p . Как и выше, это можно сделать тремя способами: рекурсивно применить описанный выше алгоритм (это эффективно, если число $\log_2 n$ достаточно велико); выполнить деление на p с остатком методом SRT (это эффективно, если число $\log_2 n$ достаточно мало, во всяком случае $\leq m$); применить таблицу с длиной $m + \lceil \log_2 n \rceil$ и шириной m .

Третий метод удобен для вычисления остатка по фиксированному малому нечетному модулю. Вначале вычислим порядок двойки по модулю p , т. е. найдем такое минимальное k , что $2^k - 1$ делится на p . Известно, что k является делителем числа $\varphi(m)$, где φ — функция Эйлера. Для вычисления

$$x \bmod p = (x_n \dots x_1)_2 \bmod p$$

тогда можно воспользоваться формулой

$$x \bmod p = (x_k \dots x_1)_2 + \dots + (x_n \dots x_{n+1-n \bmod k})_2 \bmod p.$$

Выполнить сложение можно со сложностью $5(1 + o(1))(n + k)$. К полученному числу можно или применить рекурсивно тот же алгоритм, или выполнить деление методом SRT (если $\log_2 n/k$ мало по сравнению с k), или применить таблицу с длиной

$$\left\lceil \log_2 \left\lceil \frac{n}{k} \right\rceil \right\rceil + k.$$

Например, если $p = 15$ и n кратно 4, то $k = 4$ и

$$x \bmod 15 = (x_4 \dots x_1)_2 + \dots + (x_n \dots x_{n-3})_2 \bmod 15,$$

и вычисление

$$(x_4 \dots x_1)_2 + \dots + (x_n \dots x_{n-3})_2$$

можно выполнить со сложностью $5(1 + o(1))n$.

Если $p = 5$ и n кратно 4, то $k = 4$ и

$$x \bmod 5 = (x_4 \dots x_1)_2 + \dots + (x_n \dots x_{n-3})_2 \bmod 5,$$

и для вычисления

$$(x_4 \dots x_1)_2 + \dots + (x_n \dots x_{n-3})_2$$

применяется та же схема, что и для $p = 15$.

Иногда (например, если p простое и порядок двойки по модулю p четен) можно найти такое k , что $2^k + 1$ делится на p . Известно, что минимальное такое k равно половине порядка двойки по модулю p и является делителем числа $\varphi(m)/2$, где φ — функция Эйлера.

Для вычисления

$$x \bmod p = (x_n \dots x_1)_2 \bmod p$$

тогда можно воспользоваться формулой

$$\begin{aligned} x \bmod p &= (x_k \dots x_1)_2 - (x_{2k} \dots x_{k+1})_2 + \dots \\ &\dots \pm (x_n \dots x_{n+1-n \bmod k})_2 \bmod p \end{aligned}$$

и вычислить эту знакопеременную суммы со сложностью

$$5(1 + o(1))(n + k).$$

По сравнению с предыдущим методом число разрядов в полученной сумме может уменьшиться почти в два раза (если $\log_2 n/k = o(k)$).

При $p = 5$ указанный метод применим, но не дает преимущества. Но при $p = 11$ и $n = 64$ небольшое преимущество уже

проявляется. Действительно, тогда можно выбрать $k = 5$ и вычислить знакопеременную сумму

$$(x_5 \dots x_1)_2 - (x_{10} \dots x_6)_2 + \dots + (x_{64} \dots x_{61})_2 = (y_9 \dots y_1)_2,$$

где y_9 — знаковый бит, после чего $(y_9 \dots y_1)_2 \bmod 11$ можно вычислить с помощью небольшой таблицы.

Если же взять $k = 10$, то вычисляется сумма

$$(x_{10} \dots x_1)_2 + (x_{20} \dots x_{11})_2 + \dots + (x_{64} \dots x_{61})_2 = (y_{13} \dots y_1)_2,$$

после чего $(y_{13} \dots y_1)_2 \bmod 11$ можно вычислить с помощью таблицы.

При $p = 13$ и $n = 64$ можно выбрать $k = 6$ и вычислить знакопеременную сумму

$$(x_6 \dots x_1)_2 - (x_{12} \dots x_7)_2 + \dots + (x_{64} \dots x_{61})_2 = (y_{10} \dots y_1)_2,$$

где y_{10} — знаковый бит, после чего $(y_{10} \dots y_1)_2 \bmod 13$ можно вычислить с помощью таблицы.

Если при нахождении суммы

$$(x_k \dots x_1)_2 + \dots + (x_n \dots x_{n+1-n \bmod k})_2$$

сложение k -битных чисел выполнять по модулю p с m -битным результатом, то полученная сумма тоже будет m -битной и приводить ее по модулю уже будет не нужно. Этот прием эффективен при малых p . Например, для сложения по модулю 3 можно воспользоваться вместо формулы

$$(x_2 x_1)_2 + (y_2 y_1)_2 = (z_3 z_2 z_1)_2$$

формулой

$$(x_2 x_1)_2 + (y_2 y_1)_2 \bmod 3 = (u_2 u_1)_2,$$

$$u_1 = z_1 \oplus z_3, \quad u_2 = z_2 \oplus z_1 z_3.$$

Так как

$$z_1 = x_1 \oplus y_1, \quad z_2 = x_2 \oplus y_2 \oplus x_1 y_1, \quad z_3 = (x_2 \oplus y_2) x_1 y_1 \oplus x_2 y_2$$

и

$$(x_1 \oplus y_1)x_1y_1 = 0,$$

то

$$\begin{aligned} u_2 &= z_2 \oplus z_1z_3 = z_2 \oplus x_2y_2(x_1 \oplus y_1) = \\ &= ((x_2 \oplus y_2) \oplus x_1y_1) + (x_2y_2)(x_1 \oplus y_1). \end{aligned}$$

Сложность построенной схемы равна 10.

Поэтому, применяя эту схему для сложения по модулю 3 двух 2-битных чисел, можно вычислить

$$(x_2x_1)_2 + (x_4x_3)_2 + \dots \bmod 3$$

со сложностью $5n - 7$.

Еще один метод вычисления остатка по модулю p основан на использовании китайской теоремы об остатках. Этот метод пригоден только для составных p и применяется в комбинации с описанными выше методами при фиксированном p .

Пусть $p = p_1 \dots p_t$ и числа p_j попарно взаимно просты. Положим P_i равным произведению всех чисел p_j , кроме p_i . Пусть R_i — остаток от деления P_i на p_i , тогда R_i взаимно просто с p_i , и поэтому существует такое s_i , $0 < s_i < p_i$, что $s_i R_i = 1 \bmod p_i$. Значит, для любого r_i число $r_i s_i P_i$ делится на любое p_j , $j \neq i$, а при делении на p_i получается остаток r_i . Поэтому число

$$q = r_1 s_1 P_1 + \dots + r_t s_t P_t$$

при делении на p_i дает остаток r_i для каждого $i = 1, \dots, t$. Тот же остаток будет получаться при делении на p_i числа $r = q \bmod p$. Поэтому для вычисления $r = n \bmod p$ можно вначале вычислить все $r_i = n \bmod p_i$, а потом найти r по формуле

$$r = r_1 s_1 P_1 + \dots + r_t s_t P_t \bmod p,$$

так как число

$$r = q \bmod p$$

при делении на p_i дает те же остатки, что и n , значит, $n - q$ кратно p , т.е. $n \bmod p = r = q \bmod p$.

В качестве примера рассмотрим вычисление n по модулю $p = 15$. Так как $p = 15 = p_1 p_2 = 3 \cdot 5$, то

$$\begin{aligned} P_1 &= 5, & P_2 &= 3, \\ R_1 &= P_1 \bmod p_1 = 5 \bmod 3 = 2, \\ R_2 &= P_2 \bmod p_2 = 3 \bmod 5 = 3, \end{aligned}$$

значит,

$$\begin{aligned} s_1 &= R_1^{-1} \bmod p_1 = 2^{-1} \bmod 3 = 2, \\ s_2 &= R_2^{-1} \bmod p_2 = 3^{-1} \bmod 5 = 2, \end{aligned}$$

поэтому

$$n \bmod 15 = r_1 s_1 P_1 + r_2 s_2 P_2 \bmod 15 = 10r_1 + 6r_2 \bmod 15,$$

где $r_i = n \bmod p_i$. Однако указанный метод не будет в этом случае иметь преимущество перед описанным выше методом вычисления $x \bmod 15$.

Рассмотрим еще в качестве примера вычисление n по модулю $p = 143$. Так как

$$\begin{aligned} P_1 &= 13, & P_2 &= 11, \\ R_1 &= P_1 \bmod p_1 = 13 \bmod 11 = 2, \\ R_2 &= P_2 \bmod p_2 = 11 \bmod 13 = -2, \end{aligned}$$

значит,

$$\begin{aligned} s_1 &= R_1^{-1} \bmod p_1 = 2^{-1} \bmod 11 = 6, \\ s_2 &= R_2^{-1} \bmod p_2 = -2^{-1} \bmod 13 = 6, \end{aligned}$$

поэтому

$$n \bmod 143 = r_1 s_1 P_1 + r_2 s_2 P_2 \bmod 143 = 78r_1 + 66r_2 \bmod 143,$$

где $r_i = n \bmod p_i$. Так как p_i 4-битные числа, то и r_i тоже. Для вычисления $78r_1 + 66r_2 \bmod 143$ проще всего применить подходящую таблицу.

Эффективность указанных методов при применении к большим числам все же ненамного выше простого деления. Чтобы убедиться в этом, оценим сложность их применения.

Обозначим $D_1(m, n)$ сложность вычисления остатка от деления m -значного числа на n -значное. Введем еще обозначение

$D_{a,b}(N)$ для сложности вычисления остатка при делении N на a , который далее обозначаем $N \bmod a$, где b — основание системы счисления, в которой заданы m -разрядное число N и n -разрядное число a . Обозначим $\text{ord}_a b$ такое наименьшее r , что $b^r - 1$ кратно a . Это обозначение имеет смысл только при $(a, b) = 1$.

Обобщением признаков делимости на 9 и 11 являются следующие признаки делимости на $b - 1$ и $b + 1$ в b -ичной системе счисления: для любого $N = (u_m \dots u_1)_b$ справедливы равенства

$$N = nu_b(N) \bmod (b - 1) \quad \text{и} \quad N = u_1 - u_2 + u_3 - \dots \pm \bmod (b + 1).$$

Если вместо операций обычного сложения и вычитания использовать операции по $\bmod(b - 1)$ и $\bmod(b + 1)$, каждая из которых сводится к двум обычным операциям сложения—вычитания (одна из которых к тому же почти тривиальна), то получаем оценку

$$D_{a,b}(N) \leq 2(m - 1)$$

при $a = b \pm 1$. Справедливы следующие теоремы, которые приводятся без доказательства.

Теорема 24. Пусть $(a, b) = 1$, $r = \text{ord}_a b < m$. Тогда при $m \rightarrow \infty$

$$D_{a,b}(N) \leq \frac{\nu_b(N)}{b-1} + 2m + \frac{\alpha_b m}{r} + \\ + (2r + \alpha_b + o(1)) \log_b \left(\frac{m}{r} \right)^{1/r} + D_1(r, n) + O(r),$$

где $\alpha_b = \frac{1}{2|b| - 1}$.

Следствие. Если $r \rightarrow \infty$ и $r = o(m^{\log_2(4/3)})$, то

$$D_{a,b}(N) \lesssim \frac{\nu_b(N)}{b-1} + 2m \lesssim 3m,$$

а в среднем $D_{a,b}(N) \lesssim 2\frac{1}{2}m$.

Представим a в виде произведения $a_0 a_1$, где $(a_1, b) = 1$, a_0 делит b^k при некотором k , и положим $r = \text{ord}_{a_1} b$.

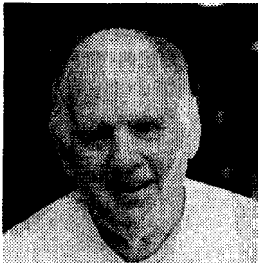
Теорема 25. При $m/\log_2 b \rightarrow \infty$

$$D_{a,b}(N) \leq \frac{\nu_b(N)}{b-1} + 2m + \frac{\alpha_b m}{r} + (2r + \alpha_b + o(1)) \log_b \left(\frac{m}{r} \right)^{1/r} + D_1(r, n) + O(r) + (1 + o(1)) n^2 \log_2 b.$$

Следствие. Если $b^n = o(m^{\log_2(4/3)})$, то $D_1(m, n) \lesssim 3m$, и

$$D(m, n) < D_0(m, n) + 3m + o(m).$$

Если остаток от деления на данное n -разрядное число a найден, то, вычитая его из делимого, получаем число, заведомо делящееся на a . Есть ли какой-нибудь способ быстро выполнить деление в этом случае, т. е. когда делимое делится на a без остатка?



Петер Монтгомери

Такой способ предложили Гренлунд и Монтгомери.

Они предложили свести деление к умножению. Пусть вначале a нечетно. Найдем $(m - n + 1)$ -разрядное число A , такое, что $aA \equiv 1 \pmod{2^{m-n+1}}$.

Они предложили свести деление к умножению. Пусть вначале a нечетно. Найдем $(m - n + 1)$ -разрядное число A , такое, что $aA \equiv 1 \pmod{2^{m-n+1}}$.

Теорема 26. Деление данного m -разрядного числа N на a можно выполнить по формуле

$$\frac{N}{a} = NA \pmod{2^{m-n+1}}$$

со сложностью $M_0(m - n + 1)$.

Доказательство. Заметим, что

$$\frac{N}{a} = \left(\frac{N}{a} \right) (aA) \pmod{2^{m-n+1}} = NA \pmod{2^{m-n+1}}.$$

Так как число $0 \leq N/a < 2^{m-n+1}$, то по произведению $NA \bmod 2^{m-n+1}$ это число восстанавливается однозначно.

Указанный алгоритм деления очевидно быстрее общего быстрого алгоритма деления с остатком, и его можно выполнять, используя как усеченный школьный алгоритм, так и усеченный алгоритм Карацубы.

Если же число a четно, то оно имеет вид $2^k a_1$, где a_1 нечетно. Так как $N/a = (N/2^k)/a_1$, первое деление выполняется очевидно бесплатно (просто отбрасываются нули в младших разрядах), и задача сводится к делению на нечетное число a_1 .

Число A можно вычислить заранее, например, с помощью расширенного алгоритма Евклида.

Есть и другой способ это сделать. Он основан на использовании ньютоновых итераций

$$x_{n+1} = x_n(2 - ax_n).$$

В качестве x_1 можно взять любое нечетное число, например 1, но лучше взять $x_1 = a$, так как тогда сразу

$$x_1 a = a^2 = 1 \bmod 8.$$

Если $ax_n = 1 \bmod 2^k$, то

$$1 - ax_{n+1} = (ax_n - 1)^2 = 0 \bmod 2^{2k},$$

откуда по индукции получаем, что $ax_n = 1 \bmod 2^{3 \cdot 2^{n-1}}$, значит, при $l = \lceil \log_2 2k/3 \rceil$

$$ax_l = 1 \bmod 2^k.$$

Полагая $k = m - n + 1$, получаем, что в качестве A можно взять

$$x_l \bmod 2^{n-m+1},$$

где $l = \lceil \log_2 2(n - m + 1)/3 \rceil$. Оценим сложность вычисления x_l . Для вычисления x_{k+1} при уже вычисленном x_k можно использовать формулу

$$x_{k+1} = x_k(2 - ax_k) \bmod 2^{3 \cdot 2^k}.$$

Сложность вычисления по ней не больше $2M_0(3 \cdot 2^k) + 12 \cdot 2^k$. Суммируя и учитывая, что $M_0(x/2) \leq M_0(x)/2$, получаем окончательную оценку сложности вычисления x_l при $l = \lceil \log_2 2(n - m + 1)/3 \rceil$ в виде

$$\begin{aligned} 4M_0(3 \cdot 2^l) + 24 \cdot 2^l &\leq 4M_0(2(m - n) + 1) + 16(m - n + 1) = \\ &= 4M_0(2(m - n)) + O(m - n). \end{aligned}$$

Указанный способ имеет меньшую сложность, чем основанный на расширенном алгоритме Евклида, во всяком случае, если пользоваться оценкой последнего в общем случае. Если применить указанный алгоритм для деления произвольного числа на произвольное же число, то оценка сложности будет

$$M_0(m - n) + 4M_0(2(m - n)) + O(m - n),$$

причем в некоторых случаях она может понижаться до

$$5M_0(m - n) + O(m - n).$$

При $(m - n)/n \rightarrow 0$ данная оценка лучше, чем оценка теоремы 18, но при $m/n \rightarrow \infty$ она хуже, чем оценка теоремы 17.

