ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

И.А. Панкратова

ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ

Учебное пособие

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

И.А. Панкратова

ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ

Учебное пособие

Томск
2009

УДК 519.7
П 16

Панкратова И.А.
П 16 Теоретико-числовые методы криптографии: Учебное пособие. – Томск: Томский государственный университет, 2009. – 120 с.

Излагаются алгоритмы решения некоторых теоретико-числовых задач, возникающих в криптографических приложениях.

Для студентов, аспирантов, научных работников и преподавателей университетов, знакомых с основами криптографии, теории чисел и общей алгебры

УДК 519.7

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Данное учебное пособие представляет собой конспект лекций по дисциплинам «Вычислительные методы в теории чисел и общей алгебре» и «Теоретико-числовые методы в криптографии», которые автор в течение ряда лет читает в Томском государственном университете студентам специальности «Компьютерная безопасность». Основной (и практически единственный на момент введения этих дисциплин в ТГУ) источник материала – это замечательное руководство по прикладной криптографии А. Менезеса, П. Ооршота и С. Ванстона [20]. В нём приводятся все необходимые для обоснования излагаемых алгоритмов математические понятия и теоремы, каждый алгоритм сопровождается примерами и обширными комментариями, но отсутствие доказательств большинства теорем затрудняет использование руководства в качестве учебного пособия. Некоторые алгоритмы работы с большими числами представлены также в фундаментальном труде Д. Кнута «Искусство программирования» [7]. В последние годы появилось много книг, в том числе учебников, посвящённых теоретико-числовым аспектам криптографии [4–6, 8–10, 12, 16–18], однако в силу небольших тиражей они не стали легко доступными, по крайней мере, в Томске. Кроме того, ни одна из этих книг в отдельности не охватывает всех тем, составляющих содержание рассматриваемых дисциплин.

Учебное пособие состоит из пяти глав. *Первая глава* посвящена алгоритмам над большими числами, которые не представимы в ЭВМ с помощью стандартных типов данных. Поэтому даже простейшие операции, такие как сложение, вычитание, умножение и деление больших чисел, приходится программировать. Две основные задачи, рассматриваемые здесь, – это возведение в степень по модулю (операция, чрезвычайно широко используемая в различных криптографических приложениях) и быстрое умножение чисел. Алгоритмы быстрого умножения, может быть, на данный момент имеют не столь большое практическое значение (для чисел актуального для криптографии размера в 1000–2000 битов вполне работоспособен алгоритм умножения в столбик), но они поражают красотой идей и той математики, на которых построены. Кроме этих основных, приводятся алгоритмы решения двух вспомогательных задач – нахождения наибольшего общего делителя двух чисел и целой части квадратного корня из числа. Во *второй главе* рассматриваются методы гене-

рации простых чисел. Важность этой задачи для криптографии трудно переоценить, так как простые числа являются параметрами многих криптосистем с открытым ключом. Изучаются методы разных типов, доставляющие «вероятно простые» числа, «доказуемо простые» числа, а также простые числа специального вида. *Третья* и *четвёртая главы* посвящены двум фундаментальным проблемам, на сложность которых полагаются криптосистемы с открытым ключом, – задачам факторизации и дискретного логарифмирования. Приводятся девять алгоритмов факторизации и пять методов логарифмирования и обсуждаются достоинства, недостатки и сферы эффективного применения каждого из них. В *пятой главе* изучаются алгоритмы над полиномами, заданными над конечным полем. Рассматриваются задачи проверки полиномов на неприводимость и на примитивность, которые приходится решать при построении регистровых поточных шифров, а также задача факторизации полиномов.

В пособии автор стремился сочетать математическую строгость изложения с практической направленностью. Ко всем алгоритмам даются формальное обоснование, чаще всего в виде теорем с доказательствами, и примеры на данных небольшого (пригодного для ручного счёта) размера; многие алгоритмы сопровождаются рекомендациями по их программной реализации. Главы учебного пособия можно изучать в любом порядке независимо друг от друга; предполагается знание читателем базовых университетских курсов общей алгебры и теории чисел. Некоторые необходимые определения и теоремы из них приводятся в приложении. Желательно также знакомство, хотя бы начальное, с основными понятиями криптографии.

Автор искренне надеется, что данное учебное пособие будет полезно всем, кто интересуется теоретико-числовыми и алгоритмическими аспектами криптографии. Для более глубокого изучения затронутых здесь вопросов можно порекомендовать книги [4, 12]. Информация о возможных ошибках и опечатках, а также критические замечания по содержанию и оформлению пособия будут с благодарностью приняты по адресу pank@isc.tsu.ru. Автор выражает признательность Геннадию Петровичу Агибалову за помощь при написании пособия, а также студентам кафедры защиты информации и криптографии ТГУ, чьи вопросы и замечания во время лекций способствовали устранению ряда неточностей в изложении.

1. АЛГОРИТМЫ НАД БОЛЬШИМИ ЧИСЛАМИ

Введение

В современных асимметричных криптосистемах в качестве ключей, как правило, используются целые числа длиной 1000 и более битов. Для задания чисел такого размера не подходит ни один стандартный целочисленный тип данных современных языков программирования. Представление чисел в формате с плавающей точкой позволяет задать очень большие числа (например, тип long double языка C++ – до 10^{5000}), но не удовлетворяет требованию абсолютной точности, характерному для криптографических приложений. Поэтому большие целые числа представляются в криптографических пакетах в виде последовательности цифр в некоторой системе счисления (обозначим основание системы счисления b; о его выборе пойдет речь ниже):

$$x = (x_{n-1}\, x_{n-2} \ldots x_1\, x_0)_b,$$

где $0 \le x_i < b$ для $i = 0, \ldots, n-1$.

Основание системы счисления b выбирается так, чтобы существовали машинные команды для работы с однозначными и двузначными числами; как правило, b равно 2^8, 2^{16} или 2^{32}. В этой главе описываются арифметические алгоритмы работы с большими числами: возведение в квадрат, приведение по модулю, возведение в степень, возведение в степень по модулю, вычисление наибольшего общего делителя, извлечение квадратного корня, быстрое умножение. За рамками изложения остались операции сложения, вычитания, умножения в столбик, нахождения неполного частного и остатка, которые применительно к большим числам тоже приходится программировать и подробное описание алгоритмов выполнения которых можно найти в прекрасной книге Д. Кнута «Искусство программирования для ЭВМ» (Т. 2. Получисленные алгоритмы).

Некоторые понятия о сложности алгоритмов. Алгоритмы обычно оцениваются по двум параметрам – время работы, или количество операций, и объем требуемой памяти. Под операцией понимается любое действие, время выполнения которого не зависит от размера данных.

Пусть n – размер данных, от которого зависит время работы алгоритма – $t(n)$. В арифметических алгоритмах это количество цифр в числе: $n = [\log_b x] + 1$. Существуют два больших класса алгоритмов. Если $t(n) = O(n^c)$, где $c = \text{const}$, то алгоритм называется полиномиальным. Если

же $t(n) = O(c^n)$, где $c = \text{const}$ и $c > 1$, то алгоритм называется экспоненциальным. Полиномиальные алгоритмы считаются быстрыми (выполнимыми за реальное время), экспоненциальные – медленными (практически невыполнимыми), хотя для небольших значений n время работы первых может превосходить, и значительно, время работы вторых.

Примеры сложности алгоритмов:

– сложение/вычитание чисел: $t(n) = O(n)$ – полиномиальный (линейный) алгоритм;

– умножение чисел в столбик: $t(n) = O(n^2)$ – полиномиальный (квадратичный) алгоритм;

– проверка числа x на простоту методом пробных делений на числа, меньшие $\sqrt{x}$: $t(n) = O(x^{1/2}) = O(b^{n/2}) = O((\sqrt{b})^n)$ – экспоненциальный алгоритм.

1.1. Быстрое возведение в квадрат

Если $x = (x_{n-1} \dots x_0)_b$, то вычисление x^2 с помощью умножения в столбик потребует n^2 операций (умножений цифр – каждой на каждую). Если учесть повторения цифр в сомножителях, то количество операций можно сократить. Рассмотрим эту идею на примере двузначного числа. Пусть $x = (x_1 x_0)_b = x_1 b + x_0$. Тогда $x^2 = x_1^2 b^2 + 2bx_1x_0 + x_0^2$, что вычисляется за три умножения цифр, поскольку умножение на b (b^2) – это сдвиг числа на одну (две) цифру влево, умножение на 2 при $b = 2^k$ – сдвиг на один бит влево, при $b \neq 2^k$ – одна операция сложения.

Алгоритм быстрого возведения числа в квадрат.

Вход: $x = (x_{n-1} \dots x_0)_b$.

Выход: $y = (y_{2n-1} \dots y_0)_b = x^2$.

Обозначения: u, v – цифры в системе счисления с основанием b, $c \in \{0, 1\}$ – бит переноса, $(de)_b$ – двузначные числа, $(cde)_b$ – трёхзначные.

1. Для i от 0 до $2n - 1$ положить $y_i = 0$.
2. Для i от 0 до $n - 1$

2.1. Вычислить $(uv)_b = y_{2i} + x_i \cdot x_i$; положить $y_{2i} = v$, $c = 0$.

2.2. Для j от $i + 1$ до $n - 1$ вычислить $(cuv)_b = y_{i+j} + 2\, x_i \cdot x_j + (cu)_b$; положить $y_{i+j} = v$.

2.3. Число $(y_{i+n+1}\, y_{i+n})_b$ увеличить на оставшийся перенос $(cu)_b$.

3. Ответ: y.

Обоснование алгоритма. Докажем, что в шаге 2.1 достаточно действий с двойной точностью, а в шаге 2.2 старшая цифра (c) – это один бит.

Шаг 2.1: $y_{2i} \le b - 1$, $x_i \le b - 1$, следовательно,

$$y_{2i} + x_i \cdot x_i \le b - 1 + (b - 1)^2 = b(b - 1) < b^2.$$

Шаг 2.2: докажем по индукции. На первой итерации до вычисления выражения $c = 0$,

$$y_{i+j} + 2\,x_i x_j + (cu)_b \le b - 1 + 2(b-1)^2 + b - 1 = 2b^2 - 2b < 2b^2,$$

следовательно, после вычисления $c \in \{0, 1\}$. На всех последующих итерациях аналогично:

$$y_{i+j} + 2\,x_i x_j + (cu)_b \le b - 1 + 2(b-1)^2 + b + b - 1 = 2b^2 - b < 2b^2.$$

Оценка сложности алгоритма

В шаге 2.1 выполняется n раз по одному умножению.

В шаге 2.2: $(n-1) + (n-2) + \ldots + 1 = n(n-1)/2$ умножений.

Итого $n + n(n-1)/2 = n(n+1)/2$ умножений.

Замечания по реализации:

1. Шаг 2.1 выполняется с помощью стандартного типа данных двойного по отношению к b размера.

2. Шаг 2.2 выполняется с помощью стандартного типа данных тройного (обычно учетверённого) размера, если таковой имеется для выбранных языка программирования и основания системы счисления b. В противном случае вычисления ведутся с использованием двузначных чисел, отдельно определяется и хранится бит переноса c.

3. В шаге 2.3 на последней итерации при $i = n - 1$ предписывается работа с цифрой y_{2n}, которая в числе y не предусмотрена. Легко проверяется, что эта цифра всегда получится равной 0. Возможны два выхода из этой ситуации: проверять всякий раз в шаге 2.3, не последняя ли это итерация, либо выделить в памяти для числа y на одну цифру больше, чем требуется. С точки зрения эффективности алгоритма второй способ представляется более предпочтительным.

Пример. Пусть $b = 10$, $n = 3$, $x = 999$.

Шаг	i	j	c	u	v	y_5	y_4	y_3	y_2	y_1	y_0
1						0	0	0	0	0	0
2.1	0		0	8	1						1
2.2		1	1	7	0					0	
		2	1	7	9				9		
2.3							1	7			
2.1	1		0	9	0				0		
2.2		2	1	7	8			8			
2.3						1	8				
2.1	2		0	8	9		9				
2.3						9					

Ответ: $x^2 = 998001$.

1.2. Дихотомический алгоритм возведения в степень

Задача ставится так: для данных x и y найти x^y. Решить задачу можно с помощью y последовательных умножений на x; получим алгоритм экспоненциальной сложности. Однако уже в XV в. великим вычислителем аль-Каши был сформулирован метод, называемый дихотомическим, или бинарным, методом возведения в степень. Идея метода предельно проста.

Зададим показатель y в двоичной системе счисления (отсюда – название метода): $y = (y_{n-1} \dots y_0)_2 = \sum_{i=0}^{n-1} y_i \cdot 2^i$. Тогда

$$x^y = \prod_{0 \le i < n} x^{y_i \cdot 2^i} = \prod_{\substack{0 \le i < n, \\ y_i = 1}} x^{2^i}.$$

Например, $9 = (1001)_2$; $x^9 = x^8 \cdot x^1$. Значение x^{2^i} вычисляется последовательным i-кратным возведением числа в квадрат.

Алгоритм возведения в степень (вариант «справа налево»)

Вход: $x, y = (y_{n-1} \dots y_0)_2$.

Выход: $z = x^y$.

1. Положить $q = x$. Положить $z = x$, если $y_0 = 1$, и $z = 1$, если $y_0 = 0$.
2. Для i от 1 до $n - 1$
2.1. Положить $q = q^2$.
2.2. Если $y_i = 1$, то положить $z = z \cdot q$.
3. Ответ: z.

Пример 1. Пусть $y = 78 = (1001110)_2$.

i	0	1	2	3	4	5	6
y_i	0	1	1	1	0	0	1
q	x	x^2	x^4	x^8	x^{16}	x^{32}	x^{64}
z	1	x^2	x^6	x^{14}			x^{78}

Оценка сложности алгоритма

В шаге 2.1 $(n - 1)$ раз выполняется возведение в квадрат.

В шаге 2.2 $(w(y) - y_0)$ раз выполняется умножение, где $w(y)$ – вес числа y в двоичном представлении.

Поскольку $w(y) \le n$, то $t(n) \le 2n - 1$. Заметим, что на данный момент алгоритмов возведения в степень с лучшей оценкой не существует, хотя в некоторых частных случаях можно добиться сокращения числа операций. Например, значение x^{15} нашим алгоритмом будет вычислено за 3 возведения в квадрат и 3 умножения. Можно предложить другую схему, последовательно получая x^2, $x^3 = x^2 \cdot x$, $x^6 = (x^3)^2$, $x^{12} = (x^6)^2$, и наконец, $x^{15} = x^{12} \cdot x^3$ – всего 3 возведения в квадрат и 2 умножения.

Рассмотренный вариант алгоритма называют вариантом «справа налево» – по порядку просмотра разрядов в показателе y. Очевидно, того же результата можно добиться при обратном порядке просмотра. Будем на каждом шаге возводить число z в квадрат и, в случае $y_i = 1$, домножать его на x.

Пример 2. Пусть, как и в примере 1, $y = 78 = (1001110)_2$.

i	6	5	4	3	2	1	0
y_i	1	0	0	1	1	1	0
z	x	x^2	x^4	x^9	x^{19}	x^{39}	x^{78}

Алгоритм возведения в степень (вариант «слева направо»).

Вход: $x, y = (y_{n-1} \dots y_0)_2$, причем $y_{n-1} = 1$.

Выход: $z = x^y$.

1. Положить $z = x$.
2. Для i от $n - 2$ до 0

2.1. Положить $z = z^2$.

2.2. Если $y_i = 1$, то положить $z = z \cdot x$.

3. Ответ: z.

Заметим, что оценка числа необходимых операций в варианте «слева направо» та же, что и в первом варианте алгоритма. Существенное отличие второго варианта от первого состоит в том, что в шаге 2.2 умножение всегда происходит на одно и то же число x, что может дать выигрыш в эффективности в случае малого x.

В криптографических приложениях чаще всего приходится вычислять не x^y, а $x^y \bmod m$, т.е. остаток от деления x^y на m. Понятно, что вычисление x^y и деление полученного числа на модуль m не эффективно ввиду слишком большого значения x^y. Для решения задачи возведения в степень по модулю в алгоритме «справа налево» шаг 2 модифицируется следующим образом:

2.1. Положить $q = q^2 \bmod m$.

2.2. Если $y_i = 1$, то положить $z = z \cdot q \bmod m$.

Аналогичной модификации подвергается вариант «слева направо».

Задача возведения в степень по модулю очень широко используется как в криптосистемах с открытым ключом (при генерации их параметров и зачастую непосредственно при шифровании и расшифровании), так и при решении других задач (в частности, во многих алгоритмах факторизации и дискретного логарифмирования). Ввиду её важности этой задаче уделяется большое внимание в литературе. В следующих трёх разделах настоящего пособия излагаются два эффективных алгоритма приведения чисел по модулю и некое «синтетическое» преобразование (метод Монтгомери), позволяющее вычислять $xy \bmod m$, не получая в явном виде произведения xy.