Укажем также способ быстрой проверки, делится ли нацело данное беззнаковое число n на нечетный делитель d . Для этого заранее вычислим мультипликативное обратное D к числу d по модулю 2^W , где W — длина машинного слова, одним из указанных выше методов. Очевидно, что D нечетно, и если n пробегает все возможные значения от 0 до $2^W - 1$, то $nD \bmod 2^W$ — остаток от деления nD на 2^W (записываемый очевидно младшими W битами произведения) тоже пробегает (но в другом порядке) все числа от 0 до $2^W - 1$. \square

Задача 40. Докажите сформулированный выше факт.

Если же n кратно d , то

$$\frac{n}{d} = nD \bmod 2^w,$$

поэтому, когда n пробегает все кратные d числа из рассматриваемого интервала

$$0, d, 2d, \dots \left\lfloor \frac{2^W - 1}{d} \right\rfloor d,$$

то $nD \bmod 2^W$ пробегает (в некотором порядке) все числа от 0 до $\lfloor (2^W - 1)/d \rfloor$.

Задача 41. Убедитесь и в этом.

Из указанных фактов немедленно следует, что если n кратно d , то $nD \bmod 2^W$ не больше $\lfloor (2^W - 1)/d \rfloor$, а если n не кратно d , то $nD \bmod 2^W$ больше $\lfloor (2^W - 1)/d \rfloor$.

На этом основан следующий алгоритм тестирования делимости нацело на данное нечетное число d , который мы приведем для примера в случае $d = 25$ (фактически это признак делимости на d в двоичной системе). Заранее вычисляем, что число

$$D = (11000010100011110101110000101001)_2 = (C28F5C29)_{16},$$

такое, что $dD = 1 \bmod 2^{32}$. Также заранее вычисляем число

$$C = \left\lfloor \frac{2^{32} - 1}{25} \right\rfloor = (A3D70A3)_{16}.$$

Умножаем D на n командой умножения, позволяющей вычислять 64-битный результат, и оставляем от него младшие 32 бита. Сравниваем результат с числом C . Если он больше, то n не делится на 25, в противном случае делится. Так как умножение 32-битных чисел — медленно выполняемая команда, его можно заменить на сложения—вычитания и сдвиги влево. Для минимизации числа сложений—вычитаний можно записать D в двоичной системе с отрицательными битами, минимизируя количество ненулевых битов. Так как $(1111)_2 = (10000 - 1)_2$, $(111)_2 = (1000 - 1)_2$, то, выполнив указанные преобразования, получим число, записанное 12 ненулевыми битами. Значит, умножение можно заменить на 9 сложений, 2 вычитания и 11 сдвигов влево. Все операции фактически проводятся по модулю 2^{32} .

Конечно, можно определить, делится ли число на 25, переводя его в десятичную систему, в которой указанная проверка очевидна. Но эта процедура требует более сложного программирования и выполняется медленнее, хотя и может быть сведена только к операциям сложения—вычитания.

Почти все результаты этого раздела имеют аналоги для операций в системах счисления с произвольным основанием b и для операций с многочленами, но мы не будем здесь их рассматривать.

Глава 17

Умножение и деление на константы

Когда он [Тьюринг] переходил к чтению моей работы, то обычно бывал великодушен в оценке; особенно он любил маленькие программистские трюки (кое-кто сказал бы, что он слишком их любил, чтобы быть «хорошим» программистом) и добродушно посмеивался, как мальчик, над маленькими хитростями, которые я использовал.

Джеймс Х. Уилкинсон.

Некоторые замечания математика-вычислителя
(лекция при получении премии Тьюринга за 1970 г.)

Вопрос о делении на константы не представляет особого интереса для тех, кто пользуется только калькулятором. И в том и в другом случае вычисления выполняются по алгоритмам, работающим с любыми числами, а в случае вычислений на калькуляторе скорость их выполнения практически не зависит от того, с какими числами они выполняются. Однако для вычислений на компьютере это уже не так. Дело в том, что умножение и деление целых чисел на компьютере — довольно медленные операции по сравнению со сложением—вычитанием и тем более с логическими операциями типа покомпонентной конъюнкции или дизъюнкции машинных слов, а также операциями сдвига машинного слова в ту или иную сторону. Если же в умножении или делении один операнд — заранее известная константа, и тем более, если операции с этой константой надо выполнять часто, то их можно существенно ускорить с помощью следующих простых приемов.

Умножение на константу положительных целых чисел в ряде случаев можно свести к небольшому числу сложений и сдвигов.

Например, если константа в двоичной записи имеет мало единиц, т. е. представима в виде

$$c = 2^{a_0} + 2^{a_1} + \dots + 2^{a_k},$$

то умножение x на нее сводится к сдвигам числа x на a_0, a_1, \dots, a_k позиций влево (сдвиг на s позиций влево означает появление s нулей в младших разрядах и равносильен умножению на 2^s , разумеется, в предположении отсутствия переполнения, которое происходит, если результат не вмещается в разрядную сетку компьютера) и $k - 1$ сложению. При небольших k соответствующая подпрограмма будет работать быстрее, чем использование команды умножения.

На самом деле вопрос об умножении на константы не так прост, как это кажется на первый взгляд. Для его оптимального решения полезно применить аддитивные цепочки. Рассмотрим, например, умножение x на 45. Так как $45 = (101101)_2$, то $45x$ можно, применяя бинарный метод для аддитивных цепочек, вычислить следующим образом:

$$\begin{aligned} t &\leftarrow 4x, & r &\leftarrow t + x, & t &\leftarrow 2r, \\ t &\leftarrow t + x, & t &\leftarrow 4t, & t &\leftarrow t + x. \end{aligned}$$

Однако возможны и другие варианты, столь же эффективные, например

$$\begin{aligned} t_1 &\leftarrow 4x, & t_2 &\leftarrow 8x, & t_3 &\leftarrow 32x, \\ r &\leftarrow t_1 + x, & t_3 &\leftarrow t_3 + t_2, & r &\leftarrow r + t_1 \end{aligned}$$

(второй вариант бинарного метода). Этот вариант требует большое число ячеек памяти и в этом он уступает первому варианту, но зато он будет быстрее работать на компьютерах, допускающих параллелизацию вычислений (второй вариант бинарного метода имеет ту же сложность, что и первый, но меньшую глубину, равную $\lambda(n) + 1$. В то же время его ширина также равна $\lambda(n) + 1$.) Однако бинарный метод не всегда оптимален, как нам известно. Например, его иногда превосходит метод множителей. И действительно,

применяя его к умножению на $45 = 5 \cdot 9$, получаем следующую программу:

$$t \leftarrow 4x + x, \quad r \leftarrow 8t + t.$$

Еще один метод ускорения вычислений связан с использованием аддитивных цепочек с вычитаниями и метода Бута (или любого другого метода) для минимизации числа ненулевых разрядов в представлении данного числа в двоичной системе с использованием отрицательных единиц. Например, любое 32-разрядное число может быть записано таким образом с помощью не более чем 16 разрядов, что требует не более 15 сложений—вычитаний.

Рассмотрим теперь вопрос о том, как можно быстро делить на некоторые константы, не пользуясь операцией деления. Деление — самая сложная операция компьютерной арифметики не только из-за медленности ее выполнения. Трудности при попытке ускорить компьютерные вычисления с большим количеством делений связаны еще и с тем, что используются разные операции деления в разных компьютерах, и вначале надо выяснить, какое именно деление выполняет ваш компьютер. Известны три разновидности компьютерного деления. Для всех из них выполняется тождество $a = bq + r$, где q — частное, r — остаток, и для положительных a, b эти операции не отличаются друг от друга. Отличия возникают при *знаковом делении*. Мы в этой книге в основном рассматривали операции с положительными числами, а там, где возникали отрицательные числа, использовали так называемое *модулярное деление*, в котором остаток всегда неотрицателен и меньше делителя. Этот вид деления наиболее естественен с чисто математической точки зрения. В некоторых компьютерах используется *деление с округлением частного к меньшему значению*. В этом случае частное $q = \lfloor a/b \rfloor$ не только при положительных a, b , но и во всех остальных случаях тоже. Так называемое *отсекающее деление* определяется так, чтобы выполнялось тождество

$$(-a) \div b = a \div (-b) = -(a \div b),$$

где $a \div b$ означает частное от деления a, b . Остатки у различных видов деления могут сильно отличаться друг от друга, но частные

отклоняются друг от друга не более чем на 1. Приведем формулу для вычисления частного при отсекающем делении:

$$n \div d = \begin{cases} \left\lfloor \frac{n}{d} \right\rfloor, & \text{если } d \neq 0, \quad nd \geq 0; \\ \left\lceil \frac{n}{d} \right\rceil, & \text{если } d \neq 0, \quad nd \leq 0. \end{cases}$$

Укажем несколько быстрых способов выполнять деление на некоторые константы.

Деление числа a на степень двойки 2^k выполняется путем сдвига числа на k позиций вправо (разумеется, мы предполагаем, что k меньше длины машинного слова, которое, как правило, равно 32). Такая команда есть в большинстве компьютеров и выполняется она очень быстро. В результате получается частное. Чтобы найти остаток, нужно выполнить побитовое умножение a на число $2^k - 1 = (0 \dots 01 \dots 1)_2$. В результате останутся только k младших битов, а старшие биты занулятся. Интересно, что указанный способ деления годится и для деления чисел любого знака при использовании в компьютере модулярного деления и представления отрицательных чисел в дополнительном коде (как было разъяснено в разделе 9).

Задача 42. Докажите факт, указанный выше.

Но в случае отсекающего деления указанный простой способ не годится.

Задача 43. Придумайте быстрый алгоритм отсекающего деления.

Деление на 3, 5, 7 и другие небольшие числа можно свести к умножению тем же методом, который использовался для быстрого деления в разделе 12. Рассмотрим эти вопросы подробнее. Предполагаем, что длина машинного слова 32 и компьютер имеет операцию умножения 32-разрядных чисел, или операцию, которая вычисляет старшие 32-бита этого произведения. Эта операция

равносильна делению 64-битного произведения двух 32-битных чисел на 2^{32} . Для нахождения частного $q = \lfloor n/3 \rfloor$ в случае неотрицательного n можно вычислить

$$\left\lfloor \frac{2^{32} + 2}{3} \frac{n}{2^{32}} \right\rfloor$$

с помощью указанной выше операции умножения n на число $(2^{32} + 2)/3$. Это число натуральное, и его, конечно, надо вычислить самому заранее и написать в тексте программы явно (например в шестнадцатеричной системе), а не поручать делать это за вас компьютеру (так можно не ускорить, а замедлить деление).

Для доказательства корректности полезны следующие факты, которые остаются читателю в качестве задач.

Задача 44. Докажите, что для целых $n, d \neq 0$ и действительного x

$$\left\lfloor \frac{n}{d} + x \right\rfloor = \left\lfloor \frac{n}{d} \right\rfloor, \quad \text{если } 0 \leq x < \left\lfloor \frac{1}{d} \right\rfloor,$$

и

$$\left\lceil \frac{n}{d} + x \right\rceil = \left\lceil \frac{n}{d} \right\rceil, \quad \text{если } 0 \leq x < \left\lceil \frac{1}{d} \right\rceil.$$

Задача 45. Докажите, что для целых $n, d > 0$

$$\left\lfloor \frac{n}{d} \right\rfloor = \left\lceil \frac{n-d+1}{d} \right\rceil \quad \text{и} \quad \left\lceil \frac{n}{d} \right\rceil = \left\lfloor \frac{n+d-1}{d} \right\rfloor.$$

При $d < 0$

$$\left\lfloor \frac{n}{d} \right\rfloor = \left\lceil \frac{n-d-1}{d} \right\rceil \quad \text{и} \quad \left\lceil \frac{n}{d} \right\rceil = \left\lfloor \frac{n+d+1}{d} \right\rfloor.$$

Действительно, при $n \geq 0$

$$\left\lfloor \frac{2^{32} + 2}{3} \frac{n}{2^{32}} \right\rfloor = \left\lfloor \frac{n}{3} + \frac{2n}{3 \cdot 2^{32}} \right\rfloor,$$

а так как $n < 2^{31}$, то

$$0 \leq x = \frac{2n}{3 \cdot 2^{32}} < \frac{1}{3},$$

значит, согласно задаче 44 имеем $q = \lfloor n/3 \rfloor$.

Если же $n < 0$, то к частному надо еще прибавить 1. Тогда

$$q = \left\lfloor \frac{2^{32} + 2}{3} \frac{n}{2^{32}} \right\rfloor + 1 = \left\lfloor \frac{2^{32}n + 2n + 3 \cdot 2^{32}}{3 \cdot 2^{32}} \right\rfloor = \left\lfloor \frac{2^{32}n + 2n + 1}{3 \cdot 2^{32}} \right\rfloor$$

согласно задаче 45. Поэтому

$$q = \left\lfloor \frac{n}{3} + \frac{2n + 1}{3 \cdot 2^{32}} \right\rfloor.$$

Так как $-2^{31} \leq n < 0$, то

$$\frac{-1}{3} + \frac{1}{3 \cdot 2^{32}} \leq \frac{2n + 1}{3 \cdot 2^{32}} \leq -\frac{1}{3 \cdot 2^{32}}.$$

Значит, согласно задаче 44 имеем $q = \lceil n/3 \rceil$ в этом случае. Поэтому отсекающее деление выполнено корректно. Если еще нужно найти остаток от деления на 3, то его можно вычислить как $n - 3q$. Можно проверить, что переполнение при выполнении этих операций не возникает. Умножение на 3 заменяется на сдвиг влево и сложение по формуле $3q = 2q + q$.

Беззнаковое деление на 3 (т.е. деление положительного числа) выполняется аналогичным образом, но с небольшими отличиями. В этом случае число n может достигать величины $2^{32} - 1$, и использовать умножение на число $(2^{32} + 2)/3$ уже нельзя, так как возникающая ошибка $2n/(3 \cdot 2^{32})$ может стать больше $1/3$. Однако можно использовать умножение на число $(2^{33} + 1)/3$, при этом применять команду, вычисляющую старшие 32 бита у беззнакового произведения 32-битных чисел:

$$q = \left\lfloor \frac{2^{33} + 1}{3} \frac{n}{2^{32}} \right\rfloor = \left\lfloor \frac{n}{3} + \frac{n}{3 \cdot 2^{33}} \right\rfloor.$$

Так как всегда $0 \leq n < 2^{32}$, то $0 < n/(3 \cdot 2^{33}) < 1/3$, и согласно задаче 44 имеем $q = \lfloor n/3 \rfloor$. Остаток от беззнакового (т.е. обычного,

«модулярного») деления на 3 вычисляется так же, как и в случае знакового деления.

Для выполнения знакового деления на 5 аналогичным образом хотелось бы использовать число $(2^{32} + 4)/5$, но ошибка будет слишком большой и иногда результат будет отличаться на единицу от верного. Однако, можно использовать множитель $(2^{33} + 3)/5$ следующим образом: умножаем n на этот множитель командой умножения знаковых чисел, которая вычисляет старшие 32 бита результата, потом полученное число сдвигается вправо на одну позицию. Эти действия равносильны делению 64-битного результата обычного знакового умножения на 2^{32} и взятию целой части от полученного числа:

$$q = \left\lfloor \frac{2^{33} + 3}{5} \frac{n}{2^{33}} \right\rfloor = \left\lfloor \frac{n}{5} + \frac{3n}{5 \cdot 2^{33}} \right\rfloor.$$

Для $0 \leq n < 2^{31}$ ошибка равна $3n/(5 \cdot 2^{33})$ и заключена в пределах от 0 до $1/5$. Поэтому согласно задаче 44 имеем $q = \lfloor n/5 \rfloor$. Если же $n < 0$, то алгоритм прибавляет к вычисленному ранее q единицу, и получается

$$\begin{aligned} q &= \left\lfloor \frac{2^{33} + 3}{5} \frac{n}{2^{33}} \right\rfloor + 1 = \left\lfloor \frac{2^{33}n + 3n + 5 \cdot 2^{33}}{5 \cdot 2^{33}} \right\rfloor = \\ &= \left\lfloor \frac{2^{33}n + 3n + 1}{5 \cdot 2^{33}} \right\rfloor = \left\lfloor \frac{n}{5} + \frac{3n + 1}{5 \cdot 2^{33}} \right\rfloor \end{aligned}$$

согласно задаче 45. А так как

$$-\frac{1}{5} < \frac{3n + 1}{5 \cdot 2^{33}} < 0,$$

то согласно задаче 44 имеем $q = \lceil n/5 \rceil$. Поэтому отсекающее деление выполнено корректно. Если еще нужно найти остаток от деления на 5, то его можно вычислить как $n - 5q$. Можно проверить, что переполнение при выполнении этих операций не возникает. Умножение на 5 заменяется на сдвиг влево на две позиции и сложение по формуле $5q = 4q + q$.

При знаковом делении на 7 не удастся аналогичным выше приведенному способу использовать множители $(2^{32} + 3)/7$ и $(2^{33} + 6)/7$ из-за слишком больших ошибок, а лишенный этого недостатка множитель $(2^{34} + 5)/7$ не помещается в 32-битовом слове. Однако умножение на это число можно заменить на умножение на отрицательное число $(2^{34} + 5)/7 - 2^{32}$ и последующее сложение и сдвиг вправо на две позиции. Более точно, алгоритм таков: умножаем n на $M = (2^{34} + 5)/7 - 2^{32}$ с вычислением старших 32 разрядов, что дает результат $q = \lfloor nM/2^{32} \rfloor$; прибавляем к нему n и получаем $q \leftarrow q + n = \lfloor nM/2^{32} + n \rfloor$; сдвигаем на две позиции вправо, т. е. делим на 4, и получаем $q \leftarrow \lfloor q/4 \rfloor = \lfloor \lfloor nM/2^{32} + n \rfloor / 4 \rfloor$. В итоге при $n \geq 0$ имеем

$$q = \left\lfloor \left\lfloor \frac{nM}{2^{32}} + n \right\rfloor / 4 \right\rfloor = \left\lfloor \left\lfloor \frac{2^{34}n + 5n}{7 \cdot 2^{32}} \right\rfloor / 4 \right\rfloor = \left\lfloor \frac{n}{7} + \frac{5n}{7 \cdot 2^{34}} \right\rfloor$$

согласно тождеству

$$\left\lfloor \left\lfloor \frac{a}{b} / d \right\rfloor \right\rfloor = \left\lfloor \frac{a}{bd} \right\rfloor.$$

Задача 46. Докажите вышеприведенное тождество и двойственное к нему тождество

$$\left\lceil \left\lceil \frac{a}{b} / d \right\rceil \right\rceil = \left\lceil \frac{a}{bd} \right\rceil.$$

Для $0 \leq n < 2^{31}$ ошибка равна $5n/(7 \cdot 2^{33})$ и заключена в пределах от 0 до $1/7$. Поэтому согласно задаче 44 имеем $q = \lfloor n/7 \rfloor$. Если же $n < 0$, то алгоритм прибавляет к вычисленному ранее q единицу, и получается согласно задаче 45

$$q = \left\lfloor \left\lfloor \frac{nM}{2^{32}} + n \right\rfloor / 4 \right\rfloor + 1 = \left\lfloor \frac{n}{7} + \frac{5n + 1}{7 \cdot 2^{34}} \right\rfloor.$$

В этом случае ошибка лежит в пределах от $-1/7$ до 0, поэтому согласно задаче 44 имеем $q = \lceil n/7 \rceil$. При выполнении всех указанных вычислений переполнение нигде не возникает. Поэтому отсекающее деление выполнено корректно. Если нужно найти

остаток от деления на 7, то его можно вычислить как $n - 7q$. Можно проверить, что переполнение при выполнении этих операций не возникает. Умножение на 7 заменяется на сдвиг влево на три позиции и вычитание по формуле $7q = 8q - q$.

При беззнаковом делении на 7 нельзя использовать множители $(2^{32} + 3)/7$, $(2^{33} + 6)/7$, $(2^{34} + 5)/7$ из-за слишком больших ошибок, но можно использовать множитель $M = (2^{35} + 3)/7 - 2^{32}$. Однако более элегантная схема была предложена Гренлундом и Монтгомери. Она основана на следующем тождестве

$$\left\lfloor \frac{q+n}{2^p} \right\rfloor = \left\lfloor \left(\left\lfloor \frac{n-q}{2} \right\rfloor + q \right) / 2^{p-1} \right\rfloor, \quad p \geq 1.$$

Применяем это тождество к $q = \lfloor Mn/2^{32} \rfloor$. Вычитание не вызывает переполнения, так как $0 \leq q \leq n$. Сложение также не вызывает переполнения, поскольку

$$\left\lfloor \frac{n-q}{2} \right\rfloor + q = \left\lfloor \frac{n-q}{2} + q \right\rfloor = \left\lfloor \frac{n+q}{2} \right\rfloor$$

и $0 \leq q, n < 2^{32}$. Алгоритм заключается в выполнении следующих операций:

$$\begin{aligned} q &= \left\lfloor \frac{Mn}{2^{32}} \right\rfloor, \quad t = n - q, \quad t \leftarrow \left\lfloor \frac{t}{2} \right\rfloor, \\ t &\leftarrow t + q = \left\lfloor \frac{n+q}{2} \right\rfloor, \quad q \leftarrow \left\lfloor \frac{t}{4} \right\rfloor. \end{aligned}$$

Тогда

$$q = \left\lfloor \left(n + \frac{Mn}{2^{32}} \right) / 8 \right\rfloor = \left\lfloor \frac{n}{7} \right\rfloor.$$

Быстрое деление можно организовать и в случае произвольного постоянного (т. е. заранее известного) делителя $d > 2$. Для этого надо уметь по данному размеру машинного слова W (обычно $W = 32$) и данному делителю d , $2 \leq d < 2^{W-1}$, находить наименьшее целое m , $0 \leq m < 2^W$, и целое $p \geq W$, такие, что

$$\left\lfloor \frac{mn}{2^p} \right\rfloor = \left\lfloor \frac{n}{d} \right\rfloor \quad \text{для} \quad 0 \leq n < 2^{W-1}$$

и

$$\left\lfloor \frac{mn}{2^p} \right\rfloor + 1 = \left\lfloor \frac{n}{d} \right\rfloor \quad \text{для} \quad -2^{W-1} \leq n < 0.$$

Число, которое будет использоваться в алгоритме для умножения, тогда определяется следующим образом:

$$M = \begin{cases} m, & 0 \leq m < 2^{W-1}; \\ m - 2^W, & 2^{W-1} \leq m < 2^W. \end{cases}$$

Число p выбирается из условия

$$2^p > n_c(d - (2^p \bmod d)),$$

где

$$n_c = \left\lfloor \frac{2^{W-1}}{d} \right\rfloor d - 1 = 2^{W-1} - 1 - (2^{W-1} \bmod d),$$

а знак $a \bmod b$ означает остаток от деления a на b при обычном («модулярном») делении. Можно доказать, что если предыдущее неравенство всегда имеет решение и если оно выполнено для некоторого p , то оно будет выполнено и для больших значений p . Решить это неравенство можно с помощью бинарного или даже линейного поиска.

После выбора числа p полагаем

$$m = \frac{2^p + d - (2^p \bmod d)}{d}.$$

Можно показать, что $0 \leq m < 2^W$, и если p и m выбраны, исходя из указанных выше условий, то будет обязательно выполнено нужное нам равенство

$$\left\lfloor \frac{mn}{2^p} \right\rfloor = \left\lfloor \frac{n}{d} \right\rfloor \quad \text{для} \quad 0 \leq n < 2^{W-1} \quad \text{и} \quad \left\lfloor \frac{mn}{2^p} \right\rfloor + 1 = \left\lfloor \frac{n}{d} \right\rfloor.$$

Доказательство этих фактов громоздкое и поэтому опускается. Читатель может его найти в книге Г. Уоррена «Алгоритмические трюки для программистов», где также приведены программы для вычисления p и m и рассмотрен случай $d < 0$, а также случай беззнакового деления.

Глава 18

Некоторые быстрые алгоритмы работы с битами

Я изучал эти маленькие программные самородки с тем же наслаждением, с которым другие пьют пиво или едят сладости. Я просто не мог остановиться. Каждая такая программа оказывалась для меня находкой, в каждой из них была интеллектуальная глубина, элегантность и даже своеобразная поэзия.

Гай Л. Стил младший, 2002
(из предисловия к кн. Генри С. Уоррена
Hacker's Delight)

Встречаются ситуации, когда приходится выполнять операции не с числами, а с битами, составляющими эти числа (или машинные слова). Для этого используются команды языка ассемблера. Но и язык С (и С++) имеет возможности для непосредственной работы с битами. Для быстрого выполнения различных манипуляций с битами в программистском фольклоре и в литературе известно множество различных эффективных программ. Замечательная коллекция таких трюков содержится в уже упоминавшейся книге Г. Уоррена. Приведем здесь некоторые из них.

Рассмотрим задачу подсчета числа единичных битов в машинном слове (т. е. количества единичных разрядов в данном 32-битном числе x). Известно много элегантных и эффективных решений этой задачи. Приведем одно из самых лучших. Заметим, что рассматриваемая задача очень близка к задаче построения логической схемы, выполняющей суммирование n однобитных

чисел в двоичной системе счисления. Известно, что такую схему можно построить следующим образом. Разобьем эти числа на пары и сложим каждую пару. Получим $n/2$ двухбитных чисел. Эти числа опять разобьем на пары и сложим каждую пару. Получим $n/4$ трехбитных чисел. Разобьем эти числа на пары и сложим каждую пару. Получим $n/8$ четырехбитных чисел. Далее получаем $n/16$ пятибитных чисел и т. д. В результате получится одно $(\log_2 n + 1)$ -битное число (предполагаем, что n есть степень двойки), которое и равно сумме всех n чисел, а другими словами, количеству единиц среди них.

Указанную схему при $n = 32$ можно промоделировать следующей компьютерной программой. Сначала вычисляем

$$x \& (010101 \dots 0101)_2$$

и получаем число

$$(0x_{31}0x_{29} \dots 0x_1)_2.$$

Аналогично получаем число $(0x_{32}0x_{30} \dots 0x_2)_2$ в результате операции

$$x \leftarrow (x \gg 1) \& (010101 \dots 0101)_2,$$

где $x \gg 1$ означает сдвиг битов числа x на один разряд вправо. Потом складываем полученные числа. Все вместе это выполняется следующим образом:

$$x \leftarrow x \& (010101 \dots 0101)_2 + (x \gg 1) \& (010101 \dots 0101)_2.$$

В результате получится число $(y_{32} \dots y_1)_2$, такое, что

$$(y_{2i}y_{2i-1})_2 = x_{2i} + x_{2i-1}, \quad i = 1, \dots, 16.$$

Значит, выполнив 4 команды, мы параллельно сложили 16 пар чисел. Далее выполняем команды

$$x \leftarrow (x \& (00110011 \dots 0011)_2) + ((x \gg 2) \& (00110011 \dots 0011)_2).$$

В результате получится число, состоящее из четверок битов, которые являются двоичными записями восьми сумм

$$x_{4i} + x_{4i-1} + x_{4i-2} + x_{4i-3}, \quad i = 1, \dots, 8.$$

Заметим, что первый бит в каждой четверке нулевой, так как сумма четырех однобитных чисел трехбитна.

Далее выполняем аналогичным образом команды

$$x \leftarrow (x \& (00001111 \dots 00001111)_2) + \\ + ((x \gg 4) \& (00001111 \dots 00001111)_2).$$

В результате получится число, состоящее из восьмерок битов, которые являются двоичными записями четырех сумм

$$x_{8i} + x_{8i-1} + \dots + x_{8i-7}, \quad i = 1, \dots, 4.$$

Заметим, что левая половина битов в каждой восьмерке нулевая, так как сумма восьми однобитных чисел четырехбитна.

Далее выполняем аналогичным образом команды

$$x \leftarrow (x \& (00000000111111110000000011111111)_2) + \\ + ((x \gg 8) \& (00000000111111110000000011111111)_2).$$

В результате получится число, состоящее из двух блоков по 16 бит, которые являются двоичными записями двух сумм

$$x_{16i} + x_{16i-1} + \dots + x_{16i-15}, \quad i = 1, \dots, 2.$$

Заметим, что левая половина битов в каждом блоке нулевая, так как сумма 16 однобитных чисел пятибитна.

Далее выполняем аналогичным образом команды

$$x \leftarrow (x \& (00000000000000001111111111111111)_2) + \\ + ((x \gg 16) \& (00000000000000001111111111111111)_2)$$

и получаем окончательно нужную нам сумму, в которой левая половина битов нулевая, а ненулевыми могут быть только правые 6 битов. Заметим, что каждую строку, кроме первой и второй, можно упростить, устраняя первую конъюнкцию и перегруппировывая скобки. Например, третью скобку можно переписать в виде

$$x \leftarrow (x + (x \gg 4)) \& (00001111 \dots 00001111)_2.$$

При этом результат работы каждой строки не изменится, так как каждый из четырехбитных блоков, на которые разбивается число x , полученное в результате работы первых двух строк, имеет нуль в самом левом бите, поэтому при сложении этих четырехбитных блоков не происходит переноса в другие блоки. Первую строку можно упростить, заменив на следующую:

$$x \leftarrow x - ((x \gg 1) \& (0101 \dots 01)_2).$$

Результат при этом не изменится, потому что при вычитании в каждой паре соседних битов будет производиться операция

$$2^{2i}(2x_{2i+1} + x_{2i}) - x_{2i+1}2^{2i} = 2^{2i}(x_{2i+1} + x_{2i}).$$

Далее эту программу можно несколько упростить, заменив последнюю строку на следующую:

$$x \leftarrow x + (x \gg 16).$$

Полученный результат будет совпадать с правильным только в 16 правых битах, но нужная нам информация содержится именно в них, точнее даже в 6 самых правых. Поэтому для правильной работы программы достаточно добавить строку

$$\text{return } x \& (00 \dots 0111111)_2.$$

Аналогичным образом можно заменить и предпоследнюю строку в исходной программе на

$$x \leftarrow x + (x \gg 8).$$

Результат тогда будет совпадать с правильным в самой правой восьмерке битов, в следующей восьмерке он, возможно, будет неправильным (ненулевым), а в следующей — опять правильным, причем сумма этих правильных результатов (чисел, двоичные записи которых задают указанные восьмерки) будет окончательным результатом программы. Поэтому, если после этой строки выполнять уже измененные командные строки

$$x \leftarrow x + (x \gg 16), \quad \text{return } x \& (00 \dots 0111111)_2,$$

окончательный результат все равно будет верным. Известны также много других эффективных программ для решения той же задачи. Некоторые из них основаны на следующих формулах, доказательство которых предлагается в виде задач.