1.3. Алгоритм Барретта приведения чисел по модулю

Постановка задачи: для заданных положительных чисел x, m найти $x \bmod m$ – остаток от деления x на m. Тривиальные способы решения:

1) с помощью операции деления найти остаток в виде $r = x \bmod m = x - qm$, где q – неполное частное: $q = [x/m]$;

2) найти остаток с помощью цепочки вычитаний: пока $x \geq m$, уменьшать x на m.

В первом способе используется одна, но сложная операция (деление), во втором – операции простые, но их количество может оказаться неприемлемо большим. Как это часто бывает, эффективный способ решения задачи состоит в разумном сочетании этих двух подходов.

Идея алгоритма Барретта: попытаться «угадать» значение q, т. е. найти некоторым способом q', примерно равное q. После этого вычисляется $r' = x - q'm$; заметим, что $r' = x \pmod m$. Для получения результата осталось добиться выполнения условия $0 \leq r' < m$, что достигается несколькими вычитаниями, как в способе 2. При хорошем приближении q' к q вычитаний будет немного.

Будем рассматривать случай $b^{k-1} < m < b^k$, т. е. $m = (m_{k-1} \ldots m_0)_b$ и $m_{k-1} \neq 0$. Заметим, что модуль $m = b^i$ не представляет интереса, поскольку значение $x \bmod b^i$ находится просто выделением i младших цифр числа x. Алгоритм Барретта работает в следующих ограничениях: $b > 3$, $|x| \leq 2 \cdot |m|$, где $|a|$ – количество цифр в числе a. Эти ограничения не доставляют никаких неприятностей на практике: конечно, реально всегда применяется $b > 3$, и при использовании алгоритма Барретта в задаче возведения в степень по модулю второе ограничение тоже автоматически выполняется, так как x есть результат произведения двух приведённых по модулю m чисел. Для работы алгоритма Барретта требуется предварительно вычислить значение $z = [b^{2k}/m]$, которое зависит только от m и подаётся на вход алгоритма.

Алгоритм Барретта приведения числа по модулю

Вход: $m = (m_{k-1} \ldots m_0)_b$, где $m_{k-1} \neq 0$, $b > 3$;
$x = (x_{n-1} \ldots x_0)_b$, $n \leq 2k$;
$z = [b^{2k}/m]$.

Выход: $r = x \bmod m$.

1. Вычислить $q' = [([x/b^{k-1}] \cdot z) / b^{k+1}]$.
2. Вычислить $r_1 = x \bmod b^{k+1}$, $r_2 = (q' \cdot m) \bmod b^{k+1}$.
3. Если $r_1 \geq r_2$, то положить $r' = r_1 - r_2$, иначе $r' = b^{k+1} + r_1 - r_2$.
4. Пока $r' \geq m$, уменьшать r' на m.

5. Ответ: $r = r'$.

Пример. Пусть $b = 10$, $x = 1234$, $m = 20$.

Вычисляем $z = [10^4/20] = 500$.

1. $q' = [([1234/10^1] \cdot 500) / 10^3] = 61$.
2. $r_1 = 1234 \bmod 10^3 = 234$, $r_2 = 61 \cdot 20 \bmod 10^3 = 220$.
3. $r' = 234 - 220 = 14$.
4. Условие $r' < m$ выполнено.
5. Ответ: $1234 \bmod 20 = 14$.

Обоснование алгоритма

1. Докажем, что q' – достаточно хорошее приближение к q, а именно $q - 2 \le q' \le q$.

Напомним, что $[a]$ обозначает целую часть (ближайшее целое снизу) числа a, $\{a\} = a - [a]$ – его дробную часть. Имеем

$$q = \left[\frac{x}{m}\right] = \left[\frac{x}{b^{k-1}} \cdot \frac{b^{2k}}{m} \cdot \frac{1}{b^{k+1}}\right] = [d \cdot e \cdot f],$$

где $d = \dfrac{x}{b^{k-1}}$, $e = \dfrac{b^{2k}}{m}$, $f = \dfrac{1}{b^{k+1}}$.

В этих обозначениях $q' = [[d] \cdot [e] \cdot f]$, откуда сразу ввиду $[a] \le a$ следует $q' \le q$. Вернемся к равенству для q:

$$q = [d \cdot e \cdot f] = [([d] + \{d\}) \cdot ([e] + \{e\}) \cdot f] = [[d] \cdot [e] \cdot f + d \cdot \{e\} \cdot f + \{d\} \cdot [e] \cdot f].$$

В последнем выражении во втором слагаемом ввиду ограничения $|x| \le 2 \cdot |m|$ выполняется $d \cdot f = \dfrac{x}{b^{2k}} < 1$ и, следовательно, $d \cdot \{e\} \cdot f < 1$; в третьем слагаемом $e \cdot f = \dfrac{b^{k-1}}{m} < 1$, значит, и $\{d\} \cdot [e] \cdot f < 1$. Из строгого неравенства $a < c$ следует нестрогое неравенство $[a] \le [c]$. С учётом всего этого получаем $q \le [[d] \cdot [e] \cdot f + 2] = [[d] \cdot [e] \cdot f] + 2 = q' + 2$.

2. Докажем, что для r', получаемого в шаге 3, выполняется сравнение $r' = x \pmod m$. По построению

$$r' = x - q' \cdot m \pmod{b^{k+1}}. \qquad (1)$$

Но $x - q' \cdot m = (q \cdot m + r) - q' \cdot m = (q - q') \cdot m + r < 3m < b^{k+1}$; здесь первое неравенство выполняется вследствие только что доказанного $q - q' \le 2$ и выполнения условия $r < m$; второе неравенство верно ввиду $b > 3$ и $m < b^k$. Поскольку правая часть сравнения (1) меньше модуля, то имеем просто равенство: $r' = x - q' \cdot m$, откуда следует $r' = x \pmod m$.

3. В предыдущем пункте показано, что $r' < 3m$. Это означает, что в шаге 4 вычитание выполняется не более двух раз.

Замечания по реализации:

1. Алгоритм Барретта имеет смысл применять только в тех случаях, когда много раз необходимо приводить различные числа по одному и тому же модулю, потому что в качестве предвычисления здесь выполняется операция деления, которая сама по себе решает задачу приведения по модулю. В частности, алгоритм Барретта имеет смысл использовать при возведении в степень по модулю.

2. Операции деления на b^l (приведения по модулю b^l), многократно использующиеся в алгоритме, реализуются с помощью отбрасывания (выделения) младших l цифр в числе.

1.4. Приведение по модулю специального вида

Рассмотрим случай, когда модуль имеет вид $m = b^l - c$, где $c < b^l$ и c мало. В этом случае можно предложить алгоритм приведения числа x по модулю m, не содержащий операций деления. Идея алгоритма состоит в следующем. Представим x в виде

$$x = q_0 \cdot b^l + r_0.$$

Это представление находится просто: число r_0 состоит из l младших цифр числа x, а q_0 – из остальных. Подставив значение $b^l = m + c$, получим:

$$x = q_0 \cdot m + q_0 \cdot c + r_0 = q_0 \cdot c + r_0 \pmod m. \qquad (1)$$

Далее то же представление найдём для $q_0 \cdot c$:

$$q_0 \cdot c = q_1 \cdot b^l + r_1 = q_1 \cdot m + q_1 \cdot c + r_1 = q_1 \cdot c + r_1 \pmod m. \qquad (2)$$

Из (1) и (2) следует равенство

$$x = q_1 \cdot c + r_1 + r_0 \pmod m.$$

Продолжим те же действия для $q_1 \cdot c$, и так далее, пока на очередном шаге не получим $q_s = 0$. Такой момент неизбежно наступит, поскольку из условий $q_i \cdot c = q_{i+1} \cdot b^l + r_{i+1}$ и $c < b^l$ следует $q_{i+1} < q_i$, а последовательность целых неотрицательных чисел не может убывать бесконечно. Сумма r_i, полученных на всех шагах, будет сравнима с x по модулю m, и нужный результат найдём с помощью последовательного уменьшения суммы на m.

Алгоритм приведения числа по модулю специального вида

Вход: x, $m = b^l - c$.

Выход: $r = x \bmod m$.

1. Представить x в виде $x = q \cdot b^l + r$.
2. Пока $q > 0$, выполнять:
2.1. Вычислить $q \cdot c$ и представить в виде $q \cdot c = q' \cdot b^l + r'$.
2.2. Положить $r = r + r'$.
2.3. Положить $q = q'$.

3. Пока $r \geq m$, уменьшать r на m.
4. Ответ: r.

Пример. Пусть $b = 10$, $m = 899 = 10^3 - 101$, $x = 2525$.

1. $2525 = 2 \cdot 10^3 + 525$.
2.1. $q \cdot c = 2 \cdot 101 = 202 = 0 \cdot 10^3 + 202$.
2.2. $r = 525 + 202 = 727$.
2.3. $q = 0$.
3. Условие $r < m$ выполнено.
4. Ответ: 727.

1.5. Метод Монтгомери

Вспомним дихотомический алгоритм возведения в степень по модулю. На каждом шаге этого алгоритма надо находить $q^2 \bmod m$ и, возможно, $z \cdot q \bmod m$. Стандартное вычисление этих выражений состоит в выполнении двух операций – возведении в квадрат (или умножении) и последующем приведении результата по модулю. Идея метода Монтгомери состоит в объединении этих операций в единое целое, с тем чтобы получать, например, $z \cdot q \bmod m$ без явного вычисления произведения $z \cdot q$.

Рассмотрим сначала **преобразование Монтгомери**, решающее задачу приведения по модулю без деления. Это преобразование выглядит достаточно сложным для понимания, хотя основано на следующих двух очевидных фактах:

1) $x \bmod m = (x + u \cdot m) \bmod m$ для любого u;

2) точный результат от деления числа, которое оканчивается на 0, на основание системы счисления b получается отбрасыванием этого 0.

Попробуем понять суть преобразования на примерах.

Пример 1. Пусть $b = 2$, $x = 99$, $m = 15$.

Будем всякий раз добиваться того, чтобы рассматриваемое число было чётным (прибавляя к нему, если надо, модуль 15), после чего полученное чётное число точно и легко делить на $b = 2$.

99 mod 15 = (99 + 15) mod 15;
114 / 2 = 57;
57 mod 15 = (57 + 15) mod 15;
72 / 2 = 36;
36 / 2 = 18;
18 / 2 = 9.

Преобразования в первой и третьей строках не влияют на результат приведения по модулю 15. Четырёхкратное деление на 2 фактически оз-

начает вычисление вместо $99 \bmod 15$ выражения $99 \cdot 2^{-4} \bmod 15 = 9$, что совпало с правильным результатом $99 \bmod 15 = 9$ по счастливой случайности – ввиду равенства $2^{-4} \bmod 15 = 1$.

Пример 2. Пусть $b = 10$, $x = 203$, $m = 11$.

Как и в прошлом примере, будем добиваться точной делимости числа на 10, а именно подбирать множитель u таким образом, чтобы последняя цифра числа $203 + 11u$ была равна 0.

$203 \bmod 11 = (203 + 7 \cdot 11) \bmod 11 = 280 \bmod 11$;

$280 / 10 = 28$;

$28 \bmod 11 = (28 + 2 \cdot 11) \bmod 11 = 50 \bmod 11$;

$50 / 10 = 5$.

В этом примере фактически мы посчитали $203 \cdot 10^{-2} \bmod 11$, что в силу равенства $10^{-2} \bmod 11 = 1$ совпало с правильным результатом $203 \bmod 11 = 5$. Преобразование Монтгомери состоит в вычислении $x \cdot b^{-k} \bmod m$, что парадоксальным образом выполняется проще нужного нам вычисления $x \bmod m$; здесь k – количество цифр в модуле m. Конечно, далеко не всегда выполняется равенство $b^{-k} \bmod m = 1$, и на первый взгляд такое преобразование решает не ту задачу, но... Немного терпения, скоро мы увидим пользу такого преобразования для достижения поставленной нами перед собой глобальной цели – возведения в степень по модулю.

Определение. Преобразованием Монтгомери величин x, R и m называется следующее выражение: $\mathrm{MR}(x, R, m) = x \cdot R^{-1} \bmod m$.

Очевидно, для корректности этого определения, т. е. для существования $R^{-1} \bmod m$, должно в силу *теоремы обратимости* выполняться условие $(R, m) = 1$.

На практике в качестве R используется b^k; в этом случае условие $(R, m) = 1$ равносильно условию $(b, m) = 1$, что для $b = 2^s$ означает требование нечётности m. Решим теперь технический вопрос – как добиться равенства 0 младшей цифры в выражении $x + u \cdot m$? Для этого рассмотрим сравнение

$x + u \cdot m = 0 \pmod b$,

что равносильно $\quad x_0 + u \cdot m = 0 \pmod b$,

откуда $\quad u = x_0 \cdot (-m^{-1}) \pmod b$.

Алгоритм преобразования Монтгомери

Вход: $m = (m_{k-1} \ldots m_0)_b$ с условием $(m, b) = 1$;

$m' = -m^{-1} \bmod b$;

$x = (x_{2k-1} \ldots x_0)_b$ такой, что $x < m \cdot b^k$.

Выход: $y = \mathrm{MR}(x, b^k, m) = x \cdot b^{-k} \bmod m$.

1. Положить $y = x$.

2. Для i от 0 до $k-1$

2.1. Вычислить $u = y_i \cdot m' \bmod b$.

2.2. Положить $y = y + u \cdot m \cdot b^i$.

3. Положить $y = y / b^k$.

4. Если $y \geq m$, уменьшить y на m.

5. Ответ: y.

Обоснование алгоритма:

1. После шага 2 имеем $y = x \pmod m$, поскольку к первоначальному значению $y = x$ всё время прибавляется кратное модулю m.

2. Докажем, что в шаге 3 деление выполняется точно, т.е. до деления $y = 0 \pmod{b^k}$. Рассмотрим последовательно:

при $i = 0$ в шаге 2 получим

$$y_0 = (y_0 + y_0 \cdot m' \cdot m) \bmod b = (y_0 - y_0 \cdot m^{-1} \cdot m) \bmod b = 0;$$

при $i = 1$ аналогично:

$$y_1 = (y_1 + y_1 \cdot m' \cdot m) \bmod b = (y_1 - y_1 \cdot m^{-1} \cdot m) \bmod b = 0;$$

и т.д.; за k шагов получим $y_0 = y_1 = \ldots = y_{k-1} = 0$, что означает $y = 0 \pmod{b^k}$.

Таким образом, после шага 3 имеем $y = x / b^k = x \cdot b^{-k} \pmod m$.

3. Докажем, что в шаге 4 требуется не более одного вычитания для обеспечения условия $y < m$, т.е. что к началу этого шага выполняется неравенство $y < 2m$. В самом деле, по результатам вычислений в шаге 2 имеем

$$y \leq x + m \cdot (b-1) \cdot \sum_{i=0}^{k-1} b^i = x + m \cdot (b^k - 1) < 2 \cdot m \cdot b^k;$$

здесь последнее неравенство выполнено в силу ограничения $x < m \cdot b^k$. Тогда после деления в шаге 3 получим $y < 2m$.

Оценка сложности алгоритма

Подсчитаем количество операций умножения цифр (операциями сложения/вычитания, как более быстрыми, можно пренебречь).

В шаге 2.1: 1 умножение.

В шаге 2.2: k умножений при вычислении $u \cdot m$; умножение на b^i выполняется приписыванием i нулей справа.

Итого в шаге 2 необходимо выполнить $k \cdot (k+1)$ умножений цифр, что и составляет оценку сложности алгоритма, так как в шаге 3 деление выполняется отбрасыванием k младших нулей.

Пример. Пусть $b = 10$, $k = 2$, $m = 17$, $x = 1234$.

По определению:

$\mathrm{MR}(1234, 10^2, 17) = 1234 \cdot 100^{-1} \bmod 17 = (10 \cdot 8) \bmod 17 = 12$.

По алгоритму: $m' = -7^{-1} \bmod 10 = 7$.

1. $y = 1234$.

2. $i = 0$: $u = (4 \cdot 7) \bmod 10 = 8$; $y = 1234 + 8 \cdot 17 = 1370$.
$i = 1$: $u = (7 \cdot 7) \bmod 10 = 9$; $y = 1370 + 9 \cdot 17 \cdot 10 = 2900$.
3. $y = 2900 / 100 = 29$.
4. $y = 29 - 17 = 12$.
5. Ответ: 12.

Теперь, когда мы разобрались с преобразованием Монтгомери, не составит труда освоить ещё одно понятие – **произведение Монтгомери**.

Определение. Произведением Монтгомери величин x, y, R и m называется выражение $\mathrm{MP}(x, y, R, m) = \mathrm{MR}(xy, R, m) = x \cdot y \cdot R^{-1} \bmod m$.

Алгоритм произведения Монтгомери

Вход: $m = (m_{k-1} \ldots m_0)_b$ с условием $(m, b) = 1$;
$m' = -m^{-1} \bmod b$;
x, y, где $x < m$, $y < m$.

Выход: $z = \mathrm{MP}(x, y, b^k, m) = x \cdot y \cdot b^{-k} \bmod m$.

1. Положить $z = 0$.
2. Для i от 0 до $k - 1$
2.1. Вычислить $u = (z_0 + x_i \cdot y_0) \cdot m' \bmod b$.
2.2. Положить $z = (z + x_i \cdot y + u \cdot m) / b$.
3. Если $z \geq m$, уменьшить z на m.
4. Ответ: z.