Задача 47. Обозначим сумму битов n -битного числа x через $\|x\|$. Докажите, что

$$\|x\| = x - \left\lfloor \frac{x}{2} \right\rfloor - \left\lfloor \frac{x}{4} \right\rfloor - \dots - \left\lfloor \frac{x}{2^{n-1}} \right\rfloor.$$

Задача 48. Обозначим $(x \ll i)^{\text{rot}}$ циклический битовый сдвиг на i позиций. Докажите, что

$$\|x\| = - \sum_{i=0}^{n-1} (x \ll i)^{\text{rot}} \bmod 2^n.$$

Указание. Каждый бит при сдвигах пробегает все возможные позиции, и сумма полученных таким образом чисел по модулю 2^n равна

$$x(11 \dots 1)_2 \bmod 2^n = -x.$$

В качестве применения быстрого алгоритма вычисления $\|x\|$ можно найти знакопеременную сумму битов

$$s(x) = x_1 - x_2 + x_3 - \dots + x_{31} - x_{32}.$$

Очевидно

$$s(x) = \|x \& (0101 \dots 01)\| - \|x \& (1010 \dots 10)\|, \quad -16 \leq s(x) \leq 16.$$

В разделе 16 было показано, что остаток от деления неотрицательного числа x на 3 можно найти, не выполняя деления, вычислив $s(x)$ и заметив, что $x \bmod 3 = s(x) \bmod 3$. Далее быстрее всего воспользоваться предвычисленной таблицей остатков по модулю 3 чисел от -16 до 16 . В случае знаковых чисел в определении $s(x)$ нужно заменить $-x_{32}$ на $+x_{32}$. Тогда в формулу для вычисления $s(x)$ надо добавить слагаемое $((x \gg 31) \ll 1)$, если компьютер выполняет сдвиг знаковых чисел вправо с заполнением слева нулями, а если такой команды нет, но есть команда правого знако-

вого сдвига, при котором слева все биты заполняются знаковым битом, тогда надо вычесть это слагаемое.

Вместо $s(x)$ можно использовать с той же целью формулу

$$x_1 + 2x_2 + x_3 + 2x_4 + \dots + x_{31} + 2x_{32}$$

(для положительных чисел). Тогда вычисление чуть быстрее:

$$s(x) = \|x\| + \|x \& (10 \dots 10)_2\|.$$

Но размеры используемой далее таблицы немного возрастут.

Подобный же прием можно использовать и при вычислении остатка от деления на 7, но в нем уже придется три раза применять функцию $\|x\|$. Возможно, в этом случае более быстрой окажется программа, вычисляющая

$$s(x) = x_1 + 2x_2 + 4x_3 + x_4 + 2x_5 + 4x_6 + \dots$$

по формуле

$$\begin{aligned} s(x) &= (x \gg 30) \& (0 \dots 0111)_2 + \dots \\ &\dots + (x \gg 3) \& (0 \dots 0111)_2 + x \& (0 \dots 0111)_2. \end{aligned}$$

Остаток от деления на 15 можно вычислять с помощью функции

$$s(x) = x_1 + 2x_2 + 4x_3 + 8x_4 + x_5 + 2x_6 + 4x_7 + 8x_8 + \dots,$$

задаваемой формулой

$$\begin{aligned} s(x) &= (x \gg 28) \& (0 \dots 01111)_2 + \dots \\ &\dots + (x \gg 4) \& (0 \dots 01111)_2 + x \& (0 \dots 01111)_2 \end{aligned}$$

еще быстрее. Правда, размер используемой в конце таблицы возрастет до 120 чисел. Если составляющие ее числа от 0 до 15 заменить на их остатки по модулю 3 или 5, получим быстрые программы вычисления $x \bmod 3$ и $x \bmod 5$. Возможно, первая из них будет быстрее, чем указанная выше. Но для 64-разрядной машины, видимо, первая программа будет быстрее.

В разделе 16 был приведен алгоритм Гренлунда—Монтгомери для быстрого беззнакового деления в случае, когда заранее известно, что остаток равен 0. Там же отмечалось, что его можно применять и для деления с нахождением остатка, если найти остаток каким-нибудь алгоритмом без вычисления частного, вычесть его из делимого и применить алгоритм деления без остатка. Например, для деления на 3 или на 7 можно применить указанные выше алгоритмы вычисления остатка, не используя команды умножения и деления, а потом применить алгоритм Гренлунда—Монтгомери. Например, в случае деления нацело на 7 этот алгоритм выглядит следующим образом. Заранее вычисляем мультипликативное обратное к 7 по модулю 2^{32} , которое равно $(5 \cdot 2^{32} + 1)/7$.

Задача 49. Проверьте этот факт без калькулятора.

Умножаем делимое на это число и оставляем в результате младшие 32 бита (если не происходит переполнения, то результат находится простой командой умножения, которая может выполняться чуть быстрее, чем умножение с получением старших 32 битов, используемая в алгоритмах деления, указанных в разделе 17).

Функцию $\|x\|$ можно также применить для быстрой проверки машинного слова x на четность, т. е. для вычисления суммы его битов по модулю 2. Для этой цели достаточно обычную сумму битов z преобразовать в $z \& (00 \dots 01)_2$. Но еще быстрее будет работать следующая программа:

$$\begin{aligned} y &= x \wedge (x \gg 1); & y &= y \wedge (y \gg 2); & y &= y \wedge (y \gg 4); \\ y &= y \wedge (y \gg 8); & y &= y \wedge (y \gg 16), \end{aligned}$$

в которой знак \wedge означает покомпонентное сложение по модулю 2. Нужный нам бит четности оказывается в младшем разряде числа y , поэтому в конце надо добавить команду $y = y \& (0 \dots 01)_2$.

С помощью функции $\|x\|$ можно эффективно решить задачу поиска числа старших нулевых битов в данном машинном слове

(как всегда, 32-битном). Сначала выполняется следующая программа:

$$\begin{aligned}x &= x|(x \gg 1); & x &= x|(x \gg 2); & x &= x|(x \gg 4); \\x &= x|(x \gg 8); & x &= x|(x \gg 16)\end{aligned}$$

(знак $|$ в ней означает покомпонентную дизъюнкцию), в конце работы которой получается число y , составленное из следующих битов:

$$(x_{32}, x_{32} \vee x_{31}, x_{32} \vee x_{31} \vee x_{30}, \dots, x_{32} \vee x_{31} \vee \dots \vee x_1).$$

Очевидно в начале этого слова стоят нули, а потом идут сплошь единицы. Если к нему применить команду $\sim y$ (замена всех битов на их отрицания: нулей на 1 и наоборот), то получим слово, в начале которого идут единицы, а потом сплошь нули. К этому слову применяем подпрограмму, вычисляющую сумму битов. Она и дает нам число ведущих нулей в слове. Функция $nl(x)$ — число ведущих нулевых битов, тесно связана с функцией $\lfloor \log_2 x \rfloor$.

Задача 50. Докажите, что

$$\lfloor \log_2 x \rfloor = 31 - nl(x), \quad \lfloor \log_2 x \rfloor = 32 - nl(x - 1).$$

Задача 51. Как быстро вычислить число завершающих нулевых битов?

Маленькие хитрости в работе с битами

Как обнулить младший (крайний справа) единичный бит в данном числе x , т. е., например, вместо числа $x = (\dots 1010001000)_2$ получить число $(\dots 10100000000)_2$? Для этого надо выполнить вычисление $x \& (x - 1)$.

Задача 52. Докажите данное утверждение.

Указание. В рассмотренном примере $x - 1 = (\dots 1010000111)_2$ и покомпонентная конъюнкция уничтожает крайние справа единицы.

Как выделить в числе x крайний справа единичный бит (или получить нуль, если $x = 0$), т. е. например из числа

$$x = (10 \dots 1010001000)_2$$

получить число $(\dots 0000001000)_2$? Для этого надо выполнить вычисление $x \& (-x)$.

Задача 53. Докажите данное утверждение.

Указание. Пусть знаковое число $x = -x_{32}2^{31} + (x_{31} \dots x_1)_2$, тогда

$$\begin{aligned} -x &= x_{32}2^{31} - (x_{31} \dots x_1)_2 = \\ &= -(1 - x_{32})2^{31} + (2^{31} - (x_{31} \dots x_1)_2) = \\ &= -\neg x_{32}2^{31} + (2^{31} - 1 - (x_{31} \dots x_1)_2) + 1 = \\ &= -\neg x_{32}2^{31} + (\neg x_{31} \dots \neg x_1)_2 + 1 = \neg x + 1, \end{aligned}$$

т. е. для смены знака в дополнительном коде нужно инвертировать все биты и к результату прибавить 1. Например, если $x = (10 \dots 1010001000)_2$, то

$$-x = (01 \dots 0101110111)_2 + 1 = (01 \dots 0101111000)_2,$$

т. е. последние нули в x сначала заменяются на единицы, а после прибавления единицы опять превращаются в нули, а предшествующая им единица сначала превращается в нуль, а потом опять становится единицей (после прибавления единицы). Дальше переносы, появившиеся при прибавлении единицы, не распространяются, и все биты, лежащие левее упомянутой единицы, остаются инвертированными к соответствующим битам числа x . После покомпонентной конъюнкции они превращаются в нули.

Как выделить в числе x крайний справа нулевой бит, т. е., например, из числа $(01 \dots 0111)_2$ получить число $(000 \dots 0001000)_2$? Для этого можно использовать формулу

$$\neg x \& (x + 1).$$

Задача 54. Докажите данное утверждение.

Существует много тождеств, связывающих побитовые логические и арифметические операции друг с другом. Приведем список

таких тождеств, оставив доказательства читателю. Отрицательные числа в них представлены в дополнительном коде.

- $-x = \neg x + 1$;
- $-x = \neg(x - 1)$;
- $\neg x = -x - 1$;
- $-\neg x = x + 1$;
- $\neg(-x) = x + 1$;
- $x + y = x - (\neg y) - 1$;
- $x + y = (x \oplus y) + 2(x \& y)$;
- $x + y = (x \vee y) + (x \& y)$;
- $x + y = 2(x \vee y) - (x \oplus y)$;
- $x - y = x + \neg y + 1$;
- $x - y = (x \oplus y) - 2(\neg x \& y)$;
- $x - y = (x \& \neg y) - (\neg x \& y)$;
- $x - y = 2(x \& \neg y) - (x \oplus y)$;
- $x \oplus y = (x \vee y) - (x \& y)$;
- $x \& \neg y = (x \vee y) - y$;
- $x \& \neg y = x - (x \& y)$;
- $\neg(x - y) = y - x - 1$;
- $\neg(x - y) = \neg x + y$;
- $x \vee y = (x \& \neg y) + y$;
- $x \& y = (\neg x \vee y) - \neg x$;
- $x \equiv y = (x \& y) - (x \vee y) - 1$;
- $x \equiv y = (x \& y) + \neg(x \vee y)$.

В рассмотренных задачах речь шла о представлении некоторых отображений множества всех машинных слов в себя с помощью некоторой комбинации логических побитовых операций и операций сложения—вычитания и умножения. Возникает вопрос: как охарактеризовать все такие отображения, представимые указанным способом? Ответ дает следующая теорема.

Теорема 27. Отображение машинных слов может быть реализовано посредством побитовых логических операций и операций сложения—вычитания и умножения тогда и только тогда, когда в данном отображении каждый бит результата зависит только от битов исходных операндов в той же позиции или правее ее.

Доказательство. Чтобы доказать эту теорему в одну сторону достаточно заметить, что любая из упомянутых в ее формулировке операций обладает указанным в теореме свойством. Если представить себе, что рассматриваемое отображение реализуется некоторым абстрактным устройством, на входы которого в первый такт его работы подаются младшие (крайние справа) биты операндов, во второй такт подаются следующие биты (вторые с правого края) каждого операнда и т. д., а на выходе этого устройства в первом такте появляется младший бит результата, во втором такте — следующий бит и т. д., то указанное свойство рассматриваемого отображения есть не что иное, как свойство детерминированности, известное в теории конечных автоматов, а рассматриваемые отображения есть детерминированные отображения (ограниченные, впрочем, тем, что количество тактов работы этого устройства не превосходит 32). Понятно, что композиция детерминированных отображений снова будет детерминированным отображением, поэтому отображение, реализуемое рассматриваемым в теореме способом, обязательно будет детерминированным. Из теории автоматов известно, что любое детерминированное отображение, рассматриваемое на ограниченном числе тактов, можно представить с помощью побитовых операций дизъюнкции, конъюнкции и отрицания, а также операции сдвига влево обнулением младшего бита (операции $x \rightarrow (x \ll 1)$) при условии использования константы $1 = (0 \dots 01)_2$. Этим теорема доказана и в обратную сторону. \square

Задача 55. Проведите доказательство теоремы 27 во всех деталях.

Указание. Допустим, что $y = f(x)$ удовлетворяет условиям теоремы, тогда $y_i = f_i(x_1, \dots, x_i)$, где f_i — некоторая булева функция. Согласно известной

теореме алгебры логики функцию f_i можно выразить в виде формулы через булевы операции конъюнкции, дизъюнкции и отрицания. Рассмотрим машинные слова

$$x = (x_{32} \dots x_1)_2, \quad (x \ll 1) = (x_{31} \dots x_1 0)_2, \quad \dots, \\ (x \gg i) = (x_{32-i} \dots x_1 0 \dots 0)$$

и применим к ним побитовые операции $x \& y$, $x \vee y$, $\neg x$ в том же порядке, в каком они применялись в упомянутой формуле. В результате получится машинное слово Y_i , в котором i -я компонента будет равна $y_i = f_i(x_1, \dots, x_i)$. Поэтому

$$Y_i \& (0 \dots 0 \underbrace{10 \dots 0}_i)_2 = (0 \dots 0 \underbrace{y_i 0 \dots 0}_i)_2 = Z_i.$$

Остается выполнить указанные вычисления при всех i и заметить, что

$$(\dots (Z_1 \vee Z_2) \vee Z_3 \dots Z_{32}) = (y_{32} \dots y_1)_2 = y = f(x).$$

Из теоремы следует невозможность представления указанным в ней способом, например, отображения, которое заключалось бы в обнулении в слове x крайнего левого единичного бита. Так, невозможно определить, является ли данный единичный бит крайним левым, имея только информацию о битах, расположенных справа от него. По той же причине невозможно указанным способом выразить циклический сдвиг, сдвиг вправо или функции $nl(x)$, $\|x\|$. Тем не менее многие словарные функции, не удовлетворяющие условиям теоремы, можно компактно выразить через машинные операции.

Задача 56. Функция вычисления модуля знакового числа $\text{abs}(x)$ может быть выполнена без применения команд ветвления следующим образом. Сначала вычисляем $y = (x \gg 31)$ с помощью команды знакового сдвига. В результате получаем слово y , все биты которого совпадают со знаковым битом числа x . Потом вычисляем

$$\text{abs}(x) = (x \oplus y) - y.$$

Поле битов машинного слова или массива машинных слов можно использовать для представления множеств в компьютере.

Для этого элементам множества достаточно сопоставить единичные биты числа, и само множество будет представляться этим числом (или массивом чисел, если рассматриваемые множества слишком велики). При таком представлении теоретико-множественные операции (объединение, пересечение и пр.) будут представляться битовыми логическими операциями (пересечению соответствует конъюнкция, объединению — дизъюнкция, дополнению — отрицание, симметрической разности — сумма по модулю 2 и т. д.), равенство множеств будет соответствовать равенству чисел, включение одного множества в другое равносильно равенству $x \& y = x$. Иногда требуется рассматривать только множества одинаковой мощности (с равным числом элементов). Чтобы сгенерировать все такие множества, полезно иметь функцию, которая машинное слово с данным числом единиц переводит в машинное слово с тем же числом единиц, и изображающее большее число, причем следующее по величине среди чисел, изображающих множества с данным числом единицы. Например, эта функция должна переводить число $(\dots 00101110)_2$ в число $(\dots 00110011)$.

Задача 57. Докажите, что указанную функцию можно вычислить следующим образом:

$$s \rightarrow x \& (-x); \quad r \rightarrow s + x; \quad y \rightarrow r \vee (((x \oplus r) \gg 2) \div s).$$

Указание. Нужно найти крайнюю справа группу единичных битов, перед которой стоит хотя бы один ноль. Пусть, например $x = (\dots 011110000)_2$. Тогда $s = (0 \dots 010000)_2$, т. е. s представляет крайний справа единичный бит числа x . Далее

$$r = s + x = (\dots 100000000)_2$$

и крайняя справа единица в этом числе стоит на позиции нуля, соседнего слева с упомянутой группой единиц в числе x . Потом находим число

$$z = r \oplus x = (\dots 100000000)_2 \oplus (\dots 011110000)_2 = (0 \dots 0111110000)_2,$$

в котором по сравнению с x левый соседний с блоком единиц ноль заменился на единицу, а все левые по отношению к нему позиции заполнились нулями. Далее, вычисляя $(z \gg 2) \div s$, сдвигаем число y вправо на две позиции, а потом еще на $k - 1 = \log_2 s$ позиций вправо, потому

что $s = 2^{k-1}$, где k — номер позиции крайней правой единицы в x . Получаем в рассматриваемом примере $(0 \dots 0111)_2$, т.е. единичный блок переместился на правый край и стал короче на один бит в сравнении с исходным блоком. Наконец, находим

$$y = z \vee r = (\dots 100000111)_2.$$

В этом числе первая единица единичного блока сдвинулась влево на одну позицию, а остальные сдвинулись на правый край. Полученное число содержит столько же единиц, как и число x , но оно больше его, так как в нем вместо нуля, соседнего слева с упомянутым блоком единиц, появилась единица, причем среди всех чисел, больших x и содержащих такое же число единиц, полученное число будет наименьшим.

Задача 58. Как обменять в одном числе (одном регистре) два поля битов равной длины? Более точная формулировка. Пусть машинное слово $x = ABCDE$, где A, B, C, D, E его подслова, причем подслова B и D имеют одинаковую длину и их надо поменять местами, т.е. получить слово $y = ADCBE$. Обозначим m маску поля D , т.е. слово, в котором на позициях поля D стоят единицы, а на остальных позициях — нули. Пусть k — суммарная длина слов C, D (т.е. длина слова CD). Тогда для вычисления y можно сделать следующие манипуляции:

$$u = (x \oplus (x \gg k)) \& m; \quad v = (u \ll k); \quad y = x \oplus u \oplus v.$$

Указание. Число u в позициях поля D содержит биты подслова $B \oplus D$ и имеет нули в остальных позициях (из-за конъюнкции с маской m). Поэтому число v содержит биты слова $B \oplus D$ на позициях поля B (и нули в остальных позициях). Значит,

$$\begin{aligned} x \oplus u \oplus v &= (ABCDE)_2 \oplus (000(B \oplus D)0)_2 \oplus (0(B \oplus D)000)_2 = \\ &= (ADCBE)_2 = y. \end{aligned}$$

Глава 19

Вычисление некоторых целочисленных элементарных функций

Поразительно, сколько всего можно сделать, используя только операции двоичного сложения и вычитания вместе с некоторыми поразрядными операциями.

Гай Л. Стил младший, 2002
(из предисловия к кн. Генри С. Уоррена
Hacker's Delight)

Целочисленным квадратным корнем называется функция $\lfloor \sqrt{x} \rfloor$, а целочисленным логарифмом — $\lfloor \log_b x \rfloor$.

Целочисленный квадратный корень

Один из методов вычисления этой функции приведен в разделе 13 (теорема 20). Там же дана оценка сложности этого вычисления. Приведем еще один метод (близкий к упомянутому), но без оценки сложности. Опишем вариант этого метода для применения на 32-разрядном компьютере. Он начинает работу с выбора некоторой начальной целочисленной аппроксимации g_0 к корню \sqrt{a} , а затем делается серия уточнений этой аппроксимации, вычисляемой по формуле Ньютона—Герона $g_{n+1} = (g_n + a/g_n)/2$. Точнее, для проведения очередной итерации используется следующая модификация этой формулы:

$$g_{n+1} = \left\lfloor \left(g_n + \left\lfloor \frac{a}{g_n} \right\rfloor \right) / 2 \right\rfloor.$$

Корректность алгоритма обосновывается следующим утверждением:

если $g_n > \lfloor \sqrt{a} \rfloor$, то $\lfloor \sqrt{a} \rfloor \leq g_{n+1} < g_n$;

если $g_n = \lfloor \sqrt{a} \rfloor$, то $\lfloor \sqrt{a} \rfloor \leq g_{n+1} \leq \lfloor \sqrt{a} \rfloor + 1$.

Для доказательства заметим, что так как g_n — целое, то

$$\begin{aligned} g_{n+1} &= \left\lfloor \left(g_n + \left\lfloor \frac{a}{g_n} \right\rfloor \right) / 2 \right\rfloor = \left\lfloor \left\lfloor g_n + \frac{a}{g_n} \right\rfloor / 2 \right\rfloor = \\ &= \left\lfloor \left(g_n + \frac{a}{g_n} \right) / 2 \right\rfloor = \left\lfloor \frac{g_n^2 + a}{2g_n} \right\rfloor. \end{aligned}$$

Далее так как $g_n > \lfloor \sqrt{a} \rfloor$ и g_n — целое, то $g_n > \sqrt{a}$. Выберем ϵ так, чтобы $g_n = (1 + \epsilon)\sqrt{a}$. Тогда $\epsilon > 0$, и так как

$$\left\lfloor \frac{g_n^2 + a}{2g_n} \right\rfloor = g_{n+1} \leq \frac{g_n^2 + a}{2g_n},$$

то

$$\left\lfloor \frac{2 + 2\epsilon + \epsilon^2}{2(1 + \epsilon)} \sqrt{a} \right\rfloor = \left\lfloor \frac{(1 + \epsilon)^2 a + a}{2(1 + \epsilon)\sqrt{a}} \right\rfloor = g_{n+1} < \frac{g_n^2 + g_n^2}{2g_n} = g_n,$$

откуда

$$\lfloor \sqrt{a} \rfloor = \left\lfloor \frac{2 + 2\epsilon}{2(1 + \epsilon)} \sqrt{a} \right\rfloor \leq \left\lfloor \frac{2 + 2\epsilon + \epsilon^2}{2(1 + \epsilon)} \sqrt{a} \right\rfloor = g_{n+1} < g_n,$$

что и доказывает первое утверждение.

Для доказательства второго заметим, что если $g_n = \lfloor \sqrt{a} \rfloor$, то $\sqrt{a} - 1 < g_n \leq \sqrt{a}$, значит, $g_n^2 \leq a < (g_n + 1)^2$, поэтому

$$g_n = \lfloor g_n \rfloor = \left\lfloor \frac{g_n^2 + g_n^2}{2g_n} \right\rfloor \leq g_{n+1} \leq \left\lfloor \frac{g_n^2 + (g_n + 1)^2}{2g_n} \right\rfloor = \left\lfloor g_n + 1 + \frac{1}{2g_n} \right\rfloor,$$

откуда в силу целочисленности g_n и неравенства $1/(2g_n) < 1$ имеем

$$\lfloor \sqrt{a} \rfloor = g_n \leq g_{n+1} \leq \lfloor g_n + 1 \rfloor = g_n + 1 = \lfloor \sqrt{a} \rfloor + 1.$$

Начальное приближение к \sqrt{x} можно вычислить по формуле $g_0 = 2^s$, где $s = 16 - nl(x - 1)/2$, $nl(x)$ — функция подсчета числа старших нулевых битов, быстрый алгоритм вычисления которой был указан

в разделе 18. Разумеется, 2^s компьютер вычисляет командой $1 \ll s$. На первом шаге алгоритм вычисляет

$$g_1 = \frac{g_0 + x/g_0}{2}$$

с программной строкой $g_1 = (g_0 + (x \gg s)) \gg 1$, а потом выполняет итерацию

$$g_0 = g_1; \quad g_1 = g_0 + \left(\frac{x}{g_0}\right) \gg 1$$

до тех пор, пока впервые не получим $g_1 \geq g_0$. Экспериментально проверяется, что алгоритм заканчивает работу после пяти итераций (а для малых x и раньше).

Другой практически приемлемый алгоритм вычисления целочисленного квадратного корня можно построить, применяя метод бинарного поиска (или «метод вилки»). В этом методе после каждой итерации вычисляются числа a, b , такие, что $a \leq [x] + 1$, $b \geq [x]$, и для выполнения следующей итерации вычисляется $m = (a + b) \gg 1$, и если $m^2 > x$, полагаем $b = m - 1$, в противном случае $a = m + 1$. Итерации продолжаются, пока $b \geq a$, и после их окончания результат выводится в виде $a - 1$. В этом алгоритме не выполняются деления, но итераций он производит больше предыдущего. Чтобы уменьшить их количество, нужно подходящим образом выбрать начальные значения a, b . Например, можно взять $a = 1, b = [x/4] + 1$. Действительно, $x/4 + 1 \geq \sqrt{x}$, так как $x^2/4 + x/2 + 1 > x$ равносильно $(x/2 - 1)^2 \geq 0$. В качестве b можно также по аналогичной причине взять $[x/8] + 2, [x/16] + 4, [x/32] + 8$ и т. д., при этом, конечно, для больших x выгоднее брать значения из конца этого списка, а так как величина x может быть заранее неизвестна, то лучше всего взять $b = x \gg 5 + 8$, но, разумеется, не больше $2^{16} - 1$. При небольших x алгоритм работает быстро, но при $1 \leq x \leq 2^{32} - 1$ в среднем требует 16 итераций. Можно также выбрать начальные значения следующим образом:

$$b = \left(1 \ll \left(\frac{33 - nl(x)}{2}\right)\right) - 1, \quad a = b + \frac{3}{2}.$$

При $x \leq 10\,000$ это дает некоторое уменьшение числа итераций.

Целочисленные логарифмы

Быстрый алгоритм для вычисления $\lfloor \log_2 x \rfloor$ был указан в разделе 18.

Укажем несколько быстрых алгоритмов для вычисления $\lfloor \log_{10} x \rfloor$. Один из них основан на поиске в таблице. Таблица состоит из 11 значений 0, 9, 99, 999, ..., 999999999, $2^{32} - 1$. Для нахождения $\lfloor \log_{10} x \rfloor$ надо найти в этой таблице два соседних числа, между которыми заключено число x . Для этого можно применить бинарный поиск, и даже линейный поиск, выполнив цикл

```
for (i=-1; i++) { if (x <= table[i+1]) } return i;
```

Использование таблицы можно заменить последовательным умножением на 10, как в следующей программе:

```
p=1;
for (i=-1; i<=8; i++) {
    if (x < p) return i;
    p=(p<<2)+p;
    p=p<<1;
}
return i;
```

Если воспользоваться подпрограммой, быстро вычисляющей $nl(x)$, и приближенной формулой

$$\lfloor \log_{10} x \rfloor \approx \frac{9 \lfloor \log_2(x) \rfloor}{32},$$

то, используя по существу ту же таблицу, можно вычислить целочисленный десятичный логарифм программой из одной строки:

```
y = (9*(31-nl(x)))>>5;
if (x > table[y+1]) y=y+1;
return y;
```

Умножение на 9 лучше выполнять так: $9*x=(x<<3)+x$.

Глава 20

Быстрые операции с дробно-рациональными функциями

Асимптотически наилучшие алгоритмы зачастую оказываются наилучшим решением для тех задач, к которым они применимы.

Георг Кантор, Ханс Цассенхауз, 1981
(цитируется по книге Д. Кнута
«Искусство программирования»)

Сначала рассмотрим операцию сложения. Операция умножения дробно-рациональных функций сводится очевидным образом к умножению многочленов, а эта операция уже рассматривалась в разделах 1, 2, 6. Алгоритм быстрого умножения многочленов будет приведен в конце этого раздела.

Быстрое сложение дробно-рациональных функций

Справедлива следующая теорема.

Теорема 28. Сумму правильных рациональных дробей

$$\sum_{i=1}^s \frac{P_i}{Q_i} = \frac{P}{Q}$$

можно вычислить со сложностью

$$(3M(\deg Q) + 2 \deg Q) \lceil \log_2 s \rceil,$$

где $M(n)$ сложность умножения многочленов степени n .

Доказательство. Предполагаем, что

$$M(x + y) \geq M(x) + M(y).$$

Доказательство можно получить, применяя итеративно для сложения дробей метод деления пополам (т. е. разбивая сумму на две равные, или примерно равные подсуммы, вычисляя их, а потом складывая результаты) и замечая, что сложение двух дробей степени d требует не более $3M(d) + 2d$ операций, а число итераций не превосходит $\lceil \log_2 s \rceil$.

Аналогичное утверждение можно доказать и для сложения числовых дробей. \square

В следующей лемме дается тождество, которое называется разложением дроби на простейшие дроби.

Лемма 58. Пусть P/Q — произвольная правильная дробь степени n , $Q = q_1 \dots q_s$, $(q_i, q_j) = 1$, $i \neq j$. Положим $Q_i = Q/q_i$, тогда справедливо равенство

$$\frac{P}{Q} = \sum_{i=1}^s \frac{p_i}{q_i},$$

где

$$p_i = (P \bmod q_i \cdot Q_i^{-1} \bmod q_i) \bmod q_i.$$

Доказательство. Достаточно проверить, что $(Q_1, \dots, Q_s) = 1$, воспользоваться линейным представлением наибольшего общего делителя

$$1 = \sum_{i=1}^s U_i Q_i$$

и заметить, что

$$U_i \bmod q_i = Q_i^{-1} \bmod q_i$$

и

$$\frac{P}{Q} = \sum_{i=1}^s \frac{P \cdot U_i}{q_i} = \sum_{i=1}^s \frac{(P \cdot U_i) \bmod q_i}{q_i}.$$

\square

Теорема 29. Разложение на простейшие дроби можно вычислить со сложностью

$$E(n) + 4M(n) + 5n + \lceil \log_2 s \rceil (22M(n) + 10n),$$

где $E(n)$ — сложность расширенного алгоритма Евклида, примененного к многочленам степени n .