Заметим, что шаг деления на b^k в этом алгоритме отсутствует; вместо этого на каждом из k повторений шага 2 выполняется деление результата на b. По желанию читатель может самостоятельно модифицировать подобным образом алгоритм преобразования Монтгомери, что приведет к экономии памяти, необходимой для хранения значения y в процессе промежуточных вычислений.

Оценка сложности алгоритма

В шаге 2.1: 2 умножения цифр.

В шаге 2.2: k умножений при вычислении $x_i \cdot y$ и k умножений при вычислении $u \cdot m$.

Итого в алгоритме необходимо выполнить $2k \cdot (k + 1)$ умножений цифр.

Пример. Пусть $b = 10$, $k = 2$, $m = 17$, $x = 12$, $y = 14$.

По определению:

$\mathrm{MP}(12, 14, 10^2, 17) = 12 \cdot 14 \cdot 100^{-1} \bmod 17 = 12 \cdot 14 \cdot 8 \bmod 17 = 1$.

По алгоритму: $m' = -7^{-1} \bmod 10 = 7$.

1. $z = 0$.
2. $i = 0$: $u = (0 + 2 \cdot 4) \cdot 7 \bmod 10 = 6$; $z = (0 + 2 \cdot 14 + 6 \cdot 17) / 10 = 13$.
$i = 1$: $u = (3 + 1 \cdot 4) \cdot 7 \bmod 10 = 9$; $z = (13 + 1 \cdot 14 + 9 \cdot 17) / 10 = 18$.
3. $y = 18 - 17 = 1$.

4. Ответ: 1.

«Квадрат» Монтгомери

Рассмотрим два способа вычисления произведения Монтгомери числа x самого на себя.

1. Вычислить $MP(x, x, R, m)$ по последнему алгоритму; сложность $2k(k + 1)$ умножений.

2. Вычислить $q = x^2$ с помощью алгоритма быстрого возведения в квадрат (сложность $k(k + 1) / 2$), затем вычислить преобразование Монтгомери $MR(q, R, m)$ со сложностью $k(k + 1)$; итого общая сложность процедуры составит $3k(k + 1) / 2$, что на 25% меньше, чем в способе 1.

И, наконец, алгоритм, ради которого и вводились преобразование и произведение Монтгомери.

Алгоритм возведения в степень по модулю методом Монтгомери (аналог дихотомического варианта «слева направо»)

Вход: $m = (m_{k-1} \dots m_0)_b$ с условием $(m, b) = 1$;
$m' = -m^{-1} \bmod b$;
$y = (y_{n-1} \dots y_0)_2$, причем $y_{n-1} = 1$;
x, для которого выполнено $x < m$;
$R2 = b^{2k} \bmod m$.

Выход: $z = x^y \bmod m$.

1. Вычислить $x' = MP(x, R2, b^k, m)$ и положить $z = x'$.
2. Для i от $n - 2$ до 0
2.1. Вычислить $q = z^2$, затем $z = MR(q, b^k, m)$.
2.2. Если $y_i = 1$, положить $z = MP(z, x', b^k, m)$.
3. Положить $z = MR(z, b^k, m)$.
4. Ответ: z.

Прояснить то, что осталось в этом алгоритме непонятым, поможет следующий пример.

Пример. Проиллюстрируем вычисление значения $x^9 \bmod m$ для произвольных x и m. $9 = (1001)_2$; все вычисления в примере ведутся по модулю m, но для краткости выражение «mod m» будем опускать.

1. $x' = MP(x, R2, b^k, m) = x \cdot b^{2k} \cdot b^{-k} = x \cdot b^k$, $z = x' = x \cdot b^k$.
2. $i = 2$: $q = z^2 = x^2 \cdot b^{2k}$, $z = MR(q, b^k, m) = x^2 \cdot b^{2k} \cdot b^{-k} = x^2 \cdot b^k$.
$i = 1$: $q = z^2 = x^4 \cdot b^{2k}$, $z = MR(q, b^k, m) = x^4 \cdot b^{2k} \cdot b^{-k} = x^4 \cdot b^k$.
$i = 0$: $q = z^2 = x^8 \cdot b^{2k}$, $z = MR(q, b^k, m) = x^8 \cdot b^{2k} \cdot b^{-k} = x^8 \cdot b^k$,
$z = MP(z, x', b^k, m) = x^8 \cdot b^k \cdot x \cdot b^k \cdot b^{-k} = x^9 \cdot b^k$.
3. $z = MR(z, b^k, m) = x^9 \cdot b^k \cdot b^{-k} = x^9$.

И теперь стала понятной не «опасность», а, напротив, полезность домножения на вроде бы ненужный множитель b^{-k} в преобразовании и

произведении Монтгомери, благодаря которому на протяжении шага 2 множитель b^k остаётся неизменным, в то время как степень x увеличивается. В заключение этого длинного раздела заметим, что возведение в степень методом Монтгомери используется, например, в протоколе SSH и описывается в литературе много чаще, нежели метод Барретта. В то же время по опыту нескольких студенческих реализаций можно сказать, что методы Барретта и Монтгомери сравнимы между собой по эффективности.

1.6. Вычисление наибольшего общего делителя: бинарный алгоритм

Задача ставится так: для данных положительных целых чисел a и b найти их наибольший общий делитель (a, b). Знаменитый алгоритм Евклида (III в. до н. э.) решения этой задачи считается «дедушкой» всех алгоритмов и известен всем с первого курса. Вспомним два варианта этого алгоритма.

Вариант 1

1. Пока a не делится на b, выполнять:
1.1. Вычислить $r = a \bmod b$.
1.2. Положить $a = b$, $b = r$.
2. Ответ: b.

Вариант 2

1. Пока a и b не равны, выполнять:
1.1. Если $a < b$, поменять a и b местами.
1.2. Заменить число a разностью $a - b$.
2. Ответ: a.

В первом варианте используется дорогостоящая операция деления; во втором нет деления, но количество итераций, как правило, существенно больше, чем в варианте 1. Истина (в нашем случае – более эффективный алгоритм), как обычно, лежит посередине. Учтём, что деление числа в двоичной системе счисления на степень двойки выполняется быстро (с помощью сдвига вправо), а также примем во внимание следующее соображение. Пусть $a = 2^i \cdot a_1$, $b = 2^j \cdot b_1$, где a_1, b_1 – нечётные. Тогда $(a, b) = 2^{\min(i,j)} \cdot (a_1, b_1)$. Теперь вариант 2 алгоритма Евклида с целью ускорения сходимости можно модифицировать следующим образом.

Бинарный алгоритм нахождения НОД

Вход: $a > 0, b > 0$.

Выход: (a, b).

1. Представить a и b в виде $a = 2^i \cdot a_1$, $b = 2^j \cdot b_1$, где a_1, b_1 – нечётные.
2. Положить $a = a_1$, $b = b_1$, и найти $k = \min(i, j)$.

3. Пока a и b не равны, выполнять:

3.1. Если $a < b$, поменять a и b местами.

3.2. Вычислить $c = a - b$ и представить в виде $c = 2^s \cdot c_1$, где c_1 – нечётное.

3.3. Положить $a = c_1$.

4. Ответ: $2^k \cdot a$.

Пример. Пусть $a = 693$, $b = 609$.

Вариант 1

a	b	r
693	609	84
609	84	21
84	21	0

Ответ: 21.

Вариант 2

a	b
693	609
84	
609	84
525	
441	
357	
273	
189	
105	
21	
84	21
63	
42	
21	

Ответ: 21.

Бинарный алгоритм

a	b	c
693	609	$84 = 2^2 \cdot 21$
21		
609	21	$588 = 2^2 \cdot 147$
147		$126 = 2 \cdot 63$
63		$42 = 2 \cdot 21$
21		

Ответ: 21.

1.7. Извлечение квадратного корня

Задача: для заданного положительного целого a найти целую часть квадратного корня из a, т.е. найти значение $x = [\sqrt{a}]$, удовлетворяющее условию $x^2 \le a < (x + 1)^2$. Для приближённого вычисления вещественных квадратных корней известна итерационная формула Ньютона, которая выводится следующим образом. Пусть a_i – очередное приближение к $\sqrt{a}$. Представим $\sqrt{a}$ в виде $\sqrt{a} = a_i + \varepsilon$.

Тогда $$\varepsilon^2 = (\sqrt{a} - a_i)^2 = a - 2 \cdot \sqrt{a} \cdot a_i + a_i^2 \approx 0;$$

последнее приближённое равенство здесь имеет место в силу малости ε. Отсюда получаем формулу для вычисления следующего приближения к значению $\sqrt{a}$:

$$\sqrt{a} \approx a_{i+1} = (a/a_i + a_i)/2.$$

Процесс заканчивается, когда будет достигнута требуемая точность δ, т.е. когда на очередном шаге получим $|a_{i+1} - a_i| < \delta$. Оказывается, аналогичная процедура применима и для вычисления целой части квадратного корня, что обосновывается следующей теоремой.

Теорема об извлечении квадратного корня

Дано целое $a > 0$. Пусть a_0 – целое и $a_0^2 > a$. Построим последовательность Ньютона $\{a_i: i = 0, 1, \ldots\}$ по правилу

$$a_{i+1} = [([a/a_i] + a_i)/2], \; i = 0, 1, \ldots$$

Тогда:

1) найдётся p такое, что $a_p \le a_{p+1}$, и

2) для наименьшего p со свойством 1 выполняется $a_p = [\sqrt{a}\,]$.

Доказательство.

Рассмотрим вещественную функцию $F(x) = (a/x + x)/2$. Отметим, что $F(x)$ строго возрастает на интервале $(\sqrt{a}, \infty)$, $F(\sqrt{a}) = \sqrt{a}$, $x > F(x)$ при $x > \sqrt{a}$.

Докажем, что элементы последовательности Ньютона подчиняются правилу $a_{i+1} = [F(a_i)]$. Поскольку для любого целого y выполняется неравенство $[y/2] \ge y/2 - ½$ и для любого вещественного y имеет место $[y] > y - 1$, можем записать:

$$a_{i+1} = [([a/a_i] + a_i)/2] \ge ([a/a_i] + a_i)/2 - ½ >$$
$$> (a/a_i + a_i - 1)/2 - ½ = (a/a_i + a_i)/2 - 1 = F(a_i) - 1.$$

Итак, $F(a_i) - 1 < a_{i+1} \le F(a_i)$, откуда следует доказываемое равенство $a_{i+1} = [F(a_i)]$. Тогда при условии $a_i > \sqrt{a}$ получим

$$a_i > F(a_i) \ge [F(a_i)] = a_{i+1},$$

т.е. на интервале $(\sqrt{a}, \infty)$ последовательность $\{a_i\}$ строго убывает. Но последовательность целых положительных чисел не может убывать бесконечно. Значит, найдётся p такое, что $a_p \le \sqrt{a}$. Рассмотрим разность $a_{p+1} - a_p$:

$$a_{p+1} - a_p = [F(a_p)] - a_p > F(a_p) - 1 - a_p = (a/a_p - a_p)/2 - 1 \ge -1;$$

здесь последнее неравенство имеет место в силу $a/a_p \ge \sqrt{a}$ при условии $a_p \le \sqrt{a}$, откуда $a/a_p \ge a_p$. Таким образом, получили, что $a_{p+1} - a_p \ge 0$. Утверждение 1 теоремы доказано.

Для доказательства утверждения 2 убедимся, что из условия $a_{p-1} > \sqrt{a}$ следует неравенство $a_p > \sqrt{a} - 1$, что вкупе с условием $a_p \le \sqrt{a}$ и означает равенство $a_p = [\sqrt{a}\,]$. В самом деле,

$$a_p = [F(a_{p-1})] > F(a_{p-1}) - 1 > F(\sqrt{a}) - 1 = \sqrt{a} - 1;$$

последнее неравенство в этой цепочке выполняется ввиду возрастания функции $F(x)$ на интервале $(\sqrt{a}, \infty)$. Теорема доказана.

Теорема обосновывает следующий очевидный алгоритм нахождения целой части квадратного корня из заданного числа.

Алгоритм извлечения квадратного корня

Вход: a.

Выход: $[\sqrt{a}]$.

1. Выбрать каким-либо способом начальное приближение x, удовлетворяющее условию $x^2 > a$.
2. Положить $x_0 = x$, $x = [([a/x] + x)/2]$.
3. Если $x \geq x_0$, то выход, ответ: x_0; иначе переход к шагу 2.

Пример 1. Пусть $a = 10$.

1. В качестве начального приближения возьмём $x = a = 10$.
2. x_0 ——— x

10 [([10 /10] + 10)/2] = 5;
5 [([10 /5] + 5)/2] = 3;
3 [([10 /3] + 3)/2] = 3.

3. Условие $x \geq x_0$ выполнено. Ответ: 3.

Пример 2. Пусть $a = 99$.

1. $x = a = 99$.
2. x_0 ——— x

99 [([99 /99] + 99)/2] = 50;
50 [([99 /50] + 50)/2] = 25;
25 [([99 /25] + 25)/2] = 14;
14 [([99 /14] + 14)/2] = 10;
10 [([99 /10] + 10)/2] = 9;
9 [([99 /9] + 9)/2] = 10.

3. Условие $x \geq x_0$ выполнено. Ответ: 9.

Утверждение о периоде последовательности Ньютона

Пусть $a_0^2 > a$, $\{a_i: i = 0, 1, \ldots\}$ – последовательность Ньютона и p – наименьший индекс, для которого выполнено условие $a_p \leq a_{p+1}$. Тогда $a_{p+1} = a_p + 1$, если и только если число $(a + 1)$ является полным квадратом; в этом случае последовательность Ньютона периодична с периодом 2. В противном случае $a_{p+1} = a_p$, и последовательность стационарна.

Доказательство. Поскольку, по доказанной теореме, $a_p = [\sqrt{a}]$, то имеет место неравенство $a < (a_p + 1)^2$. Представим a в виде $a = a_p^2 + r$, получим $a_p^2 + r < (a_p + 1)^2 = a_p^2 + 2a_p + 1$, откуда $r \leq 2a_p$. Запишем далее

$$a_{p+1} = [F(a_p)] = [([(a_p^2 + r)/a_p] + a_p)/2] = a_p + [r/(2a_p)].$$

Теперь очевидно, что $a_{p+1} = a_p + 1$, если и только если $r = 2a_p$, и $a_{p+1} = a_p$ иначе. В случае $r = 2a_p$ имеем, во-первых:

$$a + 1 = a_p^2 + 2a_p + 1 = (a_p + 1)^2;$$

во-вторых:

$$a_{p+2} = [([((a_p + 1)^2 - 1)/(a_p + 1)] + a_p + 1)/2] = [a_p + 1 - 1/(2(a_p + 1))] = a_p.$$

Утверждение доказано.

В заключение приведём итерационную формулу для извлечения корня произвольной степени n:

$$a_{i+1} \approx \frac{1}{n} \cdot \left(\frac{a}{a_i^{n-1}} + (n-1) \cdot a_i \right).$$

1.8. Быстрое умножение: метод Карацубы

Алгоритм умножения двух n-разрядных чисел «в столбик» состоит в умножении каждой цифры на каждую и имеет, таким образом, сложность порядка n^2 операций. Долгое время считалось, что более быстрого алгоритма умножения не существует, пока в 1962 г. наш соотечественник молодой математик А.А. Карацуба не предложил изящный и очень простой метод умножения чисел с меньшей, нежели квадратичная, сложностью. Идея Карацубы состоит в следующем.

Будем рассматривать два $2n$-разрядных числа

$$u = (u_{2n-1}\, u_{2n-2} \ldots u_0)_b$$

и

$$v = (v_{2n-1}\, v_{2n-2} \ldots v_0)_b.$$

Представим их в виде $u = U_1 \cdot b^n + U_0$, $v = V_1 \cdot b^n + V_0$, где U_1, U_0, V_1, V_0 – n-разрядные числа (старшие и младшие половины чисел u и v соответственно). Можем записать:

$$u \cdot v = (U_1 \cdot b^n + U_0) \cdot (V_1 \cdot b^n + V_0) = U_1 \cdot V_1 \cdot b^{2n} + (U_1 \cdot V_0 + U_0 \cdot V_1) \cdot b^n + U_0 \cdot V_0,$$

и всё вычисление требует четырёх умножений n-разрядных чисел (умножения на b^{2n} и b^n не считаются, так как реализуются с помощью сдвигов). Пока что мы ничего не выиграли по сравнению с умножением в столбик. Заметим следующее:

$$(U_1 + U_0) \cdot (V_1 + V_0) = U_1 \cdot V_1 + (U_1 \cdot V_0 + U_0 \cdot V_1) + U_0 \cdot V_0.$$

С учётом этого произведение $u \cdot v$ можно подсчитать следующим образом. Вычислим $A = U_1 \cdot V_1$, $B = U_0 \cdot V_0$, $C = (U_1 + U_0) \cdot (V_1 + V_0)$ и запишем $u \cdot v = A \cdot b^{2n} + (C - A - B) \cdot b^n + B$.