Доказательство. Предполагаем, что $E(x+y) \geq E(x) + E(y)$. Допустим, что $P \bmod q_i$ и $Q_i \bmod q_i$ уже вычислены. Тогда $Q_i^{-1} \bmod q_i$ находится со сложностью $E(\deg q_i)$ применением к паре $Q_i \bmod q_i, q_i$ расширенного алгоритма Евклида, который вычисляет линейное представление

$$1 = (Q_i, q_i) = UQ_i + Vq_i,$$

где $U = Q_i^{-1} \bmod q_i$, а $p_i = (P \bmod q_i) \cdot (Q_i^{-1} \bmod q_i)$ находится со сложностью $6M(\deg q_i) + 5 \deg q_i$ путем умножения и последующего деления результата на q_i (с помощью теоремы 15). Отсюда имеем оценку $E(n) + 6M(n) + 5n$. \square

Лемма 59. Сложность вычисления последовательности

$$P \bmod q_i, \quad i = 1, \dots, s,$$

оценивается как

$$\lceil \log_2 s \rceil (8M(n) + 4n) - M(n).$$

Доказательство. Для вычисления $P \bmod q_i$ применяем метод «деления пополам». Строим последовательность многочленов

$$Q_{\alpha_1, \dots, \alpha_k}, \alpha_i = 0, 1, \quad k \leq l(s) = \lceil \log_2 s \rceil,$$

такую, что

$$Q = Q_0 Q_1, \dots, Q_{\alpha_1, \dots, \alpha_k} = Q_{\alpha_1, \dots, \alpha_k, 0} \cdot Q_{\alpha_1, \dots, \alpha_k, 1}, \dots$$

и количество сомножителей в соседних многочленах $Q_{\bar{\alpha}, 0}, Q_{\bar{\alpha}, 1}$ отличалось бы не более, чем на 1. Вычисляем последовательность

многочленов $P_{\tilde{\alpha}}$ рекурсивно:

$$P_{\tilde{\alpha}, \alpha_{k+1}} = P_{\tilde{\alpha}} \bmod Q_{\tilde{\alpha}, \alpha_{k+1}}.$$

Если положить $\deg Q_{\tilde{\alpha}} = n_{\tilde{\alpha}}$, то сложность вычисления обоих многочленов $P_{\tilde{\alpha}, 0}$, $P_{\tilde{\alpha}, 1}$ оценивается (с помощью лемм 11 и 25) как

$$3(M(n_{\tilde{\alpha}, 1}) + n_{\tilde{\alpha}, 1}) + 2M(n_{\tilde{\alpha}}) + n_{\tilde{\alpha}, 0} + 3(M(n_{\tilde{\alpha}, 0}) + n_{\tilde{\alpha}, 0}) + \\ + 2M(n_{\tilde{\alpha}}) + n_{\tilde{\alpha}, 1} \leq 3M(n_{\tilde{\alpha}}) + 4n_{\tilde{\alpha}} + 4M(n_{\tilde{\alpha}}) = 7M(n_{\tilde{\alpha}}) + 4n_{\tilde{\alpha}},$$

а полная сложность — как

$$\sum_{k=1}^{l(s)} \sum_{\alpha_1=0, 1; \dots; \alpha_k=0, 1} 7M(n_{\tilde{\alpha}}) + 4n_{\tilde{\alpha}} \leq \\ \leq \lceil \log_2 s \rceil \left[7M \left(\sum_{\alpha_1=0, 1; \dots; \alpha_k=0, 1} n_{\tilde{\alpha}} \right) + 4 \sum_{\alpha_1=0, 1; \dots; \alpha_k=0, 1} n_{\tilde{\alpha}} \right] \leq \\ \leq \lceil \log_2 s \rceil (7M(n) + 4n).$$

Сложность вычисления последовательности многочленов $Q_{\tilde{\alpha}}$ также оценивается с помощью метода «деления пополам» как $(\lceil \log_2 s \rceil - 1)M(n)$. Поэтому полная сложность вычисления $P \bmod q_i$ оценивается как

$$\lceil \log_2 s \rceil (8M(n) + 4n) - M(n).$$

□

Лемма 60. Сложность вычисления последовательности

$$Q_i \bmod q_i, \quad i = 1, \dots, s,$$

оценивается как

$$\lceil \log_2 s \rceil (14M(n) + 6n) - M(n).$$

Доказательство. Так как

$$Q_i \bmod q_i = \frac{Q \bmod q_i^2}{q_i},$$

то сложность вычисления этой последовательности оценивается тем же методом (используем лемму 25) как

$$\lceil \log_2 s \rceil (8M(2n) + 8n) - M(2n) + 4M(n) + 4n.$$

Заметим, что возникающая в вычислениях последовательность остатков по $\text{mod } Q_{\tilde{\alpha}}^2$ представляется в виде $Q_{\tilde{\alpha}} \cdot R_{\tilde{\alpha}}$, где последовательность $R_{\tilde{\alpha}}$ вычисляется рекурсивно:

$$R_{\tilde{\alpha}, \alpha_{k+1}} = Q_{\tilde{\alpha}, 1 - \alpha_{k+1}} \cdot R_{\tilde{\alpha}} \text{ mod } Q_{\tilde{\alpha}, \alpha_{k+1}},$$

причем окончательные результаты $Q_i \text{ mod } q_i$ имеют вид $R_{\alpha_1, \dots, \alpha_{l(s)}}$. Так как $\deg R_{\tilde{\alpha}} < n_{\tilde{\alpha}}$, то сложность вычисления обоих многочленов $R_{\tilde{\alpha}, 0}$, $R_{\tilde{\alpha}, 1}$ оценивается (с помощью теоремы 15) как

$$\begin{aligned} & M(n_{\tilde{\alpha}}) + 3(M(2n_{\tilde{\alpha}, 0}) + 2n_{\tilde{\alpha}, 0}) + M(n_{\tilde{\alpha}} + n_{\tilde{\alpha}, 0}) + n_{\tilde{\alpha}, 1} + \\ & + M(2n_{\tilde{\alpha}, 0}, n_{\tilde{\alpha}, 1}) + M(n_{\tilde{\alpha}}) + 3(M(2n_{\tilde{\alpha}, 1}) + 2n_{\tilde{\alpha}, 1}) + M(n_{\tilde{\alpha}} + n_{\tilde{\alpha}, 1}) + \\ & + n_{\tilde{\alpha}, 0} + M(2n_{\tilde{\alpha}, 1}, n_{\tilde{\alpha}, 0}) \leq 4M(2n_{\tilde{\alpha}}) + M(3n_{\tilde{\alpha}}) + 3M(n_{\tilde{\alpha}}) + 7n_{\tilde{\alpha}}, \end{aligned}$$

а полная сложность как

$$\lceil \log_2 s \rceil (4M(2n) + M(3n) + 3M(n) + 7n).$$

Если вычислить последовательность $R_{\tilde{\alpha}}$ раньше, чем последовательность $P_{\tilde{\alpha}}$, то при вычислении последней не надо будет «обращать» многочлены $Q_{\tilde{\alpha}}$, так как это уже сделано при вычислении предыдущей последовательности (это ясно из леммы 25), что позволяет уменьшить оценку сложности на $3M(n) + 3n$, и оценка будет иметь вид

$$\lceil \log_2 s \rceil (4M(2n) + M(3n) + 3M(n) + 4n).$$

Можно еще уменьшить оценку, если перед умножением

$$R_{\tilde{\alpha}, \alpha_{k+1}} = Q_{\tilde{\alpha}, 1 - \alpha_{k+1}} \cdot R_{\tilde{\alpha}} \text{ mod } Q_{\tilde{\alpha}, \alpha_{k+1}}$$

привести $R_{\tilde{\alpha}}$ по $\text{mod } Q_{\tilde{\alpha}, \alpha_{k+1}}$, тогда «обращать» многочлен $Q_{\tilde{\alpha}}$ придется тоже только один раз, на умножение будет уходить $2M(n)$, а на приведение по модулю —

$$3M(n) + 3n + 4M(n) + 2n$$

операций. Суммарная оценка тогда будет иметь вид

$$\lceil \log_2 s \rceil (14M(n) + 6n) - M(n).$$

Можно еще заметить, что при выполнении условия $n_{\bar{\alpha},0} = n_{\bar{\alpha},1}$ равенства степеней соседних многочленов эта оценка еще уменьшается на $2\lceil \log_2 s \rceil M(n)$. Применяя леммы 59, 60, получаем утверждение теоремы.

Аналогичное утверждение можно доказать и для числовых дробей. \square

Быстрый китайский алгоритм

Интересно отметить, что быстрый вариант китайского алгоритма тесно связан с указанным выше быстрым алгоритмом разложения на простейшие дроби.

Пусть даны многочлены q_1, \dots, q_k , $(q_i, q_j) = 1$, $i \neq j$, и многочлены p_1, \dots, p_k , $\deg p_i < n_i = \deg q_i$, $n = n_1 + \dots + n_k$. Китайская теорема об остатках утверждает существование и единственность по mod Q многочлена P , такого, что $P \bmod q_i = p_i$, $1 \leq i \leq k$. Положим

$$Q = q_1 \dots q_k, \quad Q_i = \frac{Q}{q_i}, \quad R_i = Q_i^{-1} \bmod q_i,$$

тогда справедливы равенства

$$P = \sum_{i=1}^s (p_i Q_i R_i) \bmod Q = Q \sum_{i=1}^s \frac{(p_i R_i) \bmod q_i}{q_i}.$$

Теорема 30. Вычисление многочлена степени меньшей n по данным остаткам от деления на данные многочлены q_i имеет сложность не выше

$$E(n) + l(k)(13M(n) + 6n) + 6M(n) + 5n.$$

Доказательство. Поэтому для вычисления многочлена P достаточно найти произведения $r_i = p_i R_i \bmod q_i$, что делается так же,

как и выше, со сложностью

$$E(n) + 6M(n) + 5n + \lceil \log_2 k \rceil (10M(n) + 5n) - M(n),$$

а потом вычислить сумму правильных дробей степеней n_i

$$\frac{P}{Q} = \sum_{i=1}^k \frac{r_i}{q_i}$$

так же, как и в теореме 28, со сложностью $\lceil \log_2 k \rceil (3M(n) + n)$.
Итоговая оценка имеет вид

$$E(n) + \lceil \log_2 k \rceil (13M(n) + 6n) + 6M(n) + 5n. \quad \square$$

Быстрая интерполяция

В случае когда $q_i(x) = x - c_i$, а c_i и p_i принадлежат полю коэффициентов, как известно, многочлен P , построенный с помощью обратного китайского алгоритма, является решением интерполяционной задачи $P(c_i) = p_i$.

Теорема 31. Интерполяция выполняется со сложностью

$$\lceil \log_2 n \rceil (13M(n) + 6n) + 6M(n) + 6n.$$

Доказательство. В этом случае вычисление $R_i = Q_i^{-1} \bmod q_i$ по известным $Q_i \bmod q_i$ вместо применения расширенного алгоритма нахождения НОД многочленов требует просто n операций деления. Из сказанного выше вытекает, что сложность ее решения оценивается как

$$l(n)(13M(n) + 6n) + 6M(n) + 6n.$$

Если n является степенью двойки, то с учетом сделанного выше замечания эту оценку можно немного уменьшить.

Прямой китайский алгоритм вычисляет по многочлену P его остатки по заданным модулям q_i и, в частности, просто его значения в заданных точках c_i . Оценка его сложности фактически и была получена выше. \square

Еще о быстром умножении многочленов

Оценим сверху сложность умножения многочленов над полем действительных или комплексных чисел. Любопытно, что это можно сделать без упоминания о быстром преобразовании Фурье.

Для вычисления многочлена r , являющегося произведением многочленов p и q степени меньшей $n/2$, естественно выполнить их умножение по модулю $x^n - 1$, для чего вначале надо вычислить остатки от деления этих многочленов на линейные попарно взаимно-простые двучлены $x - \epsilon^k$, получающиеся при разложении многочлена $x^n - 1$ на множители (например, над полем \mathbb{C} — полем комплексных чисел), т. е. попросту вычислить значения $p(\epsilon^k)$ и $q(\epsilon^k)$, $k = 0, \dots, n-1$, потом, попарно перемножив их, получить значения $r(\epsilon^k) = p(\epsilon^k)q(\epsilon^k)$ и, применяя обратный китайский алгоритм, восстановить многочлен r . Удобно при этом предполагать n равным степени двойки (добавляя, если надо, нулевые младшие коэффициенты к многочленам p и q).

Применяя для вычисления значений $p(\epsilon^k)$ и $q(\epsilon^k)$, $k = 0, \dots, n-1$, метод «деления пополам», находим вначале

$$p \bmod x^{n/2} - 1, \quad p \bmod x^{n/2} + 1, \quad q \bmod x^{n/2} - 1, \quad q \bmod x^{n/2} + 1,$$

потом вычисляем остатки по модулям $x^{n/4} + 1$, $x^{n/4} - 1$, $x^{n/4} + i$, $x^{n/4} - i$ и т. д., пока не найдем остатки по модулям $x^2 - \epsilon^{2k}$, $k = 0, \dots, n/2$, и, наконец, по модулям $x - \epsilon^k$, $k = 0, \dots, n$. Так как деление многочлена степени меньшей m на двучлен степени $m/2$ осуществляется школьным алгоритмом со сложностью m , то сложность всего алгоритма вычисления значений $p(\epsilon^k)$ и $q(\epsilon^k)$, $k = 0, \dots, n-1$, оценивается как $2n \log_2 n$ (умножений и сложений, выполненных в поле \mathbb{C}).

Для восстановления (интерполяции) многочлена r по его известным значениям $r(\epsilon^k)$, $k = 0, \dots, n-1$, естественно применить формулу Лагранжа

$$r = f(x) \sum_{k=0}^{n-1} \frac{r(\epsilon^k)/f'(\epsilon^k)}{x - \epsilon^k} = f(x) \sum_{k=0}^{n-1} \frac{r(\epsilon^k)\epsilon^k}{n(x - \epsilon^k)},$$

где $f(x) = x^n - 1$, и сложение дробей выполнить так же, как указано выше. Так как при надлежащем выборе порядка суммирования все получающиеся дроби будут иметь двучленные знаменатели, а умножение многочлена степени меньшей m на двучлен степени m школьным методом имеет сложность m , то сложение двух таких дробей имеет сложность $4m$, а весь алгоритм интерполяции — сложность $2n + 2n \log_2 n$. В итоге получаем известное равенство $M(n) = O(n \log n)$.

Глава 21

Варианты алгоритма Евклида

— Вижу, он заинтересовался математикой.
— Не думаю, что он создан быть простым счетчиком, — предупредил Кларк.
— <...> дни напролет сидеть над тетрадями, корпеть над таблицами логарифмов, кубическими корнями, косинусами <...>
— Благодарение Декарту, математикам есть чем заняться, помимо этого, — промолвил Енох.
— Скажите брату, чтобы показал мальчику Евклида, и пусть он выбирает сам.

Нил Стивенсон. Ргуть (2003)

Со времен Евклида придумано немало новых вариантов его алгоритма, в основном с целью ускорения работы. Опишем некоторые из них.

Алгоритм Евклида с выбором минимального остатка

Обобщенный алгоритм Евклида определяет последовательность вычислений вида

$$a = bq_2 + \epsilon_2 r_2, \quad b = r_2 q_3 + \epsilon_3 r_3, \quad r_2 = r_3 q_4 + \epsilon_4 r_4, \quad \dots,$$

$$r_{k-3} = r_{k-2} q_{k-1} + \epsilon_{k-1} r_{k-1}, \quad r_{k-1} = r_k q_k,$$

где

$$\epsilon_i = \pm 1, \quad 0 < r_i < r_{i-1}, \quad i \geq 1, \quad b = r_1, \quad a = r_0.$$

Число $k - 1$ назовем длиной алгоритма. Как и в обычном алгоритме Евклида $(a, b) = r_k$. Обозначим $L(a, b)$ минимальную длину обобщенного алгоритма Евклида для вычисления (a, b) .

Алгоритмом Евклида с выбором минимального остатка называется обобщенный алгоритм Евклида, в котором всегда $2r_i \leq r_{i-1}$, т.е. на каждом шаге из двух возможных вариантов деления

$$r_{i-2} = r_{i-1}q_{i-1} + r_i, \quad \text{где } 0 < r_i < r_{i-1},$$

и

$$r_{i-2} = r_{i-1}(q_{i-1} + 1) - (r_{i-1} - r_i)$$

выбираем тот, при котором получается минимальный по абсолютной величине остаток (если они равны по модулю, то берем любой из них). Обозначим $L_0(a, b)$ минимальную длину алгоритма Евклида с выбором минимального остатка. Следующая теорема показывает, что этот алгоритм является наилучшим, среди обобщенных алгоритмов Евклида.



Леопольд Кронекер

Теорема 32 (теорема Кронекера). Справедливо неравенство

$$L_0(a, b) \leq L(a, b).$$

Сначала докажем следующую лемму.

Лемма 61. При $a \geq 2b$

$$L_0(a, b) \leq L_0(a, a - b).$$

Доказательство. Применим индукцию по a . База ($a = 2$) очевидна. Для обоснования шага индукции рассмотрим три случая. Пусть $a \geq 3b$, $a = bq_2 + \varepsilon_2 r_2$, $2r_2 \leq b$, тогда $a = (a - b) \cdot 1 + b$, так как $2b \leq a - b$ и $a - b = b(q_2 - 1) + \varepsilon_2 r_2$, поэтому $L_0(a, b) = L_0(a, a - b) - 1$.