Это уже кое-что! А именно: мы вычислили произведение двух $2n$-разрядных чисел с помощью всего трёх умножений n-разрядных чисел! Это достижение А.А. Карацубы явилось революционным прорывом в

проблеме умножения целых чисел, а обобщение его идеи привело к появлению целого ряда других, ещё более быстрых алгоритмов умножения.

Оценим сложность алгоритма Карацубы. Пусть $T(n)$ – время умножения двух n-разрядных чисел. Мы получили, что $T(2n) \approx 3 \cdot T(n)$; временем выполнения операций сложения и сдвига можно пренебречь. Тогда $T(2^k) = O(3^k)$, откуда

$$T(n) = O(3^{\log_2 n}) = O(n^{\log_2 3}) = O(n^{1.58}).$$

Пример. Пусть $b = 10$, $n = 1$, $u = 25$, $v = 14$.

$A = 2 \cdot 1 = 2$, $B = 5 \cdot 4 = 20$, $C = (2+5) \cdot (1+4) = 35$,

$u \cdot v = 2 \cdot 100 + (35 - 20 - 2) \cdot 10 + 20 = 350$.

Замечания по реализации

Оценка сложности алгоритма получена нами в предположении, что n-разрядные половины исходных чисел также умножаются методом Карацубы. Это значит, что рассчитывать на выигрыш в эффективности можно в случае рекурсивной реализации алгоритма, однако при последовательном делении чисел на две половины наступает момент, когда накладные расходы на исполнение этой рекурсивной схемы начинают превышать выигрыш от её применения. Экспериментально установлено, что минимальный размер чисел, которые ещё имеет смысл умножать методом Карацубы, а не просто «в столбик», составляет несколько десятков цифр.

Задача. Перемножить два комплексных числа, используя три вещественных умножения.

1.9. Быстрое умножение: метод Тоома–Кука

А.Л. Тоом – наш соотечественник, его работа по умножению чисел опубликована в 1963 г. в Докладах Академии наук, С. Кук – учёный Гарвардского университета, который позднее (1966 г.) предложил усовершенствование метода Тоома.

Идея состоит в следующем. Будем рассматривать u и v как $(r+1) \cdot n$-разрядные числа и разбивать их не на две, как в методе Карацубы, а на $(r+1)$ частей по n разрядов:

$$u = U_r \cdot b^{rn} + U_{r-1} \cdot b^{(r-1)n} + \ldots + U_1 \cdot b^n + U_0,$$
$$v = V_r \cdot b^{rn} + V_{r-1} \cdot b^{(r-1)n} + \ldots + V_1 \cdot b^n + V_0.$$

Рассмотрим многочлены

$$U(x) = U_r \cdot x^r + U_{r-1} \cdot x^{r-1} + \ldots + U_1 \cdot x + U_0$$

и

$$V(x) = V_r \cdot x^r + V_{r-1} \cdot x^{r-1} + \ldots + V_1 \cdot x + V_0,$$

а также их произведение

$$W(x) = U(x) \cdot V(x) = W_{2r} \cdot x^{2r} + \ldots + W_1 \cdot x + W_0.$$

Понятно, что $u = U(b^n)$, $v = V(b^n)$, поэтому $u \cdot v = W(b^n)$. Значит, для вычисления произведения $u \cdot v$ достаточно найти коэффициенты многочлена $W(x)$. Известно, что многочлен степени r однозначно определяется своими $(r + 1)$ коэффициентами или значениями в $(r + 1)$ точках. Если мы вычислим значения многочленов $U(x)$ и $V(x)$ в $(2r + 1)$ точках и почленно их перемножим, то получим значения многочлена $W(x)$ в тех же точках. Осталось выполнить переход от задания многочлена значениями в точках к заданию его своими коэффициентами – и задача решена. Разные методы умножения чисел, основанные на этой общей идее, отличаются друг от друга способами выбора точек и способами переходов от одного представления многочленов к другому.

В методе Тоома–Кука в качестве точек выбираются значения 0, 1, ..., $2r$. Значения многочленов $U(x)$ и $V(x)$ вычисляются по схеме Горнера:

$$a(x) = a_r \cdot x^r + a_{r-1} \cdot x^{r-1} + \ldots + a_1 \cdot x + a_0 =$$
$$= (\ldots(a_r \cdot x + a_{r-1}) \cdot x + a_{r-2}) \cdot x + \ldots + a_1) \cdot x + a_0.$$

Будем сразу демонстрировать работу метода Тоома–Кука на примере.

Пример 1. Пусть $b = 10$, $r = 2$, $u = 123$, $v = 112$.

$U(x) = x^2 + 2x + 3$, $V(x) = x^2 + x + 2$. Вычислим значения этих многочленов в точках 0, 1, 2, 3, 4 и почленно перемножим их; результаты приводятся в таблице:

x	0	1	2	3	4
$U(x)$	3	6	11	18	27
$V(x)$	2	4	8	14	22
$W(x)$	6	24	88	252	594

Следующая наша задача – по значениям многочлена $W(x)$ в точках найти его коэффициенты. Решим эту задачу в два этапа.

1-й этап.

Перепишем $W(x)$ в виде интерполяционной формулы Ньютона:

$$W(x) = c_4 \cdot x \cdot (x - 1) \cdot (x - 2) \cdot (x - 3) +$$
$$c_3 \cdot x \cdot (x - 1) \cdot (x - 2) +$$
$$c_2 \cdot x \cdot (x - 1) +$$
$$c_1 \cdot x +$$
$$c_0.$$

Задача этапа 1 – найти коэффициенты $c_0, \ldots, c_4$. Для этого запишем $W(x + 1)$ и найдём разность $W(x + 1) - W(x)$.

$$W(x + 1) = c_4 \cdot (x + 1) \cdot x \cdot (x - 1) \cdot (x - 2) +$$
$$c_3 \cdot (x + 1) \cdot x \cdot (x - 1) +$$
$$c_2 \cdot (x + 1) \cdot x +$$
$$c_1 \cdot (x + 1) +$$
$$c_0.$$

$$W(x+1) - W(x) = c_4 \cdot x \cdot (x-1) \cdot (x-2) \cdot 4 + c_3 \cdot x \cdot (x-1) \cdot 3 + c_2 \cdot x \cdot 2 + c_1.$$

Обозначим: $\Delta_k = W(k) - W(k-1)$,

$$\Delta_k^{(2)} = \Delta_k - \Delta_{k-1},$$

...

$$\Delta_k^{(i)} = \Delta_k^{(i-1)} - \Delta_{k-1}^{(i-1)}.$$

Заметим следующее:

$c_0 = W(0)$;

$\Delta_1 = W(1) - W(0) = c_1$;

$\Delta_2 = W(2) - W(1) = c_2 \cdot 2 + c_1$, откуда $c_2 = \dfrac{\Delta_2 - \Delta_1}{2} = \dfrac{\Delta_2^{(2)}}{2}$;

$\Delta_3 = W(3) - W(2) = c_3 \cdot 2 \cdot 3 + c_2 \cdot 2 \cdot 2 + c_1 = c_3 \cdot 3! + (c_2 \cdot 2 + c_1) + c_2 \cdot 2 =$

$= c_3 \cdot 3! + \Delta_2 + \Delta_2^{(2)}$, откуда $c_3 = \dfrac{\Delta_3 - \Delta_2 - \Delta_2^{(2)}}{3!} = \dfrac{\Delta_3^{(2)} - \Delta_2^{(2)}}{3!} = \dfrac{\Delta_3^{(3)}}{3!}$.

И так далее. Въедливый читатель может самостоятельно доказать, что $c_k = \dfrac{\Delta_k^{(k)}}{k!}$. Отсюда вытекает простой метод нахождения коэффициентов c_i. Запишем в первый столбец $(2r+1)$ значений $W(x)$. В следующий столбец поместим $(2r)$ разностей соседних элементов первого столбца. В третий столбец – $(2r-1)$ разностей соседних элементов второго столбца, делённых на 2. Продолжая далее, получим треугольную таблицу, по верхнему правому ребру которой будут записаны интересующие нас значения $c_k = \dfrac{\Delta_k^{(k)}}{k!}$. Следующая таблица иллюстрирует эту процедуру.

$W(0) = c_0$				
	$\Delta_1 = c_1$			
$W(1)$		$\dfrac{\Delta_2^{(2)}}{2} = c_2$		
	Δ_2	...	...	$\dfrac{\Delta_{2r}^{(2r)}}{(2r)!} = c_{2r}$
...	...	$\dfrac{\Delta_{2r}^{(2)}}{2}$		
	Δ_{2r}			
$W(2r)$				

То же самое для нашего примера:

$6 = c_0$

$18 = c_1$

24 $\quad 23 = c_2$

64 $\quad 9 = c_3$

88 $\quad 50$ $\quad 1 = c_4$

164 $\quad 13$

252 $\quad 89$

342

594

2-й этап

Результатом 1-го этапа явилось представление многочлена $W(x)$ в виде

$$W(x) = x \cdot (x-1) \cdot (x-2) \cdot (x-3) + 9 \cdot x \cdot (x-1) \cdot (x-2) + 23 \cdot x \cdot (x-1) + 18 \cdot x + 6.$$

Следующая задача – нахождение коэффициентов многочлена $W(x)$. Перепишем его в виде, аналогичном представлению по схеме Горнера:

$$W(x) = (((1 \cdot (x-3) + 9) \cdot (x-2) + 23) \cdot (x-1) + 18) \cdot x + 6.$$

И вычислим по этой же схеме, последовательно раскрывая скобки, начиная с самых внутренних:

1	9			
	$-3 \cdot 1$			
1	6	23		
	$-2 \cdot 1$	$-2 \cdot 6$		
1	4	11	18	
	-1	-4	-11	
1	3	7	7	6

Получили $W(x) = x^4 + 3x^3 + 7x^2 + 7x + 6$, и $u \cdot v = W(10) = 13776$.

Пример 2. Пусть $b = 10$, $r = 2$, $u = 111$, $v = 555$.

$U(x) = x^2 + x + 1$, $V(x) = 5x^2 + 5x + 5$.

x	0	1	2	3	4
$U(x)$	1	3	7	13	21
$V(x)$	5	15	35	65	105
$W(x)$	5	45	245	845	2205

1-й этап

$5 = c_0$

$40 = c_1$

45 $\quad 80 = c_2$

200 $\quad 40 = c_3$

245 $\quad 200$ $\quad 5 = c_4$

600 $\quad 60$

845 $\quad 380$

1360

2205

2-й этап

5	40			
	−35			
5	25	80		
	−25	−225		
5	15	30	40	
	−5	−15	−30	
5	10	15	10	5

В отличие от предыдущего примера, здесь коэффициенты многочлена $И(x)$ получились больше, чем из одной цифры. Поэтому ответ вычисляется не просто приписыванием цифр друг к другу, а сложением с переносами:

	5				
+	1	0			
+		1	5		
+			1	0	
+					5
	6	1	6	0	5

Ответ: 61605.

1.10. Быстрое умножение: метод Шёнхаге–Штрассена

Для изложения этого (самого быстрого на данный момент) метода умножения нам предстоит ввести ряд новых понятий и познакомиться с красивой математикой.

1.10.1. Примитивные корни из 1

Изложим ещё раз общую идею умножения чисел с помощью умножения многочленов. Пусть A – преобразование, выполняющее переход от коэффициентов многочлена к его значениям в точках (интерполяция), A^{-1} – обратное преобразование, $\cdot$ – почленное произведение компонент векторов, $u(x)$ и $v(x)$ – многочлены, соответствующие числам u и v. Тогда многочлен $w(x)$, соответствующий произведению $u \cdot v$, получается следующим преобразованием: $w(x) = A^{-1}(A(u(x)) \cdot A(v(x)))$. К преобразованию A предъявляются требования обратимости и эффективной выполнимости как A, так и A^{-1}.

Пусть многочлен $u(x)$ имеет степень n и задан вектор-столбцом своих коэффициентов $U = [u_0 \ \dots \ u_{n-1}]^T$, и пусть заданы точки $x_0, \dots, x_{n-1}$. Построим матрицу Вандермонда V с элементами x_i^j, $i, j = 0, \dots, n-1$:

$$V = \begin{bmatrix} x_0^0 & \dots & x_0^{n-1} \\ \dots & & \dots \\ x_{n-1}^0 & \dots & x_{n-1}^{n-1} \end{bmatrix}$$

Рассмотрим преобразование A вектора U, состоящее в умножении матрицы V на вектор U:

$$A(U) = V \cdot U = \begin{Vmatrix} u(x_0) \\ \dots \\ u(x_{n-1}) \end{Vmatrix}$$ – это и есть преобразование интерполяции.

Соответственно, обратное преобразование есть $A^{-1}(U) = V^{-1} \cdot U$, и преобразование A обратимо, если и только если обратима матрица Вандермонда, которая, в свою очередь, обратима, если и только если обратим её определитель.

Утверждение об определителе матрицы Вандермонда

Пусть V – матрица Вандермонда порядка n. Тогда $\det V = \prod_{0 \le i < j < n} (x_j - x_i)$.

Доказательство индукцией по n.

База индукции. При $n = 2$ имеем $\det V = \begin{vmatrix} x_0^0 & x_0^1 \\ x_1^0 & x_1^1 \end{vmatrix} = \begin{vmatrix} 1 & x_0 \\ 1 & x_1 \end{vmatrix} = x_1 - x_0$.

Предположение индукции. Обозначим $\Delta(x_0, \dots, x_{m-1})$ определитель матрицы Вандермонда порядка m на точках $x_0, \dots, x_{m-1}$; предположим, что $\Delta(x_0, \dots, x_{m-1}) = \prod_{0 \le i < j < m} (x_j - x_i)$. Рассмотрим определитель порядка $m + 1$:

$$\Delta(x_0, \dots, x_m) = \begin{vmatrix} 1 & x_0 & \dots & x_0^m \\ 1 & x_1 & \dots & x_1^m \\ & \dots & & \\ 1 & x_m & \dots & x_m^m \end{vmatrix}.$$

Вычтем из каждого i-го столбца $(i-1)$-й, умноженный на x_0 ($i = 1, \dots, m$), разложим определитель по нулевой строке и вынесем за знак определителя общие множители каждой строки:

$$\Delta(x_0, \dots, x_m) = \begin{vmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & x_1 - x_0 & x_1(x_1 - x_0) & \dots & x_1^{m-1}(x_1 - x_0) \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_m - x_0 & x_m(x_m - x_0) & \dots & x_m^{m-1}(x_m - x_0) \end{vmatrix} =$$

$$= (x_1 - x_0) \cdot \dots \cdot (x_m - x_0) \cdot \begin{vmatrix} 1 & x_1 & \dots & x_1^{m-1} \\ \dots & \dots & \dots & \dots \\ 1 & x_m & \dots & x_m^{m-1} \end{vmatrix} =$$

$$= (x_1 - x_0) \cdot \ldots \cdot (x_m - x_0) \cdot \Delta(x_1, \ldots, x_m) =$$

$$= (x_1 - x_0) \cdot \ldots \cdot (x_m - x_0) \cdot \prod_{1 \le i < j \le m} (x_j - x_i) = \prod_{0 \le i < j \le m} (x_j - x_i).$$

Утверждение доказано.

Итак, условием обратимости матрицы Вандермонда, а значит, и обратимости преобразования интерполяции является обратимость всех разностей $x_j - x_i$. Если многочлен $u(x)$ задан над *полем* и точки $x_0, \ldots, x_{n-1}$ – элементы поля, то для обратимости матрицы Вандермонда необходимо и достаточно, чтобы все эти точки были различны, поскольку в поле обратимы все ненулевые элементы. Ситуация сложнее, если $u(x)$ задан над *кольцом*. Для обеспечения условия обратимости матрицы Вандермонда над кольцом и вводится, наконец, понятие примитивных корней из 1. Далее рассматриваем K – произвольное *коммутативное кольцо с* 1.

Определение 1. Элемент $w \in K$ называется *корнем n-й степени из* 1, если $w^n = 1$.

Определение 2. Корень n-й степени из 1 w в кольце K называется *примитивным*, если элементы $1 - w^i$ обратимы для всех $i = 1, \ldots, n - 1$.

Свойства примитивных корней

Пусть w – примитивный корень n-й степени из 1 в кольце K.

1. Корень w обратим.

Свойство следует из равенства $w \cdot w^{n-1} = 1$.

2. *Порядок* w в K равен n.

Свойство следует, во-первых, из равенства $w^n = 1$ и, во-вторых, из обратимости элементов $1 - w^i$ для всех $i = 1, \ldots, n - 1$, что влечёт неравенства $w^i \neq 1$ для $0 < i < n$.

3. Элементы $1, w, w^2, \ldots, w^{n-1}$ суть различные корни n-й степени из 1 в K.

Тот факт, что это всё корни, проверяется непосредственно. Их различность следует из свойства 2.

4. Если K является полем, то любой элемент порядка n есть примитивный корень n-й степени из 1.

Это верно ввиду обратимости всех ненулевых элементов поля.

5. $\sum_{j=0}^{n-1} w^{jp} = 0$ для всякого $p = 1, \ldots, n - 1$.

Следует из цепочки равенств $0 = 1 - w^{np} = (1 - w^p) \cdot \sum_{j=0}^{n-1} w^{jp}$ и из обратимости элемента $1 - w^p$.

6. w^{-1} также является примитивным корнем n-й степени из 1 в K.

Верно ввиду равенств $(w^{-1})^n = (w^n)^{-1} = 1^{-1} = 1$ и $1 - w^{-i} = 1 - w^{n-i}$.

Примеры:

1. В поле вещественных чисел 1 и -1 – корни второй степени из 1; из них -1 – примитивный.

2. В поле комплексных чисел 1, -1, i, $-i$ – корни четвёртой степени из 1; из них i и $-i$ – примитивные.

3. В поле комплексных чисел $e^{i\frac{2\pi}{n}} = \cos\frac{2\pi}{n} + i\cdot\sin\frac{2\pi}{n}$ – примитивный корень степени n из 1.

4. В поле Z_5 элементы 2 и 3 – примитивные корни степени 4 из 1.

5. В кольце Z_{15} корни второй степени из 1 суть 1, 4, 11, 14, из них 4, 11, 14 имеют порядок 2, но только 14 – примитивный корень.

Вернёмся к матрице Вандермонда и проблеме обратимости преобразования интерполяции. Пусть $w \in K$. Возьмём в качестве точек $x_0, \ldots, x_{n-1}$ степени элемента w: $x_i = w^i$. В этом случае матрица Вандермонда называется матрицей, ассоциированной с w, обозначается V_w и имеет вид $V_w = \|w^{ij}\|$, $i, j = 0, \ldots, n-1$.

Теорема о матрице Вандермонда

Пусть w – корень n-й степени из 1 (не обязательно примитивный) и $\sum_{j=0}^{n-1} w^{jp} = 0$ для всякого $p = 1, \ldots, n-1$. Тогда $V_w \cdot V_{w^{-1}} = n \cdot E$, где E – единичная матрица порядка n.

Доказательство.

Пусть $B = V_w \cdot V_{w^{-1}}$. Тогда $b_{ij} = \sum_{k=0}^{n-1} w^{ik} \cdot w^{-kj} = \sum_{k=0}^{n-1} w^{(i-j)k}$. Рассмотрим возможные случаи.

1. При $i = j$ имеем $b_{ij} = n$.

2. При $i > j$ выполняется неравенство $0 < i - j \le n - 1$, и $b_{ij} = 0$ по условию теоремы для $p = i - j$.

3. При $i < j$ верно неравенство $1 - n \le i - j < 0$, откуда $1 \le n + i - j < n$, и доказываемое $b_{ij} = 0$ следует из равенства $w^{(i-j)k} = w^{(n+i-j)k}$.

Теорема доказана.

Следствие 1. В условиях теоремы элемент n обратим, если и только если элементы $1 - w^i$ обратимы для всех $i = 1, \ldots, n-1$.

Доказательство.

По доказанному в теореме,

$$\det V_w \cdot \det V_{w^{-1}} = \det(V_w \cdot V_{w^{-1}}) = \det(n \cdot E) = n^n .$$

С другой стороны, по утверждению об определителе матрицы Вандермонда,

$$\det V_w \cdot \det V_{w^{-1}} = \prod_{0\le j<i<n}(w^i - w^j)\cdot \prod_{0\le j<i<n}(w^{-i} - w^{-j}) =$$

$$= \prod_{0\le j<i<n}(w^i - w^j)\cdot \prod_{0<n-i<n-j\le n}(w^{n-i} - w^{n-j}) =$$

$$= \prod_{0\le j<i<n}(w^i - w^j)\cdot \prod_{0<i<j<n}(w^i - w^j)\cdot \prod_{i=1}^{n-1}(w^i - w^0) = \pm\prod_{i\ne j}(w^i - w^j).$$

Следовательно, элемент n и все разности $w^i - w^j$ обратимы или не обратимы одновременно, а обратимость последних равносильна обратимости элементов $1 - w^i$ для всех $i = 1, \ldots, n-1$.

Следствие 2. Если w – примитивный корень n-й степени из 1, то n обратим и $V_w^{-1} = n^{-1}\cdot V_{w^{-1}}$.

Для доказательства достаточно обратиться к свойству 5 примитивных корней.

Проницательный читатель, наверное, уже догадался, что степени примитивного корня из 1 используются в качестве точек, в которых вычисляются значения многочленов в процессе умножения чисел. Следующая теорема помогает построить кольцо, в котором существует примитивный корень.

Теорема о примитивных корнях

Пусть $n = 2^k$, w – чётное число, $m = w^{n/2} + 1$. Тогда w – примитивный корень n-й степени из 1 в Z_m.

Доказательство.

Докажем сначала вспомогательную лемму.

Лемма. Пусть $n = 2^k$. Тогда $\sum_{i=0}^{n-1} a^i = \prod_{i=0}^{k-1}(1 + a^{2^i})$.

Доказательство индукцией по k.

База индукции. При $k = 1$ имеем тождество $1 + a = 1 + a$.

Предположение индукции. Пусть утверждение леммы верно при $n = 2^k$.

Шаг индукции. Рассмотрим $n = 2^{k+1}$.

$$\sum_{i=0}^{n-1} a^i = \sum_{i=0}^{2^k-1} a^i + a^{2^k}\cdot \sum_{i=0}^{2^k-1} a^i = (1 + a^{2^k})\cdot \prod_{i=0}^{k-1}(1 + a^{2^i}) = \prod_{i=0}^{k}(1 + a^{2^i}).$$

Лемма доказана.

Доказательство теоремы.

1. n обратим в Z_m, так как $(n, m) = 1$.

2. $w^n = (w^{n/2})^2 = (-1)^2 = 1$.

3. Рассмотрим сумму $\sum_{j=0}^{n-1} w^{jp}$ для любого $p \in \{1, \ldots, n-1\}$. По лемме верно равенство $\sum_{j=0}^{n-1} w^{jp} = \prod_{j=0}^{k-1} (1 + w^{p \cdot 2^j})$. Докажем, что в произведении существует сомножитель, равный 0. Представим p в виде $p = 2^s \cdot p'$, где p' – нечётное. Заметим, что $0 \le s < k$. Возьмём $j = k - 1 - s$; очевидно, $0 \le j < k$. Рассмотрим сомножитель $1 + w^{p \cdot 2^j}$:

$$1 + w^{p \cdot 2^j} = 1 + w^{2^{k-1} \cdot p'} = 1 + (w^{n/2})^{p'} = 1 + (-1)^{p'} = 0 .$$

Итак, получили, что $\sum_{j=0}^{n-1} w^{jp} = 0$ для любого $p \in \{1, \ldots, n-1\}$. Поскольку тем самым выполнены условия теоремы о матрице Вандермонда, то в силу обратимости n по следствию 1 из этой теоремы заключаем, что элементы $1 - w^i$ обратимы для всех $i = 1, \ldots, n-1$. Это, в свою очередь, означает, что w есть примитивный корень n-й степени из 1 в Z_m. Теорема доказана.

Интересные задачи на тему «Примитивные корни из 1» можно найти в [12], глава 5.

1.10.2. Дискретное преобразование Фурье

Определение. Пусть K – коммутативное кольцо с 1, w – примитивный корень n-й степени из 1 в K, a – вектор-столбец длины n с элементами из K. *Дискретным преобразованием Фурье* (*ДПФ*) вектора a называется вектор $F(a) = V_w \cdot a$. *Обратным дискретным преобразованием Фурье* (*ОДПФ*) вектора a называется вектор $F^{-1}(a) = V_w^{-1} \cdot a = n^{-1} \cdot V_{w^{-1}} \cdot a$.

Из определения видно, что ОДПФ – то же (с точностью до умножения на константу) преобразование, что и ДПФ, но с другим примитивным корнем.

Пример 1. Пусть $n = 4$. Матрица Вандермонда в общем виде:

$$V_w = \begin{Vmatrix} w^0 & w^0 & w^0 & w^0 \\ w^0 & w^1 & w^2 & w^3 \\ w^0 & w^2 & w^4 & w^6 \\ w^0 & w^3 & w^6 & w^9 \end{Vmatrix} = \begin{Vmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & 1 & w^2 \\ 1 & w^3 & w^2 & w \end{Vmatrix}.$$

Пусть $K = Z_5$, $n = 4$, $w = 2$. Тогда $n^{-1} = 4$, $w^{-1} = 3$.

$$V_w = \begin{Vmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \\ 1 & 3 & 4 & 2 \end{Vmatrix}, \quad V_w^{-1} = 4 \cdot \begin{Vmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 4 & 2 \\ 1 & 4 & 1 & 4 \\ 1 & 2 & 4 & 3 \end{Vmatrix}.$$

Для $a = [4\ 3\ 2\ 1]^T$ получим $F(a) = V_w \cdot a = [0\ 1\ 2\ 3]^T$.

Содержательный смысл ДПФ: если вектор a задаёт коэффициенты многочлена $a(x)$, то $F(a)$ есть вектор значений $a(x)$ в точках $1, w, \ldots, w^{n-1}$. Для нашего примера $a(x) = 4 + 3x + 2x^2 + x^3$, $a(1) = 0$, $a(w) = a(2) = 1$, $a(w^2) = a(4) = 2$, $a(w^3) = a(3) = 3$. ОДПФ – это переход от представления многочлена его значениями в точках к представлению его коэффициентами.

1.10.3. Свёртки

Определение 1. Пусть $a = [a_0 \ldots a_{n-1}]$, $b = [b_0 \ldots b_{n-1}]$. *Свёрткой* векторов a и b называется вектор c размера $2n$ (обозначается $c = a \otimes b$) с компонентами $c_i = \sum_{j+k=i} a_j \cdot b_k$, $i = 0, 1, \ldots, 2n-1$.

Пример 2. Пусть $n = 4$, $a = [a_0\, a_1\, a_2\, a_3]$, $b = [b_0\, b_1\, b_2\, b_3]$, $c = a \otimes b$. Тогда

$c_0 = a_0 \cdot b_0$,
$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0$,
$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0$,
$c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$,
$c_4 = a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1$,
$c_5 = a_2 \cdot b_3 + a_3 \cdot b_2$,
$c_6 = a_3 \cdot b_3$,
$c_7 = 0$.

Содержательный смысл понятия свёртки: если векторам a, b, $c = a \otimes b$ сопоставить многочлены $a(x)$, $b(x)$, $c(x)$ соответственно, то $c(x) = a(x) \cdot b(x)$. С учётом этого и того факта, что для задания многочлена степени $2n-1$ необходимо задать его значения в $2n$ точках, становится очевидной следующая теорема.

Теорема о свёртке

Пусть $a = [a_0 \ldots a_{n-1}]$, $b = [b_0 \ldots b_{n-1}]$.

Тогда $a \otimes b = F^{-1}(F(a') \cdot F(b'))$, где $\cdot$ – почленное произведение компонент векторов, a' и b' – векторы длины $2n$, полученные из a и b соответственно дополнением их нулями:

$$a' = [a_0 \ldots a_{n-1}\ 0 \ldots 0],\ b' = [b_0 \ldots b_{n-1}\ 0 \ldots 0].$$

Пример 3. Пусть $K = Z_5$, $n = 2$, $a(x) = 2 + x$, $b(x) = 1 + x$. Тогда $a' = [2\ 1\ 0\ 0]^T$, $b' = [1\ 1\ 0\ 0]^T$. Для вычисления ДПФ и ОДПФ воспользуемся данными из примера 1. Получим

$$F(a') = [3\ 4\ 1\ 0]^T,\ F(b') = [2\ 3\ 0\ 4]^T,$$

$$F(a') \cdot F(b') = [1\ 2\ 0\ 0]^T,$$

$$a \otimes b = F^{-1}(F(a') \cdot F(b')) = 4 \cdot [3\ 2\ 4\ 0]^T = [2\ 3\ 1\ 0]^T.$$

Проверяем соответствие содержательному смыслу: в самом деле,

$$a(x) \cdot b(x) = (2 + x) \cdot (1 + x) = 2 + 3x + x^2.$$

Технический недостаток преобразования свёртки состоит в увеличении (удваивании) числа компонент в результате по сравнению с исходными данными. Этот недостаток преодолевается в следующих двух понятиях.

Определение 2. Пусть $a = [a_0 \dots a_{n-1}]$, $b = [b_0 \dots b_{n-1}]$, $c = a \otimes b$. *Положительно обёрнутой свёрткой* векторов a и b называется вектор d размера n (обозначается $d = a\ [+]\ b$) с компонентами $d_i = c_i + c_{i+n}$, $i = 0, 1, \dots, n-1$. *Отрицательно обёрнутой свёрткой* векторов a и b называется вектор e размера n (обозначается $e = a\ [-]\ b$) с компонентами $e_i = c_i - c_{i+n}$, $i = 0, 1, \dots, n-1$.

Формулы для компонент положительно и отрицательно обёрнутых свёрток можно записать по-другому, выразив их значения через компоненты векторов a и b:

$$d_i = \sum_{j=0}^{i} a_j \cdot b_{i-j} + \sum_{j=i+1}^{n-1} a_j \cdot b_{n+i-j}, \quad e_i = \sum_{j=0}^{i} a_j \cdot b_{i-j} - \sum_{j=i+1}^{n-1} a_j \cdot b_{n+i-j}.$$

Содержательный смысл этих понятий: если векторам a, b, $d = a\ [+]\ b$, $e = a\ [-]\ b$ сопоставить многочлены $a(x)$, $b(x)$, $d(x)$, $e(x)$ соответственно, то $d(x) = a(x) \cdot b(x) \bmod (x^n - 1)$ (так как в этом случае $x^n = 1$) и $e(x) = a(x) \cdot b(x) \bmod (x^n + 1)$ (поскольку $x^n = -1$).

Продолжая пример 3, вычислим: $a\ [+]\ b = [3\ 3]$; $a\ [-]\ b = [1\ 3]$. Проверяем:

$$a(x) \cdot b(x) = 2 + 3x + x^2 = 3 + 3x \pmod{x^2 - 1};$$

$$2 + 3x + x^2 = 1 + 3x \pmod{x^2 + 1}.$$

Теорема о положительно обёрнутой свёртке

Пусть $a = [a_0 \dots a_{n-1}]$, $b = [b_0 \dots b_{n-1}]$.

Тогда $a\ [+]\ b = F^{-1}(F(a) \cdot F(b))$.

Доказательство.

Докажем следующее равносильное утверждению теоремы равенство: $F(a\ [+]\ b) = F(a) \cdot F(b)$. Пусть $d = a\ [+]\ b$, т. е.

$$d_i = \sum_{j=0}^{i} a_j \cdot b_{i-j} + \sum_{j=i+1}^{n-1} a_j \cdot b_{n+i-j};\ f = F(d);\ g = F(a) \cdot F(b).$$

Вычислим k-ю компоненту вектора f:

$$f_k = \sum_{i=0}^{n-1} w^{ik} \cdot d_i = \sum_{i=0}^{n-1} \sum_{j=0}^{i} w^{ik} \cdot a_j \cdot b_{i-j} + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} w^{ik} \cdot a_j \cdot b_{n+i-j} =$$

$$= \sum_{j=0}^{n-1} \sum_{i=j}^{n-1} w^{ik} \cdot a_j \cdot b_{i-j} + \sum_{j=0}^{n-1} \sum_{i=0}^{j-1} w^{ik} \cdot a_j \cdot b_{n+i-j}.$$

Сделаем замену $t = i - j$ в первой сумме и $t = n + i - j$ – во второй.

$$f_k = \sum_{j=0}^{n-1} \sum_{t=0}^{n-1-j} w^{(t+j)k} \cdot a_j \cdot b_t + \sum_{j=0}^{n-1} \sum_{t=n-j}^{n-1} w^{(t+j-n)k} \cdot a_j \cdot b_t =$$

$$= \sum_{j=0}^{n-1} \sum_{t=0}^{n-1} w^{(t+j)k} \cdot a_j \cdot b_t = \left(\sum_{j=0}^{n-1} w^{jk} \cdot a_j \right) \cdot \left(\sum_{t=0}^{n-1} w^{tk} \cdot b_t \right) = g_k.$$

Теорема доказана.

Теорема об отрицательно обёрнутой свёртке

Пусть w – примитивный корень n-й степени из 1 в кольце K, ψ – элемент K такой, что $\psi^n = -1$, $e = a\,[-]\,b$ – отрицательно обёрнутая свёртка векторов $a = [a_0 \ldots a_{n-1}]$ и $b = [b_0 \ldots b_{n-1}]$,

$$a' = [a_0 \ \psi \cdot a_1 \ldots \psi^{n-1} \cdot a_{n-1}]^T,$$
$$b' = [b_0 \ \psi \cdot b_1 \ldots \psi^{n-1} \cdot b_{n-1}]^T,$$
$$e' = [e_0 \ \psi \cdot e_1 \ldots \psi^{n-1} \cdot e_{n-1}]^T.$$

Тогда $e' = F^{-1}(F(a') \cdot F(b'))$.

Доказательство аналогично доказательству теоремы о положительно обёрнутой свёртке с учётом условия $\psi^n = -1$.