Пусть теперь $2b \leq a < 5b/2$. Тогда первые два шага алгоритма в применении к паре (a, b) имеют вид

$$a = b \cdot 2 + (a - 2b), \quad b = r_2 q_3 + \varepsilon_3 r_3, \quad r_2 = a - 2b,$$

а в применении к паре $(a, a - b)$ имеют вид

$$\begin{aligned} a &= (a - b) \cdot 2 - (a - 2b), \\ a - b &= b + r_2 = r_2(q_3 + 1) + \varepsilon_3 r_3. \end{aligned}$$

Поэтому

$$L_0(a, b) = L_0(a, a - b).$$

В третьем случае считаем, что $5b/2 \leq a < 3b$. Тогда первый шаг алгоритма в применении к паре (a, b) имеет вид

$$a = b \cdot 3 - (3b - a), \quad r_2 = 3b - a,$$

значит,

$$L_0(a, b) = L_0(b, r_2) + 1,$$

а в применении к паре $(a, a - b)$ имеет вид

$$a = (a - b) \cdot 2 - (a - 2b) = (a - b) \cdot 2 - (b - r_2),$$

значит,

$$L_0(a, a - b) = L_0(a - b, b - r_2) + 1.$$

Так как

$$a - b = 2b - r_2 = b + (b - r_2),$$

то из равенства

$$a - b = (b - r_2)q_3 + \varepsilon_3 r_3$$

следует равенство

$$b = (b - r_2)(q_3 - 1) + \varepsilon_3 r_3,$$

поэтому

$$L_0(a, b) = L_0(a - b, b - r_2) + 1 = L_0(b - r_2, r_3) + 2 = L_0(b, b - r_2) + 1.$$

Согласно предположению индукции

$$L_0(b, b - r_2) = L_0(b, r_2),$$

откуда

$$L_0(a, b) = L_0(a, a - b).$$

Для доказательства теоремы применим индукцию и лемму 61.

Пусть

$$a = bq_2 + \varepsilon_2 r_2, \quad 2r_2 \leq b, \quad a = bq'_2 + \varepsilon'_2 r'_2.$$

Если $r_2 = r'_2$, то $\varepsilon_2 = \varepsilon'_2$, $q_2 = q'_2$ и согласно предположению индукции

$$L_0(a, b) = L_0(b, r_2) + 1 \leq L(b, r_2) + 1 = L(a, b).$$

Если же $r_2 < r'_2$, то $\varepsilon_2 = -\varepsilon'_2$, $q_2 = q'_2 - \varepsilon_2$, $r_2 = b - r'_2$ и согласно предположению индукции и лемме 61 имеем

$$L_0(a, b) = L_0(b, r_2) + 1 \leq L(b - r_2, b) + 1 = L(a, b).$$

Теорема доказана. \square

Теорема 33 (теорема Дюпре). Справедливо неравенство

$$L_0(a, b) \leq \min \left\{ \lfloor \log_a (2\sqrt{2} \min(a, b) + \sqrt{2}) \rfloor, \right. \\ \left. \lfloor \log_a (2\sqrt{2}(a + b) + \sqrt{2}) \rfloor - 1 \right\}.$$

Доказательство. Рассмотрим рекуррентную последовательность

$$u_{n+1} = 2u_n + u_{n-1}, \quad u_1 = 1, \quad u_0 = 0.$$

Лемма 62. Справедливо неравенство

$$u_k \leq n \iff k \leq \lfloor \log_a (2\sqrt{2}n + \sqrt{2}) \rfloor.$$

Доказательство. Формула

$$u_k = 2^{-3/2} ((\sqrt{2} + 1)^k - (1 - \sqrt{2})^k)$$

доказывается по индукции. Из нее выводится, что ближайшее целое к числу $2^{-3/2}(\sqrt{2} + 1)^k$ есть u_k . Так как число $2^{-3/2}(\sqrt{2} + 1)^k - 1/2$ не целое (оно иррационально), то

$$0 < u_k - 2^{-3/2}(\sqrt{2} + 1)^k + \frac{1}{2} < 1,$$

и значит,

$$u_k \leq n \iff 2^{-3/2}(\sqrt{2} + 1)^k - \frac{1}{2} \leq n,$$

откуда следует, что

$$u_k \leq n \iff k \leq \lfloor \log_{\alpha} (2\sqrt{2}n + \sqrt{2}) \rfloor. \quad \square$$

Продолжим доказательство теоремы 33.

Пусть алгоритм Евклида порождает равенства

$$a = bq_2 + \epsilon_2 r_2, \quad b = r_2 q_3 + \epsilon_3 r_3, \quad r_2 = r_3 q_4 + \epsilon_4 r_4, \quad \dots,$$

$$r_{k-3} = r_{k-2} q_{k-1} + \epsilon_{k-1} r_{k-1}, \quad r_{k-1} = r_k q_k,$$

где $2r_i \leq r_{i-1}$, $i = 2, \dots, k$, $r_1 = b$. Тогда

$$r_i = r_{i+1} q'_{i+2} + r'_{i+2} \geq 2r_{i+1} + r_{i+2}, \quad i = 1, \dots, k-2, \quad r_{k-1} \geq 2r_k,$$

откуда с помощью индукции получается, что

$$r_{k-i} \geq u_{i+1}, \quad i = 0, \dots, k-1,$$

значит,

$$b \geq u_k, \quad a \geq b + r_2 \geq b + u_{k-1}, \quad a + b \geq u_{k-1}.$$

Из леммы 62 следует теперь, что

$$k \leq \lfloor \log_{\alpha} (2\sqrt{2} \min(a, b) + \sqrt{2}) \rfloor,$$

$$k \leq \lfloor \log_{\alpha} (2\sqrt{2}(a + b) + \sqrt{2}) \rfloor - 1.$$

Теорема доказана. □

Заметим, что алгоритм Евклида с выбором минимального по модулю остатка на каждом шаге не более чем вдвое короче по числу шагов деления обычного алгоритма Евклида. Оценка достигается на паре чисел F_{2n-1} и F_{2n} .

Бинарный алгоритм Евклида

Рассмотрим еще один вариант алгоритма Евклида — бинарный алгоритм Евклида.

Очевидно справедливы следующие утверждения: если u и v четны, то $(u, v) = 2(u/2, v/2)$; если u четно, а v нечетно, то $(u, v) =$

$= (u/2, v)$; если u и v нечетны, то $u - v$ четно,

$$|u - v| < \max(u, v) \quad \text{и} \quad (u, v) = (|u - v|, \min(u, v)).$$

Алгоритм, основанный на этих утверждениях, называется бинарным вариантом алгоритма Евклида. Оценка его сложности дается следующей теоремой.

Теорема 34 (теорема Кнута—Шаллита). Число вычитаний, выполняемых бинарным алгоритмом в применении к паре чисел u, v , не больше $1 + \lfloor \log_2 \max(u, v) \rfloor$, и равенство возможно, лишь когда $\lfloor \log_2(u + v) \rfloor > \lfloor \log_2 \max(u, v) \rfloor$.

Доказательство. Положим

$$m = \lfloor \log_2 u \rfloor, \quad n = \lfloor \log_2 v \rfloor.$$

Пусть $m = n$. Можно считать, что $u > v$. Выполнив шаг деления-сдвига, по индукции получаем, что нужно еще выполнить не более $m + 1$ шага деления. Если бы понадобился ровно $m + 1$ шаг, то мы имели бы

$$\lfloor \log_2 ((u - v)2^{-r} + v) \rfloor > \lfloor \log_2 v \rfloor,$$

где $r \geq 1$ — число уже выполненных сдвигов вправо, что невозможно, так как

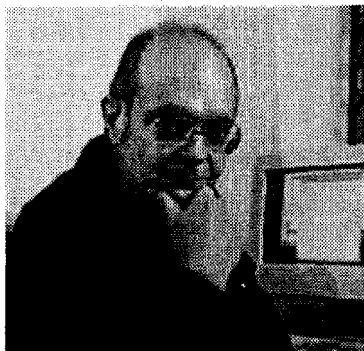
$$(u - v)2^{-r} + v \leq (u - v) + v = u.$$

Пусть $m > n$. Выполнив шаг деления, получим пару (u', v) , где

$$u' = (u - v)2^{-r}, \quad r \geq 1, \quad \lfloor \log_2 u' \rfloor = m - k, \quad k \geq 1.$$

Согласно предположению индукции осталось сделать не более $1 + \max(m - k, n) \leq m$ шагов. Если требуется ровно m шагов, то согласно предположению индукции $\lfloor \log_2(u' + v) \rfloor \geq m$, откуда

$$\begin{aligned} \lfloor \log_2(u + v) \rfloor &= \left\lfloor \log_2 \left[2 \left(\frac{u - v}{2} + v \right) \right] \right\rfloor > \lfloor \log_2 2(u' + v) \rfloor \geq \\ &\geq m + 1 > \lfloor \log_2 v \rfloor. \end{aligned} \quad \square$$



Дональд Кнут

Выше были получены оценки числа шагов для различных вариантов алгоритма Евклида. Представляют интерес также оценки сложности всех вычислений, выполняемых этим алгоритмом.

Задача 59. Докажите, что обычный алгоритм Евклида для n -разрядных чисел имеет оценку сложности $E(n) = O(n^2)$.

Если для выполнения арифметических операций в алгоритме Евклида использовать быстрые алгоритмы деления с остатком, то можно построить быструю версию алгоритма Евклида, имеющую сложность $O(M(n) \log_2 n)$, однако это нелегко доказать.

Алгоритм Евклида можно применять и к многочленам, также многие его варианты имеют полиномиальные аналоги. Например, справедливы следующие утверждения, доказательство которых предлагается выполнить читателю.

Задача 60. Докажите, что обычный алгоритм Евклида для многочленов степени не выше n имеет оценку сложности

$$E(n) \leq \frac{7}{2}n^2 + \frac{3}{2}n.$$

Указание. Надо учесть необходимость на каждом шаге алгоритма выполнять приведение полученного остатка к виду, в котором старший коэффициент будет равен единице.

Задача 61. Докажите, что обычный алгоритм Евклида для многочленов степени не выше n имеет оценку сложности

$$\frac{7}{2}(n^2 - m^2) + O(n),$$

где m — степень вычисленного наибольшего общего делителя.

Задача 62. Докажите, что обычный алгоритм Евклида для многочленов степени n и m имеет оценку сложности

$$E(n) \leq 2nm + \frac{m^2}{2} + \frac{5n}{2}.$$

Указание. Сначала рассмотрите случай, когда на каждом шаге алгоритма степень уменьшается на единицу. Потом покажите, что в общем случае оценка не выше, чем в этом частном случае.

Глава 22

Еще о схеме Горнера

Если начнем с древних греков, то мы найдем резкое разграничение чистой и прикладной математики <...> К чистой математике относится <...> евклидово построение геометрии, к прикладной принадлежат в особенности числовые операции <...> При этом к последней относились довольно презрительно — предрассудок, который во многих случаях сохранился до сих пор, но, во всяком случае, большей частью только у людей, которые сами не умеют вычислять.

Феликс Клейн. Элементарная математика с точки зрения высшей.
Т. 1 (перевод на русский 1933 г.)

Основные применения схемы Горнера связаны с алгебраическими вычислениями и относятся к компьютерной алгебре. Их известно на удивление много. Приведем некоторые из них.

Теорема 35. Младшие $k + 1$ коэффициентов многочлена $p(x + c)$, равные

$$p(c), p'(c), \dots, \frac{p^{(k)}(c)}{k!},$$

можно вычислить со сложностью $(k + 1)(2n - k)$, где $n = \deg p(x)$. В частности, все коэффициенты вычисляются со сложностью $n(n + 1)$.

Доказательство. Обозначим $G(n, k + 1)$ сложность этого вычисления. С помощью схемы Горнера со сложностью $2n$ вычислим

$q(x)$ и $p(c)$, где

$$p(x) = (x - c)q(x) + p(c).$$

Тогда

$$p(x + c) = xq(x + c) + p(c),$$

и для вычисления остальных k коэффициентов достаточно найти k младших коэффициентов у многочлена $q(x + c)$ степени $n - 1$. Поэтому

$$G(n, k + 1) \leq G(n - 1, k) + 2n, \quad G(n, 1) \leq 2n,$$

значит,

$$G(n, k + 1) \leq 2n + 2(n - 1) + \dots + 2(n - k) = (k + 1)(2n - k).$$

При $k = n$ получаем

$$G(n, n + 1) = n(n + 1).$$

Тот факт, что коэффициент при x^k во многочлене $p(x + c)$ равен $p^{(k)}(c)/k!$, следует из формулы Тейлора.

Впрочем, вот доказательство. Пусть

$$p(x + c) = c_n x^n + \dots + c_1 x + c_0.$$

Продифференцируем k раз обе части этого равенства. Используя правила дифференцирования, получаем, что

$$p^{(k)}(x + c) = k!c_k + (k + 1)!c_{k+1}x + \dots + n(n - 1) \dots (n - k + 1)c_n x^{n-k}.$$

Подставляя вместо x нуль, имеем $p^{(k)}(c) = k!c_k$. □

Следствие. Для одновременного вычисления значений многочлена и его производной достаточно $4n - 2$ операции, причем коэффициенты производной не вычисляются.

Теорема 36. Для одновременного вычисления значений $p(c)$ и $p(-c)$ достаточно $2n + 1$ операции, где n — степень многочлена $p(x)$.

Доказательство. Разделим $p(x)$ на $x^2 - c^2$. Получим равенство

$$p(x) = q(x)(x^2 - c^2) + r_1x + r_0.$$

Представляя $p(x)$ в виде суммы многочленов, содержащих только четные и только нечетные степени переменной, получим что $p(x) = p_0(x^2) + xp_1(x^2)$, где сумма степеней многочленов $p_0(x)$ и $p_1(x)$ равна $n - 1$. Деля их на $x - c^2$, имеем

$$p_0(x) = q_0(x)(x - c^2) + k_0, \quad p_1(x) = q_1(x)(x - c^2) + k_1.$$

Отсюда

$$\begin{aligned} q(x)(x^2 - c^2) + r_1x + r_0 &= p(x) = p_0(x^2) + xp_1(x^2) = \\ &= q_0(x^2)(x^2 - c^2) + k_0 + x(q_1(x^2)(x^2 - c^2) + k_1) = \\ &= (x^2 - c^2)(q_0(x^2) + x(q_1(x^2))) + k_1x + k_0, \end{aligned}$$

значит,

$$r_1 = k_1, \quad r_0 = k_0, \quad q(x) = q_0(x^2) + x(q_1(x^2)).$$

Так как вычисление r_i и коэффициентов $q(x)$ по числам k_i и коэффициентам $q_i(x)$ делается «бесплатно», то сложность деления на $x^2 - c^2$ не превосходит $2n - 2$. Для вычисления

$$p(c) = q(c)(c^2 - c^2) + rc + r = rc + r,$$

$$p(-c) = q(-c)(c^2 - c^2) + r_1(-c) + r_0 = -r_1c + r_0$$

нужно дополнительно 3 операции. □

Следствие. Если компьютер имеет два независимо работающих процессора, то сложность одновременного вычисления $p(c)$ и $p(-c)$ в 2 раза меньше, чем на однопроцессорной машине.

Рассмотренный алгоритм Горнера «второго порядка» является простейшим примером так называемого параллельного вычисления. К сожалению, не все алгоритмы можно «распараллелить». Но если это удастся, и технические возможности позволяют реализовать распараллеленный алгоритм, то получается существенная экономия времени.

Теорема 37 (параллельная схема Горнера). Если компьютер имеет k параллельно работающих процессоров, операции обмена между которыми занимают пренебрежимо малое время, то вычисление значения многочлена степени n в заданной точке требует

$$\frac{2n}{k} + \max_{j \leq k} l(j) + \log_2 k + O(1)$$

единиц времени.

Доказательство. Положим $n = mk + s$, $0 \leq s < k$, и представим $p(x)$ в виде

$$\sum_{j=0}^{k-1} x^j p_j(x^k) = \sum_{j=0}^s x^j \sum_{i=0}^m u_{ki+j} x^{ki} + \sum_{j=s+1}^{k-1} x^j \sum_{i=0}^{m-1} u_{ki+j} x^{ki}.$$

На j -м процессоре вычисляем $x^j p_j(x^k)$. Для этого на $(k-1)$ -м процессоре вычисляются со сложностью $l(k) + O(1)$ степени x^{k-1} и x^k и последняя передается на остальные процессоры, которые тем временем вычисляют степени x^j . После этого все процессоры вычисляют $x^j p_j(x^k)$ со сложностью $2n/k + O(1)$ каждый (используя обычную схему Горнера).