1.10.4. Алгоритм быстрого преобразования Фурье

К настоящему моменту мы установили, что ДПФ есть преобразование интерполяции и оно обратимо. Этого достаточно для использования ДПФ при умножении чисел в принципе, но для того чтобы умножать быстро, нужно научиться выполнять это преобразование максимально эффективно. Алгоритм быстрого преобразования Фурье (БПФ) позволяет это сделать в случае $n = 2^k$. Попробуем сначала понять идею алгоритма БПФ на примере; для этого установим следующий очевидный факт:

Задача. Пусть n – чётное, w – примитивный корень n-й степени из 1. Тогда $w^{n/2} = -1$.

Решение.

$1 - w^n = 0 = (1 - w^{n/2}) \cdot (1 + w^{n/2})$, откуда в силу обратимости элемента $(1 - w^{n/2})$ следует равенство $1 + w^{n/2} = 0$, или $w^{n/2} = -1$.

Пример 4. Пусть $n = 4$, $a = [a_0\ a_1\ a_2\ a_3]$, $b = [b_0\ b_1\ b_2\ b_3] = F(a)$. Выпишем и немного преобразуем с учётом равенств $w^2 = -1$ и $w^4 = 1$ формулы вычисления компонент вектора b:

$b_0 = a_0 + a_1 + a_2 + a_3 = (a_0 + a_2) + (a_1 + a_3)$;

$b_1 = a_0 + a_1 \cdot w + a_2 \cdot w^2 + a_3 \cdot w^3 = (a_0 - a_2) + (a_1 - a_3) \cdot w$;

$b_2 = a_0 + a_1 \cdot w^2 + a_2 \cdot w^4 + a_3 \cdot w^6 = (a_0 + a_2) + (a_1 + a_3) \cdot w^2$;

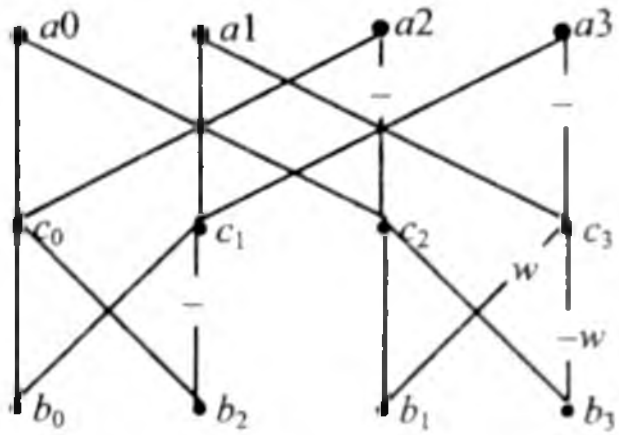
$b_3 = a_0 + a_1 \cdot w^3 + a_2 \cdot w^6 + a_3 \cdot w^9 = (a_0 - a_2) + (a_1 - a_3) \cdot w^3$.

Заметим, что в этой схеме по два раза используются значения $(a_0 + a_2)$, $(a_1 + a_3)$, $(a_0 - a_2)$ и $(a_1 - a_3)$. Вычислим предварительно их:

$c_0 = a_0 + a_2$, $c_1 = a_1 + a_3$, $c_2 = a_0 - a_2$, $c_3 = a_1 - a_3$.

Тогда $b_0 = c_0 + c_1$, $b_1 = c_2 + c_3 \cdot w$, $b_2 = c_0 - c_1$, $b_3 = c_2 - c_3 \cdot w$.

Графически процесс иллюстрируется следующей схемой:



Ключевые идеи алгоритма БПФ

1-я идея. ДПФ $F(a)$ есть по сути вычисление значений многочлена $a(x)$ в точках – степенях примитивного корня из 1. В свою очередь, вычисление значения многочлена в точке c есть нахождение остатка от деления многочлена на $x - c$:

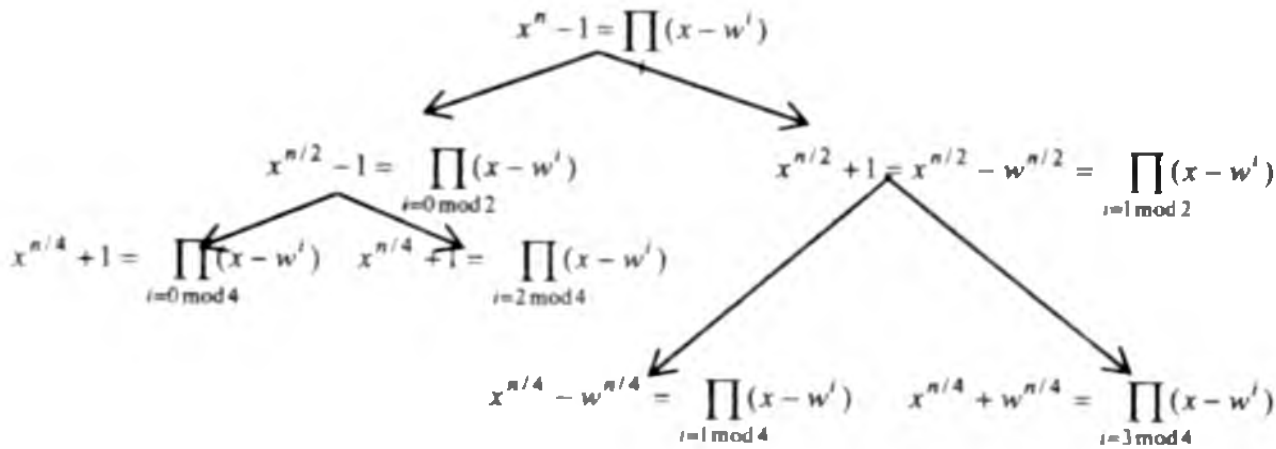
$$a(c) = a(x) \bmod (x - c).$$

Для того чтобы в этом удостовериться, поделим $a(x)$ на $(x - c)$ с остатком: $a(x) = (x - c) \cdot q(x) + r(x)$.

При $x = c$ последнее равенство принимает вид $a(c) = r(c)$, и $r(x) = \text{const}$ в силу $\deg r(x) < \deg (x - c) = 1$.

2-я идея. Применение 1-й идеи «в лоб», т.е. поочерёдное деление $a(x)$ на многочлены $x - 1$, $x - w$, ..., $x - w^{n-1}$ даст сложность порядка n^2 операций, и никакого выигрыша по сравнению со стандартным умножением матрицы на вектор мы не получим. Воспользуемся условием $n = 2^k$ и тем,

что $x^n - 1 = \prod_{i=0}^{n-1}(x - w^i)$; последнее верно ввиду того, что все степени w^i, $i = 0, \ldots, n-1$, суть различные корни многочлена $x^n - 1$ (см. свойство 3 примитивных корней). Запишем $x^n - 1 = (x^{n/2} - 1)(x^{n/2} + 1)$ и найдем остатки $r_1(x) = a(x) \bmod (x^{n/2} - 1)$ и $r_2(x) = a(x) \bmod (x^{n/2} + 1)$. Для нахождения каждого из них потребуется порядка $n/2$ операций, итого сложность шага равна n. Затем запишем в виде произведения двух сомножителей каждый из многочленов $x^{n/2} - 1$ и $x^{n/2} + 1$, и так далее, пока через $k = \log_2 n$ шагов не получим многочлены первой степени $x - 1$, $x - w$, ..., $x - w^{n-1}$. Общая сложность вычислений по такой схеме составит $O(n \log_2 n)$ операций. Разбивать на каждом шаге сомножители $x - w^i$ на две части будем так, чтобы в результате произведения получался двучлен. Для этого степени i придётся переупорядочить; для того чтобы понять, как это сделать, рассмотрим схему выполнения всей процедуры:



Таким образом, переупорядочить степени i надо так:

1) выписать все чётные i, затем нечётные, т.е. отсортировать по младшему (нулевому) биту двоичного представления i;

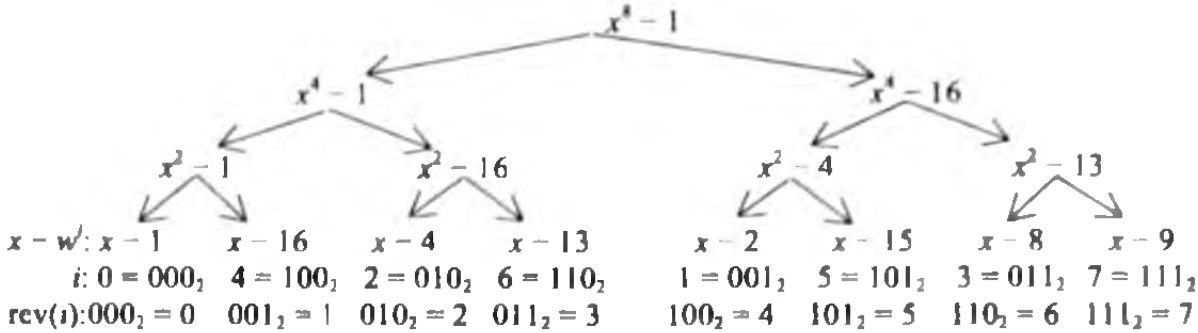
2) внутри каждой группы отсортировать по младшему биту $i/2$, т. е. по первому биту i; и так далее.

Читатель, знакомый с побитовой сортировкой, увидит здесь полную аналогию, за тем лишь исключением, что сортировка ведётся, начиная с младших битов, а не со старших. Введём операцию rev(i), которая состоит в изменении порядка битов в числе на противоположный. Значения i-х компонент вектора $F(a)$ будем получать в порядке возрастания значений rev(i).

Пример 5. Пусть $n = 8$, $w = 2$, $m = w^{n/2} + 1 = 17$.

Степени примитивного корня:

i	0	1	2	3	4	5	6	7
w^i	1	2	4	8	16 = −1	15 = −2	13 = −4	9 = −8



3-я идея. Выведем формулу нахождения остатка от деления многочлена $a(x)$ степени $2t-1$ на двучлен x^t-c:

$$a(x)=\sum_{i=0}^{2t-1}a_i\cdot x^i=(x^t-c)\cdot\sum_{i=0}^{t-1}a_{i+t}\cdot x^i+\sum_{i=0}^{t-1}(a_i+c\cdot a_{i+t})\cdot x^i .$$

Таким образом, если $r(x)=a(x) \bmod (x^t-c)$, то коэффициенты r_i находятся по формулам

$$r_i=a_i+c\cdot a_{i+t},\ i=0,\ \ldots,\ t-1.$$

Алгоритм быстрого преобразования Фурье

Вход: K – коммутативное кольцо с 1, $n=2^k$, $a=[a_0\ \ldots\ a_{n-1}]$ – вектор с элементами из K, w – примитивный корень степени n из 1 в кольце K.

Выход: $b=F(a)$.

Будем использовать вспомогательные массивы S и R длины n и обозначать их i-е элементы $S[i]$, $R[i]$ соответственно.

1. Для $i=0,\ \ldots,\ n-1$ положить $R[i]=a_i$.
2. Для $l=k-1,\ \ldots,\ 0$ выполнять:
   2.1. Положить $S[i]=R[i]$ для $i=0,\ \ldots,\ n-1$.
   2.2. Для $i=0,\ \ldots,\ n-1$:
   пусть $i=(i_{k-1}\ \ldots\ i_0)_2$; положить
   $$R[i]=S[(i_{k-1}\ \ldots\ i_{l+1}\ 0\ i_{l-1}\ \ldots\ i_0)_2]+w^{\mathrm{rev}(i/2^l)}\cdot S[(i_{k-1}\ \ldots\ i_{l+1}\ 1\ i_{l-1}\ \ldots\ i_0)_2].$$
3. Для $i=0,\ \ldots,\ n-1$ положить $b_i=R[\mathrm{rev}(i)]$.

Для вычисления ОДПФ применяется этот же алгоритм, но вместо w используется w^{-1} и в шаге 3 полагаем $b_i=n^{-1}\cdot R[\mathrm{rev}(i)]$.

Пример 6. Пусть $n=4$, $w=2$, $m=w^{n/2}+1=5$, $a=[4\ 3\ 2\ 1]^{\mathrm{T}}$.

Значение индекса i для наглядности будем всюду указывать в двоичном виде. В следующей таблице показан процесс пересчёта массива R:

i	00	01	10	11
Шаг 1	4	3	2	1
Шаг 2, $l=1$	1	4	2	2
Шаг 2, $l=0$	0	2	1	3

Пояснения к таблице:

$l=1$: $R[00]=S[00]+w^0\cdot S[10]=4+1\cdot 2=1$;
$R[01]=S[01]+w^0\cdot S[11]=3+1\cdot 1=4$;

$R[10] = S[00] + w^2 \cdot S[10] = 4 - 1 \cdot 2 = 2;$

$R[11] = S[01] + w^2 \cdot S[11] = 3 - 1 \cdot 1 = 2.$

$l = 0$: $R[00] = S[00] + w^0 \cdot S[01] = 1 + 1 \cdot 4 = 0;$

$R[01] = S[00] + w^2 \cdot S[01] = 1 - 1 \cdot 4 = 2;$

$R[10] = S[10] + w^1 \cdot S[11] = 2 + 2 \cdot 2 = 1;$

$R[11] = S[10] + w^3 \cdot S[11] = 2 - 2 \cdot 2 = 3.$

Ответ: $F(a) = [0\ 1\ 2\ 3]$, что совпадает с результатом, полученным в примере 1.

Из алгоритма БПФ и теорем о свёртках вытекает

Утверждение. ДПФ, ОДПФ, свёртку, положительно и отрицательно обёрнутые свёртки можно вычислить за $O(n \cdot \log_2 n)$ шагов.

1.10.5. Алгоритм Шёнхаге–Штрассена умножения целых чисел

Вот, наконец, мы имеем весь необходимый материал для изложения одного из самых быстрых на сегодня алгоритмов умножения. Изложим сначала основные идеи алгоритма, а затем кратко повторим всю схему его работы.

Будем рассматривать два N-разрядных двоичных числа, где $N = 2^n$:

$$u = (u_{N-1} \ldots u_0)_2,\ v = (v_{N-1} \ldots v_0)_2.$$

Будем искать их произведение по модулю $2^N + 1$. Если нужно точно вычислить $u \cdot v$, то дополним u и v слева N незначащими нулями и будем умножать их как $2N$-разрядные числа по модулю $2^{2N} + 1$.

Возьмём $l = [n/2]$, $k = n - l$ (т. е. $l + k = n$ и $k = l$ или $k = l + 1$), $K = 2^k$, $L = 2^l$ (при этом $K \cdot L = N$) и разобьём числа u и v на K групп по L разрядов:

$$u = (U_{K-1} \ldots U_0)_{2^L},\ v = (V_{K-1} \ldots V_0)_{2^L}.$$

Тогда произведение чисел u и v можно выразить через компоненты свёртки $y = u \otimes v$ следующим образом:

$$u \cdot v = \sum_{i=0}^{2K-1} y_i \cdot 2^{Li},$$

а их произведение по модулю $2^N + 1$ (в силу равенства $2^{KL} = 2^N \equiv -1$) – через компоненты отрицательно обёрнутой свёртки $W = u\ [-]\ v$:

$$u \cdot v \bmod (2^N + 1) = \sum_{i=0}^{K-1} y_i \cdot 2^{Li} - \sum_{i=0}^{K-1} y_{i+K} \cdot 2^{Li} = \sum_{i=0}^{K-1} W_i \cdot 2^{Li}.$$

Оценим коэффициенты W_i:

$$W_i = y_i - y_{i+K} = (U_i \cdot V_0 + U_{i-1} \cdot V_1 + \ldots + U_0 \cdot V_i) - (U_{K-1} \cdot V_{i+1} + U_{K-2} \cdot V_{i+2} + \ldots + U_{i+1} \cdot V_{K-1}).$$

Здесь первая скобка содержит $(i + 1)$ слагаемых, вторая $(K - 1 - i)$ слагаемых, и каждое из слагаемых вида $U_i \cdot V_j$ строго меньше, чем 2^{2L}. Поэтому

$$-(K - 1 - i)\cdot 2^{2L} < W_i < (i + 1)\cdot 2^{2L},$$

т. е. количество возможных значений W_i меньше, чем $K\cdot 2^{2L}$. Определим сначала значение $W_i \bmod (K\cdot(2^{2L} + 1))$; для этого вычислим

$$W_i' = W_i \bmod K,$$

$$W_i'' = W_i \bmod (2^{2L} + 1).$$

Поскольку K есть степень 2, а $(2^{2L} + 1)$ – нечётное, то K и $(2^{2L} + 1)$ взаимно просты. Кроме того, заметим, что $2^{2L} + 1 = 1 \pmod K$. Значение $W_i \bmod (K\cdot(2^{2L} + 1))$ найдём по *формуле Гарнера*:

$$W_i''' = (2^{2L} + 1)\cdot((W_i' - W_i'') \bmod K) + W_i'',$$

после чего, с учётом диапазона W_i, определим его точное значение:

$$W_i = \begin{cases} W_i''', & \text{если } W_i''' < (i+1)\cdot 2^{2L}, \\ W_i''' - K\cdot(2^{2L}+1) & \text{иначе.} \end{cases}$$

Наконец, последняя ключевая идея алгоритма. Значения W_i' находятся легко в силу малости K (за деталями можно обратиться к книге Д. Кнута «Искусство программирования для ЭВМ. Т. 2: Получисленные алгоритмы»). Значения же W_i'' найдём по теореме об отрицательно обёрнутой свёртке с помощью ДПФ и ОДПФ. Определим необходимые параметры этих преобразований – примитивный корень w, значение ψ, а также обратные элементы w^{-1}, ψ^{-1}, n^{-1}. Нам заданы модуль $m = 2^{2L} + 1$ и длины векторов u и v – это K. По теореме о примитивных корнях найдём примитивный корень степени K в Z_m: $m = 2^{2L} + 1 = w^{K/2} + 1$, откуда $w = 2^{4L/K}$. Ввиду $2^{2L} = -1 \bmod m$ получим $\psi = 2^{2L/K}$, $w^{-1} = -2^{2L-4L/K}$, $\psi^{-1} = -2^{2L-2L/K}$ и $K^{-1} = -2^{2L-k}$. Все показатели степеней целые в силу выбора параметров L и K.