Остается сложить полученные результаты за время $\log_2 k + O(1)$, используя параллельно $k/2$ процессоров. \square

Задача 63. Докажите утверждение следствия, что для одновременного вычисления значений многочлена и его производной достаточно $4n - 2$ операции, причем коэффициенты производной не вычисляются.

Задача 64. Предложите быстрый способ вычисления многочленов, содержащих только четные или только нечетные степени переменной.

Глава 23

Что можно вычислить на счетах

Все это выглядит как чрезвычайно замедленная киностемка выполняемого человеком процесса вычисления. Разбор такой механизации вычисления на нескольких функциях помогает освоиться с мыслью, что и человек обычно считает совершенно так же, только быстрее.

Роза Петер¹⁾. Рекурсивные функции
(русский перевод 1954 г.)

Воспользовавшись такой простейшей моделью вычислений, как абак (в России называвшейся просто счетами), можно построить все здание современной теории алгоритмов. Разумеется, это понятие надо немного идеализировать и придать ему, например, следующий вид (как это было сделано американским математиком Ламбеком в начале 60-х гг. XX века, см. [1]).

Пусть нужно вычислить данную числовую функцию

$$f(x_1, \dots, x_n).$$

Представим себе, что у нас есть счеты, содержащие n «входных» спиц, на i -й из которых имеется в начальный момент x_i костяшек, одну «выходную» (первоначально пустую) спицу, на которой будет получен результат, и некоторое количество (первоначально пустых) «рабочих» спиц. Каждая спица состоит на самом деле из двух половин, и пока речь шла только о левых половинах. В правой половине каждой спицы помещается потенциально не ограни-

¹⁾ Роза Петер (Rosza Peter, 1905–1977) — венгерский математик. На русский язык переведена также ее научно-популярная книга «Игра с бесконечностью» (М., 1967).

ченный запас костяшек, и по нашему желанию мы можем сделать в любой момент одну из двух операций: либо передвинуть самую левую костяшку из правой половины спицы в «рабочую» левую половину, увеличив тем самым «записанное» в ней число на единицу, либо передвинуть крайнюю правую костяшку из «рабочей» левой половины спицы в правую «запасную» половину, уменьшив тем самым «записанное» в ней число на единицу.

Если перейти к терминологии языков программирования, то мы здесь описали систему регистров машины и две операции, применимые к ним — инкрементцию и декрементцию.

Сама программа вычисления на счетах (или на соответствующей идеализированной машине с неограниченными регистрами) представляет из себя диаграмму, состоящую из кружочков, в которых написаны номера спиц (регистров), после которых написаны знаки плюс или минус, указывающие на операции, которые мы выполняем на этих спицах (регистрах). Из кружочков со знаком плюс выходит одна стрелка, ведущая в какой-то другой кружочек (она указывает какую следующую операцию делать). Из кружочков со знаком минус выходит две такие стрелки. Одна из них помечается специальным значком * и используется только тогда, когда на «рабочей» половине спицы не осталось костяшек (в регистре записан ноль). Тогда операция декрементции, естественно, не может быть выполнена, и просто делается переход к новой вершине диаграммы по указанной стрелке. Если же на спице оставались костяшки (регистр не пуст), то операция декрементции выполняется и тоже делается переход к новой вершине диаграммы, но, естественно, по второй стрелке. Отметим еще, что совсем не обязательно, чтобы разные вершины диаграммы выполняли операции с разными спицами.

Теперь, чтобы такая диаграмма могла определить работающую программу, осталось выделить в ней две стрелки — начало и конец работы программы. Первая из них выделяется среди других стрелок тем, что имеет конец в одной из вершин диаграммы, но не имеет начала в вершинах диаграммы, а начинается в специальном кружке со словом *begin*, а вторая, наоборот, начинается

в одной из вершин, но не ведет ни в одну из вершин диаграммы, а ведет в кружок со словом *end*.

Программа начинает работать со слова *begin* и заканчивает, когда придет в слово *end* (но может и «зациклиться» и никогда не закончить работу). Результатом работы программы можно считать число, записанное на «выходном» регистре. Если это число всегда совпадает со значением рассматриваемой функции $f(x_1, \dots, x_n)$, в случае когда она определена при заданных значениях переменных, и если программа всегда «зацикливается», в случае когда эта функция не определена при заданных значениях переменных, то говорят, что программа (вычисления на счетах!) правильно вычисляет заданную функцию.



Алан Тьюринг

С целью сокращения диаграмм у сложных программ можно вместо некоторых вершин, имеющих одну выходную стрелку, использовать не оператор инкрементации, а кружок с символическим обозначением какой угодно программы (называемой в этом случае, естественно, подпрограммой).

Работу любой такой программы можно промоделировать и на машине Тьюринга, если изображать состояние абака в каждый момент времени на ленте машины в виде массивов палочек, разделенных пробелами. В возможность обратного моделирования поверить труднее, тем не менее справедливо следующее утверждение, приводимое без доказательства: класс числовых функций, вычислимых на абакe, совпадает с классом функций, вычислимых по Тьюрингу²⁾.

А как известно, на машине Тьюринга можно промоделировать любые вычисления. Значит, и на счетах тоже можно.

²⁾ Машина Тьюринга — умозрительное вычислительное устройство, предложенное в 1936 г. английским математиком Аланом Тьюрингом (1912–1954) в качестве базиса развитого им варианта построения теории алгоритмов, ставшего с тех пор общепринятым. Определение машины Тьюринга можно найти, например, в [1].

Задача 65. Приведите пример диаграммы, никогда не заканчивающей работы.

Задача 66. Приведите пример диаграммы, складывающей содержимое двух регистров и помещающей результат во второй регистр одновременно с обнулением первого регистра.

Задача 67. Приведите пример диаграммы, складывающей содержимое двух регистров и помещающей результат во второй регистр, но не изменяющей первый регистр (используйте вспомогательный «рабочий» регистр).

Задача 68. Приведите пример диаграммы, перемножающей содержимое двух регистров и помещающей результат в третий регистр одновременно с обнулением первого регистра (используйте предыдущую программу в качестве подпрограммы).

Задача 69. Приведите пример диаграммы, перемножающей содержимое двух регистров и помещающей результат во второй регистр без изменения первого регистра (используйте вспомогательный регистр и предыдущую программу в качестве подпрограммы).

Задача 70. Покажите, как возводить в степень на «счетах» (используйте предыдущую программу в качестве подпрограммы).

Литература

Библиография по теме нашей книги огромна, и приводимый далее список не претендует на полноту. В него включены некоторые источники, использованные автором при ее подготовке, а также расширяющие и дополняющие ее. Они выбраны из числа наиболее доступных в разных смыслах, в том числе и по времени издания.

1. Булос Д., Джеффри Р. Вычислимость и логика. М.: Мир, 1994.
2. Гашков С. Б., Чубариков В. Н. Арифметика, алгоритмы, сложность вычислений. М.: Дрофа, 2005.
3. Гашков С. Б. Системы счисления и их применения. М.: МЦНМО, 2004.
4. Гашков С. Б. Современная элементарная алгебра в задачах и упражнениях. М.: МЦНМО, 2006.
5. Кнут Д. Искусство программирования. Т. 2. М.: Вильямс, 2000.
6. Севидж Д. Сложность вычислений. М.: Факториал, 1998.
7. Уоррен Г., мл. Алгоритмические трюки для программистов. М.: Вильямс, 2004.

Другие книги нашего издательства:



URSS

Популярные книги по математике

- Пухначев Ю. В., Попов Ю. П.* Математика без формул. Кн. 1, 2.
Творогов В. Б. Наглядная арифметика и технология быстрого счета: Основы.
Пантаев М. Ю. Матанализ с человеческим лицом, или Как выжить после предельного перехода: Полный курс математического анализа. В 2 т.
Жуков А. В. Вездесущее число «пи».
Жуков А. В. Прометеева искра: Античные истоки искусства математики.
Жуков А. В. и др. Элегантная математика. Задачи и решения.
Зуев Ю. А. По океану дискретной математики: От перечислительной комбинаторики до современной криптографии. В 2 т.
Оре О. Приглашение в теорию чисел.
Гильберт Д., Кон-Фоссен С. Наглядная геометрия.
Яглом И. М. Математика и реальный мир.
Яглом И. М. Как разрезать квадрат?
Яглом И. М. Необыкновенная алгебра.
Амелькин В. В. Дифференциальные уравнения в приложениях.
Федин С. Н. Математики тоже шутят.
Ушаков И. А. История науки сквозь призму озарений. Кн. 1–8.

Серия «Знакомство с высшей математикой»

- Понтрягин Л. С.* Метод координат.
Понтрягин Л. С. Анализ бесконечно малых.
Понтрягин Л. С. Алгебра.
Понтрягин Л. С. Дифференциальные уравнения и их приложения.

Серия «Классический университетский учебник»

- Колмогоров А. Н., Драгалин А. Г.* Математическая логика.
Петровский И. Г. Лекции по теории обыкновенных дифференциальных уравнений.
Гнеденко Б. В. Курс теории вероятностей.
Кононович Э. В., Мороз В. И. Общий курс астрономии.
Ишханов Б. С., Капитонов И. М., Юдин Н. П. Частицы и атомные ядра.
Каасников И. А. Термодинамика и статистическая физика. В 4 т.

Наши книги можно приобрести в магазинах:

Тел./факс:
 +7 (499) 724-25-45
 (многоканальный)

E-mail:
 URSS@URSS.ru
<http://URSS.ru>

«НАУКУ – ВСЕМ!» (м. Профсоюзная, Нахимовский пр-т, 56. Тел. (499) 724-2545)
 «Библио-Глобус» (м. Лубянка, ул. Мясницкая, 6. Тел. (495) 625-2457)
 «Московский дом книги» (м. Арбатская, ул. Новый Арбат, 8. Тел. (495) 203-8242)
 «Молодая гвардия» (м. Полинка, ул. Б. Полянка, 28. Тел. (495) 238-5001, 780-3370)
 «Дом научно-технической книги» (Ленинский пр-т, 40. Тел. (495) 137-6019)
 «Дом книги на Ладомской» (м. Бауманская, ул. Ладомская, 8, стр. 1. Тел. 267-0302)
 «СПб. дом книги» (Невский пр., 28. Тел. (812) 448-2355)
 «100 000 книг» (г. Екатеринбург, ул. Тургенева, 13. Тел. (343) 22-12-979)
 Сеть магазинов «Дом книги» (г. Екатеринбург, ул. Антона Валека, 12. Тел. (343) 253-50-10)

Уважаемые читатели! Уважаемые авторы!

Наше издательство специализируется на выпуске научной и учебной литературы, в том числе монографий, журналов, трудов ученых Российской академии наук, научно-исследовательских институтов и учебных заведений. Мы предлагаем авторам свои услуги на выгодных экономических условиях. При этом мы берем на себя всю работу по подготовке издания — от набора, редактирования и верстки до тиражирования и распространения.



URSS

Среди вышедших и готовящихся к изданию книг мы предлагаем Вам следующие:

Серия «НАУКУ — ВСЕМ! Шедевры научно-популярной литературы»

Гашков С. Б. Занимательная компьютерная арифметика. Кн. 1, 2.

Меннихен Ф. Некоторые тайны артистов-вычислителей.

Вильямс Дж. Д. Совершенный стратег, или Букварь по теории стратегических игр.

Юдин Д. Б., Юдин А. Д. Математики измеряют сложность.

Колмогоров А. Н. Математика — наука и профессия.

Босс В. Интуиция и математика.

Нагель Э., Ньюмен Дж. Р. Теорема Гёделя.

Гнеденко Б. В. Беседы о теории массового обслуживания.

Гнеденко Б. В. Беседы о математической статистике.

Гнеденко Б. В., Хинчин А. Я. Элементарное введение в теорию вероятностей.

Мизес Р. Вероятность и статистика.

Вольберг О. А. Основные идеи проективной геометрии.

Щербаков Р. Н., Пичурин Л. Ф. От проективной геометрии — к неевклидовой.

Щербаков Р. Н., Пичурин Л. Ф. Дифференциалы помогают геометрии.

Стирод Н., Чинн У. Первые понятия топологии.

Колягин Ю. М., Саркисян А. А. Познакомьтесь с топологией: На подступах к топологии.

Широков П. А. Краткий очерк основ геометрии Лобачевского.

Меньчуков А. Е. В мире ориентиров.

Фрова А. Почему происходит то, что происходит: Окружающий мир глазами ученого.

Уле О. Почему и потому: Учебник физики в вопросах и ответах.

Гартман З. Занимательная физика, или Физика во время прогулки.

Ланге В. Н. Физические парадоксы, софизмы и занимательные задачи. Кн. 1, 2.

Ланге В. Н. Физические опыты и наблюдения в домашней обстановке.

Перельман М. Е. А почему это так? Физика вокруг нас.

Перельман М. Е. А почему это так? Физика в гостях у других наук.

Перельман М. Е. Наблюдения и озарения, или Как физики выявляют законы природы.

Закейтис А. Ю. Системность — симметрия — эволюция в физике, химии, биологии.

Покровский В. В. Космос, Вселенная, теория всего почти без формул,

или Как дошли до теории суперструн.

По всем вопросам Вы можете обратиться к нам:
тел. +7 (499) 724-25-45 (многоканальный)
или электронной почтой URSS@URSS.ru
Полный каталог изданий представлен
в интернет-магазине: <http://URSS.ru>

Научная и учебная
литература

117335, Москва,
Нахимовский пр-т, 56



URSS

НАШИ НОВЫЕ КООРДИНАТЫ

ТЕЛЕФОН / ФАКС (многоканальный)
+7 (499) 724-25-45

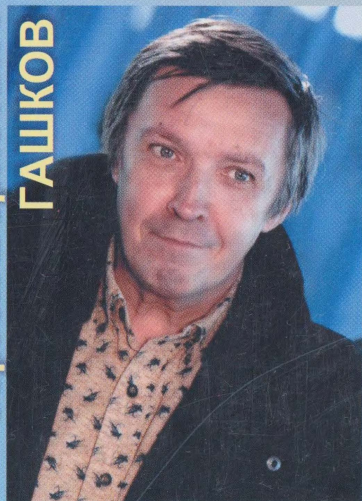


От м. Профсоюзная:

8 мин. пешком
или одна остановка
наземным транспортом:
автобусы № 67, 67к, 130;
троллейбус № 49
до остановки
«Ул. Ивана Бабушкина»

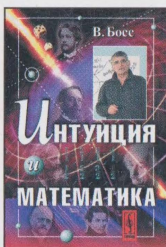
От м. Университет:

трамваи № 14, 39
до остановки
«Черемушкинский рынок»;
трамваи № 22, 26
до остановки
«Ул. Вавилова»;
автобусы № 67, 67г, 130;
троллейбус № 49
до остановки
«Ул. Ивана Бабушкина»



Доктор физико-математических наук, профессор кафедры дискретной математики МГУ им. М. В. Ломоносова. Автор и соавтор книг «Примени математику», «Арифметика. Алгоритмы. Сложность вычислений», «Системы счисления и их применения», «Современная элементарная алгебра», «Элементарное введение в эллиптическую криптографию», «Криптографические методы защиты информации».

Наше издательство предлагает следующие книги:



10585 ID 124753



Отзывы о настоящем издании, а также обнаруженные опечатки присылайте по адресу URSS@URSS.ru. Ваши замечания и предложения будут учтены и отражены на web-странице этой книги в нашем интернет-магазине <http://URSS.ru>



URSS

E-mail: URSS@URSS.ru
Каталог изданий в Интернете: <http://URSS.ru>

URSS НАШИ НОВЫЕ
КООРДИНАТЫ

ТЕЛЕФОН/ФАКС +7 (499) 724-25-45
(многоканальный)
117335, Москва, Нахимовский пр-т, 56