Алгоритм Шёнхаге–Штрассена умножения чисел

Вход: u, v – N-разрядные двоичные числа, $N = 2^n$.

Выход: $y = u\cdot v \bmod (2^N + 1)$.

1. Положить $l = [n/2]$, $k = n - l$, $K = 2^k$, $L = 2^l$ и разбить числа u и v на K групп по L разрядов: $u = (U_{K-1}\ldots U_0)_{2^L}$, $v = (V_{K-1}\ldots V_0)_{2^L}$.

   Далее ищем вектор $W = u\,[-]\,v = (W_{K-1}\ldots W_0)_{2^L}$.

2. Для $i = 0, \ldots, K - 1$ найти

$$W_i' = W_i \bmod K = \left(\sum_{j=0}^{i} U_{i-j}\cdot V_j - \sum_{j=i+1}^{K-1} U_{i+K-j}\cdot V_j\right) \bmod K .$$

3. Найти $W_i'' = W_i \bmod (2^{2L} + 1)$, $i = 0, \ldots, K - 1$:
   a. Положить $\psi = 2^{2L/K}$ и построить векторы

$u' = [U_0 \ \psi \cdot U_1 \ldots \psi^{K-1} \cdot U_{K-1}]^T$, $v' = [V_0 \ \psi \cdot V_1 \ldots \psi^{K-1} \cdot V_{K-1}]^T$.

b. Положить $m = 2^{2L} + 1$, $w = 2^{4L/K}$ и найти ДПФ $u'' = F(u')$, $v'' = F(v')$ в Z_m.

c. Вычислить покомпонентное произведение векторов $c = u'' \cdot v''$.

d. Вычислить ОДПФ $d = F^{-1}(c)$, положив $w^{-1} = -2^{2L-4L/K}$ и $K^{-1} = -2^{2L-k}$.

e. Из вектора $d = [W_0'' \ \psi \cdot W_1'' \ldots \psi^{K-1} \cdot W_{K-1}'']^T$, с учётом $\psi^{-1} = -2^{2L-2L/K}$, найти все W_i''.

4. Для $i = 0, \ldots, K-1$ вычислить

$W_i''' = (2^{2L} + 1) \cdot ((W_i' - W_i'') \bmod K) + W_i''$ и

$$W_i = \begin{cases} W_i''', & \text{если } W_i''' < (i+1) \cdot 2^{2L}, \\ W_i''' - K \cdot (2^{2L} + 1) & \text{иначе.} \end{cases}$$

5. Ответ: $y = \sum_{i=0}^{2K-1} W_i \cdot 2^{Li}$.

Пример. Пусть $u = v = 21 = (10101)_2$.

Дополним числа u и v слева незначащими нулями до $2^3 = 8$ разрядов и добавим ещё 8 нулей слева с тем, чтобы вычислить точное произведение:

$$u = v = (0000\ 0000\ 0001\ 0101)_2.$$

Получили $N = 16 = 2^4$.

1. $l = 2, k = 2, K = 4, L = 4, u = v = (0\ 0\ 1\ 5)_{16}$.
2. По модулю 4: $W' = [1\ 2\ 1\ 0]$.
3. a. $\psi = 4, u' = v' = [5\ 4\ 0\ 0]$.
   b. $m = 257, w = 16, u'' = v'' = F(u') = [9\ \ 69\ \ 1\ \ -59]$.
   c. $c = [81\ \ 135\ \ 1\ \ 140]$.
   d. $d = F^{-1}(c) = [25\ \ 40\ \ 16\ \ 0]$.
   e. $W'' = [25\ \ 10\ \ 1\ \ 0]$.
4. $W''' = [25\ \ 10\ \ 1\ \ 0]$, и $W = [25\ \ 10\ \ 1\ \ 0]$.
5. $y = 25 + 160 + 256 = 441$.

Воскликнем вслед за Д. Кнутом: это кажется почти чудом, но мы получили правильный результат!

В заключение заметим, что алгоритм Шёнхаге–Штрассена выполняет умножение двух n-разрядных чисел за $O(n \cdot \log n \cdot \log\log n)$ шагов.

2. ГЕНЕРАЦИЯ ПРОСТЫХ ЧИСЕЛ

Введение

Задача генерации простых чисел относится к классическим вычислительным проблемам, известным с античных времен. С появлением криптографии с открытым ключом эта проблема приобрела не только теоретическое, но и большое практическое значение, так как большие случайные простые числа служат параметрами (открытыми или закрытыми) многих асимметричных криптосистем. Поскольку «формула» построения простого числа не известна (хотя её поисками занимались многие выдающиеся математики), то все методы генерации простых чисел действуют по следующей схеме: строится случайное нечётное число n нужной длины и затем проверяется на простоту одним из известных тестов.

В 1976 г. Х. Уильямс привел следующую классификацию методов проверки чисел на простоту (судя по более поздним публикациям, эта классификация не потеряла актуальности по настоящее время):

1. Детерминированные методы, использующие специальные функции и/или знание полного или частичного разложения числа $n - 1$ на простые множители. Эти методы математически строго доказывают простоту или составность проверяемого числа n.

2. Методы Монте-Карло (вероятностные) – не требуют разложения на простые множители, работают очень быстро, но не дают строгого доказательства простоты (составности) числа.

3. Методы гипотез – промежуточные между методами Монте-Карло и математически строгими методами. Основаны на истинности какой-либо недоказанной пока гипотезы (например, обобщённой гипотезы Римана): в предположении, что гипотеза верна, эти методы превращаются в детерминированные.

Все детерминированные методы, которые были известны до августа 2002 г., имеют экспоненциальную оценку сложности. В работе индийских математиков M. Agrawal, N. Kayal, N. Saxena с замечательным названием «PRIMES is in P» представлен детерминированный полиномиальный тест на простоту; однако на данный момент этот тест имеет скорее теоретическое, чем практическое значение, поскольку для чисел, используемых в современной криптографии (1–2 тысячи бит) он сводится к тривиальному методу поиска делителей числа.

В данной главе мы рассмотрим несколько вероятностных методов проверки числа на простоту, методы проверки простоты и генерации чисел специального вида – чисел Ферма, Мерсенна, надёжных и сильных простых чисел, а также два алгоритма генерации простых чисел с использованием детерминированных тестов на простоту: исторически первый практический алгоритм такого класса – метод Маурера и алгоритм генерации простого числа, описанный в старом Российском стандарте электронной цифровой подписи ГОСТ Р 34.10–94. Заметим, что в новом стандарте 2001 г. ГОСТ Р 34.10–2001 описание процедуры генерации простого числа отсутствует.

2.1. Метод Ферма

Метод основан на теореме Ферма:

Теорема Ферма. Если n – простое число, то из условия «n не делит a» следует: $a^{n-1} = 1 (\mathrm{mod}\ n)$.

Другими словами, если найдётся основание a, $1 \le a \le n-1$, для которого $a^{n-1} \neq 1 (\mathrm{mod}\ n)$, то можно с полной уверенностью утверждать, что n составное. Обратное неверно: для составного числа n может найтись a такое, что $a^{n-1} = 1 (\mathrm{mod}\ n)$.

Пример. Пусть $n = 65 = 5 \cdot 13$, $a = 12$. Значение $12^{64} \bmod 65$ вычислим с помощью *Китайской теоремы об остатках*, пользуясь тем, что разложение 65 на множители мы знаем.

$12^{64} \bmod 5 = 2^{64} \bmod 5 = (2^4)^{16} \bmod 5 = 1$;

$12^{64} \bmod 13 = (-1)^{64} \bmod 13 = 1$.

Отсюда получаем, что $12^{64} \bmod 65 = 1$.

Определение 1. Число n называется *псевдопростым по основанию* a, если $a^{n-1} = 1 (\mathrm{mod}\ n)$.

Так, в нашем примере число 65 – псевдопростое по основанию 12.

Итак, с помощью теоремы Ферма можно достоверно установить лишь составность числа, но нельзя доказать его простоту. Поэтому основанный на этой теореме метод является вероятностным.

Тест Ферма проверки числа на простоту

Вход: нечётное $n > 3$, параметр надёжности t.

Выход: ответ на вопрос: «n – простое?».

1. Выполнить t раз:
   1.1. Выбрать случайное a, $2 \le a \le n-2$.
   1.2. Вычислить $r = a^{n-1} \bmod n$.
   1.3. Если $r \neq 1$, выход, ответ: «составное».
2. Ответ: «простое».

В тесте Ферма ответ «составное» является стопроцентно верным; если же тест выдаёт ответ «простое», то этому можно доверять лишь с оглядкой. Оценим вероятность ошибки. Понятно, что она зависит от параметра надёжности t, причём, поскольку испытания разных оснований a независимы друг от друга, то вероятность ошибки равна ε^t, где ε – вероятность ошибки на одной итерации теста. В свою очередь, $\varepsilon \le \varphi(n)/n$, потому что количество «неблагоприятных» исходов не превосходит $\varphi(n)$ в силу импликации $(a, n) \ne 1 \Rightarrow a^{n-1} \ne 1 (\mathrm{mod}\, n)$. (Здесь $\varphi(n)$ – *функция Эйлера*, которая определена для любого положительного n и равна количеству чисел в ряду 1, ..., n, взаимно простых с n.) Граница $\varphi(n)/n$ для вероятности ошибки достижима, т. е. существуют такие n, которые псевдопросты по *всем* основаниям, взаимно простым с n. Такие числа называются числами Кармайкла.

Определение 2. Составное n называется *числом Кармайкла*, если для любого a, взаимно простого с n, выполняется $a^{n-1} = 1 (\mathrm{mod}\, n)$.

Известно, что чисел Кармайкла бесконечно много; всё же они довольно редки. Так, в диапазоне до 10^9 есть всего 646 чисел Кармайкла и примерно 51 тысяча простых чисел. Наименьшее число Кармайкла – это $561 = 3 \cdot 11 \cdot 17$.

Все числа Кармайкла характеризуются следующей теоремой.

Теорема Кармайкла (1912 г.)

Нечётное n является числом Кармайкла, если и только если выполнены следующие три условия:

1) n свободно от квадратов;
2) $p - 1 \mid n - 1$ для любого простого $p \mid n$;
3) n есть произведение по крайней мере трёх различных простых.

Доказательство.

Необходимость. Предположим, что $n = p^k \cdot t$, где p – простое, $k \ge 2$ и $(p, t) = 1$. Пусть g – *первообразный корень* по модулю p^k. Найдём число a как решение системы сравнений

$$\begin{cases} a = g (\mathrm{mod}\, p^k), \\ a = 1 (\mathrm{mod}\, t) \end{cases}$$

Поскольку n – число Кармайкла и $(a, n) = 1$, то имеет место равенство $a^{n-1} = 1 (\mathrm{mod}\, n)$, откуда $a^{n-1} = 1 (\mathrm{mod}\, p^k)$, т. е. $g^{n-1} = 1 (\mathrm{mod}\, p^k)$. По свойствам показателей из последнего равенства следует

$$O_{p^k}(g) = \varphi(p^k) = p^{k-1} \cdot (p - 1) \mid n - 1.$$

Отсюда:

1) $p^{k-1} \mid n - 1$, что в силу условия $p^{k-1} \mid n$ возможно только при $k = 1$;
2) $p - 1 \mid n - 1$.

Необходимость утверждений 1 и 2 теоремы доказана. Утверждение 3 докажем от противного. Предположим, n есть произведение двух простых, $n = p \cdot q$. По доказанному, $p - 1 | n - 1$ и $q - 1 | n - 1$. Получим

$$q - 1 | n - 1 = p \cdot (q - 1) + p - 1 \quad \Rightarrow q - 1 | p - 1;$$
$$p - 1 | n - 1 = q \cdot (p - 1) + q - 1 \quad \Rightarrow p - 1 | q - 1.$$

Значит, $p = q$, что невозможно в силу доказанной свободы n от квадратов. Необходимость доказана.

Достаточность. Пусть $n = p_1 \cdot \ldots \cdot p_k$, где p_i – различные простые, и $(a, n) = 1$. Тогда $(a, p_i) = 1$ для любого $i = 1, \ldots, k$, откуда по теореме Ферма следует, что $a^{p_i - 1} = 1 (\mathrm{mod}\, p_i)$ и, ввиду $p_i - 1 | n - 1$, $a^{n-1} = 1 (\mathrm{mod}\, p_i)$ для любого $i = 1, \ldots, k$. В силу попарной простоты модулей отсюда получаем $a^{n-1} = 1 (\mathrm{mod}\, n)$, что означает, что n – число Кармайкла. Теорема доказана.

Введём ещё одно понятие. Будем называть число *а основанием Ферма* для n, если n является псевдопростым по основанию a; множество всех оснований Ферма, меньших n, будем обозначать F_n: $F_n = \{a \in Z_n : a^{n-1} = 1 (\mathrm{mod}\, n)\}$.

Теорема об основаниях Ферма

Множество оснований Ферма F_n является подгруппой *приведенной системы вычетов* $U(n)$.

Доказательство.

1. Докажем включение $F_n \subseteq U(n)$. Пусть $(a, n) = d$. Тогда $d | n$, $d | a^{n-1}$, откуда равенство $a^{n-1} = 1 (\mathrm{mod}\, n)$ достижимо только при условии $d | 1$.
2. Множество F_n замкнуто относительно обращения:
$$a \in F_n \Rightarrow a^{n-1} = 1 (\mathrm{mod}\, n) \Rightarrow (a^{-1})^{n-1} = (a^{n-1})^{-1} = 1^{-1} = 1 (\mathrm{mod}\, n) \Rightarrow a^{-1} \in F_n.$$
3. Множество F_n замкнуто относительно умножения:
$$a, b \in F_n \Rightarrow a^{n-1} = 1 (\mathrm{mod}\, n) \;\&\; b^{n-1} = 1 (\mathrm{mod}\, n) \Rightarrow (ab)^{n-1} = 1 (\mathrm{mod}\, n) \Rightarrow$$
$$ab \in F_n.$$

Свойства 1–3 означают, что F_n является подгруппой $U(n)$. Теорема доказана.

Следствие. Если n составное и не число Кармайкла, то $|F_n| \le \varphi(n)/2$.

Это верно ввиду теоремы Лагранжа: порядок собственной подгруппы F_n делит порядок группы $U(n)$, равный $\varphi(n)$.

2.2. Метод Соловея–Штрассена

Метод основан на критерии Эйлера, известном из курса теории чисел.

Критерий Эйлера. Пусть n – простое нечётное, a – целое. Тогда

$$\left(\frac{a}{n}\right) = a^{\frac{n-1}{2}} (\mathrm{mod}\, n).$$

На самом деле можно доказать более сильное утверждение:

Теорема о достаточности критерия Эйлера

Нечётное n является простым, если и только если $\left(\frac{a}{n}\right) = a^{\frac{n-1}{2}} (\mathrm{mod}\, n)$ для любого a.

Доказательство.

Необходимость имеет место в силу критерия Эйлера.

Докажем достаточность. Предположим, что n – составное, и выберем произвольное a с условием $(a, n) = 1$. Тогда $\left(\frac{a}{n}\right) = \pm 1$, откуда $a^{n-1} = \left(a^{\frac{n-1}{2}}\right)^2 = \left(\frac{a}{n}\right)^2 = (\pm 1)^2 = 1(\mathrm{mod}\, n)$. Значит, n есть число Кармайкла и ввиду теоремы Кармайкла имеет вид $n = p_1 \cdot \ldots \cdot p_k$, где p_i – различные простые. Выберем b – *квадратичный невычет* по модулю p_1, и решим систему

$$\begin{cases} x = b(\mathrm{mod}\, p_1), \\ x = 1(\mathrm{mod}\, p_2), \\ \ldots \\ x = 1(\mathrm{mod}\, p_k). \end{cases}$$

Вычислим символ Якоби $\left(\frac{x}{n}\right) : \left(\frac{x}{n}\right) = \left(\frac{b}{p_1}\right) \cdot \left(\frac{1}{p_2}\right) \cdot \ldots \cdot \left(\frac{1}{p_k}\right) = -1$. Поскольку $(x, n) = 1$, то должно выполняться равенство $x^{\frac{n-1}{2}} = \left(\frac{x}{n}\right) = -1(\mathrm{mod}\, n)$, которое влечёт $x^{\frac{n-1}{2}} = -1(\mathrm{mod}\, p_2)$, что не так в силу выбора x. Полученное противоречие доказывает теорему.

Как и в случае теоремы Ферма, критерий Эйлера может выполняться при каких-то основаниях для некоторых составных n.

Определение. Пусть $(a, n) = 1$. Число n называется *псевдопростым Эйлера по основанию a*, если выполнено равенство $\left(\frac{a}{n}\right) = a^{\frac{n-1}{2}} (\mathrm{mod}\, n)$.

По аналогии с основаниями Ферма введём в рассмотрение множество оснований Эйлера: $E_n = \{a \in U(n): \left(\frac{a}{n}\right) = a^{\frac{n-1}{2}} \pmod n\}$.

Свойства оснований Эйлера.

1. $E_n \subseteq F_n$.

Верно ввиду цепочки импликаций

$$a \in E_n \Rightarrow \left(\frac{a}{n}\right) = a^{\frac{n-1}{2}} \pmod n = \pm 1 \Rightarrow a^{n-1} = 1 \pmod n \Rightarrow a \in F_n.$$

2. $E_n = U(n)$ для простого n, $E_n \neq U(n)$ для составного n (т. е. по отношению к критерию Эйлера чисел, подобных числам Кармайкла, нет).

Здесь первое утверждение – следствие необходимости критерия Эйлера для простого n, второе – следствие его достаточности.

3. E_n есть подгруппа $U(n)$.

4. $|E_n| \leq \varphi(n)/2$ для составного n.

Последние два свойства доказываются аналогично подобным утверждениям об основаниях Ферма.

Пример. В предыдущем разделе мы убедились, что составное число 65 является псевдопростым по основанию 12, т. е. $12 \in F_{65}$. Проверим выполнение критерия Эйлера для 65 и 12.

$$\left(\frac{12}{65}\right) = \left(\frac{4}{65}\right)\cdot\left(\frac{3}{65}\right) = \left(\frac{2}{3}\right) = -1;$$

$12^{32} = 1 \pmod{65}$, так как $12^{32} = 1 \pmod{13}$, $12^{32} = (2^4)^8 = 1 \pmod 5$.

Таким образом, 65 не является псевдопростым Эйлера по основанию 12: $12 \notin E_{65}$. Опробуем основание 14:

$$\left(\frac{14}{65}\right) = \left(\frac{2}{65}\right)\cdot\left(\frac{7}{65}\right) = \left(\frac{7}{65}\right) = \left(\frac{2}{7}\right) = 1;$$

$14^{32} = 1 \pmod{65}$ в силу $14^{32} = 1 \pmod{13}$ и $14^{32} = (-1)^{32} = 1 \pmod 5$. Получили, что $14 \in E_{65}$.

Из включения $E_n \subseteq F_n$ следует, что тест, основанный на проверке критерия Эйлера, более точен, нежели тест Ферма. Однако несмотря на достаточность критерия Эйлера, построить абсолютно достоверный тест на его основе не удаётся в силу того, что перебрать все основания (как того требует теорема) практически невозможно.

Тест Соловея–Штрассена проверки числа на простоту

Вход: нечётное $n > 3$, параметр надёжности t.

Выход: ответ на вопрос: «n – простое?».

1. Выполнить t раз:

1.1. Выбрать случайное a, $2 \le a \le n-2$.

1.2. Вычислить $r = a^{\frac{n-1}{2}} \bmod n$.

1.3. Если $r \ne \pm 1$, выход, ответ: «составное».

1.4. Вычислить $s = \left(\frac{a}{n}\right)$.

1.5. Если $r \ne s \pmod n$, выход, ответ: «составное».

2. Ответ: «простое».

Как и в тесте Ферма, ответу «составное» теста Соловея–Штрассена доверяем на 100%. Если же тест выдаёт ответ «простое», то возможна ошибка с вероятностью ε', где ε – вероятность ошибки на одной итерации, и $\varepsilon \le \varphi(n)/(2n) < 1/2$.

В заключение этого раздела приведём алгоритм вычисления символа Якоби. Алгоритм рекурсивный и основан на следующих свойствах символа Якоби:

1. $\left(\frac{0}{n}\right) = 0, \quad \left(\frac{1}{n}\right) = 1$.

2. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\cdot\left(\frac{b}{n}\right)$.

3. $\left(\frac{2}{n}\right) = \begin{cases} 1, & \text{если } n = 1 \pmod 8 \text{ или } n = 7 \pmod 8, \\ -1 & \text{иначе.} \end{cases}$

4. $\left(\frac{a}{n}\right) = \left(\frac{a \bmod n}{n}\right)$.

5. Закон взаимности: если a нечётное, то

$$\left(\frac{a}{n}\right) = \begin{cases} \left(\frac{n}{a}\right), & \text{если } a = 1 \pmod 4 \text{ или } n = 1 \pmod 4, \\ -\left(\frac{n}{a}\right), & \text{если } a = 3 \pmod 4 \text{ и } n = 3 \pmod 4. \end{cases}$$

Алгоритм вычисления символа Якоби **Jacobi(*a*, *n*)** (рекурсивный)

1. Если $a = 0$, ответ: 0; если $a = 1$, ответ: 1.
2. Представить a в виде $a = 2^k \cdot a_1$, где a_1 нечётное.
3. Если k чётное, то положить $s = 1$,
   иначе если $n = 1 \pmod 8$ или $n = 7 \pmod 8$, то положить $s = 1$,
   иначе положить $s = -1$.
4. Если $n = 3 \pmod 4$ и $a_1 = 3 \pmod 4$, положить $s = -s$.
5. Если $a_1 = 1$, ответ: s, иначе ответ: $s \cdot \text{Jacobi}(n \bmod a_1, a_1)$.

2.4. Тест Миллера–Рабина

Пусть n – простое нечётное число. Представим его в виде $n = 2^s \cdot r + 1$, где r нечётнос, $s \geq 1$. Выберем некоторое b, $2 \leq b \leq n - 2$, и построим последовательность

$$b^r \bmod n, b^{2r} \bmod n, ..., b^{2^j \cdot r} \bmod n, ..., b^{2^{s-1} \cdot r} \bmod n, b^{2^s \cdot r} \bmod n = 1 \qquad (*)$$

Последний элемент последовательности равен 1 в силу теоремы Ферма: $b^{n-1} = 1 \pmod n$. Возможны следующие варианты:

1) $b^r = 1 (\bmod n)$, и тогда вся последовательность состоит из 1;
2) 1 впервые встречается в середине последовательности; все стоящие за ней элементы, очевидно, тоже равны 1.

Обратим внимание на этот второй вариант и на самый правый не равный 1 элемент; пусть это $b^{2^j \cdot r} \bmod n$ для некоторого j, $0 \leq j \leq s - 1$, т. е. $b^{2^j \cdot r} \neq 1 (\bmod n)$ и $b^{2^{j+1} \cdot r} = ... = b^{2^s \cdot r} = 1 (\bmod n)$. Перепишем первое равенство из последней цепочки в виде

$$b^{2^{j+1} \cdot r} - 1 = 0 (\bmod n),$$

откуда

$$(b^{2^j \cdot r} - 1)(b^{2^j \cdot r} + 1) = 0 (\bmod n),$$

$$n \mid (b^{2^j \cdot r} - 1)(b^{2^j \cdot r} + 1),$$

что по *основному свойству простых* (если p простое и $p|ab$, то $p|a$ или $p|b$) означает $n \mid (b^{2^j \cdot r} - 1)$ или $n \mid (b^{2^j \cdot r} + 1)$.

Первое из этих условий невозможно в силу выбора j ($b^{2^j \cdot r} \neq 1 (\bmod n)$); значит, имеет место делимость $n \mid (b^{2^j \cdot r} + 1)$, откуда следует равенство $b^{2^j \cdot r} = -1 (\bmod n)$.

Итак, если n простое, то последовательность (*) целиком состоит из 1 или перед первой 1 стоит -1. В другой формулировке: если в последовательности (*) нет 1 или перед первой 1 стоит не -1, то n – составное. Назовём это свойство *критерием Рабина*. Обратное не верно; например, $n = 65 = 2^6 \cdot 1 + 1$, и для $b = 8$ последовательность (*) имеет вид

$$8, -1, 1, ...,$$

т. е. по отношению к $b = 8$ число 65 «ведёт себя» как простое.

Ранее установили, что 65 является псевдопростым Эйлера по основанию 14. Проверим выполнение критерия Рабина для $b = 14$:

$$14, 1, ...;$$

эта проверка, в отличие от критерия Эйлера, устанавливает составность числа 65.

Определение. Нечётное число n вида $n = 2^s \cdot r + 1$, где r нечётное, называется *строго псевдопростым по основанию* b, если $b^r = 1 (\mathrm{mod}\, n)$ или существует j, $0 \le j \le s-1$, такое, что $b^{2^j \cdot r} = -1 (\mathrm{mod}\, n)$.

Введём множество оснований Рабина: $R_n = \{b \in U(n)$: $b^r = 1 (\mathrm{mod}\, n) \vee \exists j,\ 0 \le j \le s-1,\ (b^{2^j \cdot r} = -1 (\mathrm{mod}\, n))\}$. Непосредственно проверяется, что $R_n \subseteq F_n$. Докажем включение $R_n \subseteq E_n$.

Теорема Селфриджа

Если n – строго псевдопростое по основанию b, то n – псевдопростое Эйлера по основанию b.

Доказательство.

Пусть число n представлено в виде $n = 2^s \cdot r + 1$, где r нечётное, и является строго псевдопростым по основанию b. Возможны два варианта:

1. $b^r = 1 (\mathrm{mod}\, n)$. В этом случае $\left(\frac{b}{n}\right) = \left(\frac{b}{n}\right)^r = \left(\frac{b^r}{n}\right) = \left(\frac{1}{n}\right) = 1$; здесь первое равенство имеет место в силу нечётности r и условия $\left(\frac{b}{n}\right) = \pm 1$.

С другой стороны, $b^{\frac{n-1}{2}} = b^{2^{s-1} \cdot r} = (b^r)^{2^{s-1}} = 1^{2^{s-1}} = 1 (\mathrm{mod}\, n)$. Таким образом, выполнено равенство $\left(\frac{b}{n}\right) = b^{\frac{n-1}{2}} (\mathrm{mod}\, n)$, и n является псевдопростым Эйлера по основанию b.

2. Для некоторого j, $0 \le j \le s-1$, выполнено условие $b^{2^j \cdot r} = -1 (\mathrm{mod}\, n)$. Разложим число n на простые множители: $n = p_1 \cdot \ldots \cdot p_l$. Из условия $b^{2^j \cdot r} = -1 (\mathrm{mod}\, n)$ следует равенство $b^{2^j \cdot r} = -1 (\mathrm{mod}\, p_i)$ для любого $i = 1, \ldots, l$, откуда по свойствам показателей получаем $O_{p_i}(b) \mid 2^j \cdot r$. С другой стороны, верно равенство $b^{2^{j+1} \cdot r} = 1 (\mathrm{mod}\, p_i)$, значит, $O_{p_i}(b) \mid 2^{j+1} \cdot r$. Следовательно, показатель числа b по модулю p_i имеет вид $O_{p_i}(b) = 2^{j+1} \cdot t_i$ для некоторого нечётного t_i. Опять же по свойствам показателей запишем: $O_{p_i}(b) \mid \varphi(p_i)$, т. е. $2^{j+1} \cdot t_i \mid p_i - 1$. Значит, множитель

p_i имеет вид $p_i = 2^{j+1} \cdot t_i \cdot x_i + 1$. Обозначим $k_i = t_i \cdot x_i$. Заметим, что $b^{2^j x_i} = -1 (\mathrm{mod}\, p_i)$. Вычислим *символ Лежандра*:

$$\left(\frac{b}{p_i}\right) = b^{\frac{p_i - 1}{2}} = b^{2^j t_i x_i} = (-1)^{x_i} = (-1)^{k_i};$$

здесь последнее равенство верно ввиду нечётности t_i. Теперь можем записать выражение для *символа Якоби*:

$$\left(\frac{b}{n}\right) = (-1)^{\sum_{i=1}^{l} k_i}. \qquad (1)$$

Осталось вычислить значение $b^{\frac{n-1}{2}} \mathrm{mod}\, n$ и убедиться, что оно то же самое. Выразим значение n через полученные для множителей p_i выражения.

$$n = p_1 \cdot \ldots \cdot p_l = (2^{j+1} \cdot k_1 + 1) \cdot \ldots \cdot (2^{j+1} \cdot k_l + 1) = 1 + 2^{j+1} \cdot \sum_{i=1}^{l} k_i (\mathrm{mod}\, 2^{2j+2}),$$

$$\frac{n-1}{2} = 2^{s-1} \cdot r = 2^j \cdot \sum_{i=1}^{l} k_i (\mathrm{mod}\, 2^{2j+1}),$$

$$2^{s-1-j} \cdot r = \sum_{i=1}^{l} k_i (\mathrm{mod}\, 2^{j+1}),$$

и, наконец, в силу нечётности r, $2^{s-1-j} = \sum_{i=1}^{l} k_i (\mathrm{mod}\, 2)$.

Тогда

$$b^{\frac{n-1}{2}} = b^{2^{s-1} r} = (b^{2^j \cdot r})^{2^{s-1-j}} = (-1)^{2^{s-1-j}} = (-1)^{\sum_{i=1}^{l} k_i}. \qquad (2)$$

Совпадение правых частей равенств (1) и (2) доказывает теорему.

В отличие от оснований Ферма и Эйлера, основания Рабина не образуют группу, в частности, множество оснований Рабина не замкнуто относительно операции умножения. Поэтому доказать утверждение о мощности множества R_n так же просто, как для оснований Ферма и Эйлера, не удаётся. Следующая теорема даёт оценку мощности множества оснований Рабина.

Теорема Рабина

Нечётное составное $n > 9$ является строго псевдопростым по не более чем $\varphi(n)/4$ основаниям.

Доказательство.

Проведём доказательство для частного случая n – модуля RSA. Доказательство общего случая отличается лишь большими техническими сложностями и не имеет принципиальных отличий от приводимого.

Пусть $n = 2^s \cdot r + 1 = p \cdot q$, где r – нечётное, p, q – различные простые. Пусть также $p = 2^i \cdot p_1 + 1$, $q = 2^k \cdot q_1 + 1$, где p_1, q_1 – нечётные и без ограничения общности $i \le k$. Введём в рассмотрение множества

$$P(n) = \{b \in U(n): b^r = 1 (\mathrm{mod}\ n)\}$$

и $$Q_j(n) = \{b \in U(n): b^{2^j r} = -1 (\mathrm{mod}\ n)\}, j = 0, 1, \ldots, s-1.$$

Заметим, что множества $P(n)$ и $Q_j(n)$ попарно не пересекаются и их объединение равно R_n. Оценим мощности введённых множеств с помощью следующей известной из курса теории чисел теоремы:

Пусть p – простое. Сравнение $x^n = a \pmod p$ имеет $(n, p-1)$ решений, если и только если $a^{\frac{p-1}{(n, p-1)}} = 1 (\mathrm{mod}\ p)$, и ни одного иначе.

1. Оценим $|P(n)|$. Сравнение $b^r = 1 (\mathrm{mod}\ n)$ равносильно системе сравнений

$$\begin{cases} b^r = 1 (\mathrm{mod}\ p) \\ b^r = 1 (\mathrm{mod}\ q) \end{cases}$$

из которых первое имеет $(r, p-1)$ решений, и в силу нечётности r это число равно (r, p_1). Аналогично, второе сравнение имеет $(r, q-1) = (r, q_1)$ решений. Таким образом, система, а значит, и исходное сравнение имеет $(r, p_1) \cdot (r, q_1)$ решений.

Получили $|P(n)| = (r, p_1) \cdot (r, q_1)$.

2. Оценим $|Q_j(n)|$. Сравнение $b^{2^j r} = -1 (\mathrm{mod}\ n)$ равносильно системе

$$\begin{cases} b^{2^j r} = -1 (\mathrm{mod}\ p), \\ b^{2^j r} = -1 (\mathrm{mod}\ q). \end{cases}$$

Первое сравнение имеет решения, если $(-1)^{\frac{p-1}{(2^j r, p-1)}} = 1 (\mathrm{mod}\ p)$, т. е. если число $\dfrac{p-1}{(2^j \cdot r, p-1)}$ чётное.

Поскольку $(2^j r, p-1) = 2^{\min(j, i)} \cdot (r, p_1)$ и (r, p_1) есть нечётное число, то решения существуют при $j < i$ и их количество равно $2^j \cdot (r, p_1)$. Проведя такие же рассуждения для второго сравнения, подсчитаем общее число решений системы: $2^{2j} \cdot (r, p_1) \cdot (r, q_1)$.

Получили $|Q_j(n)| = 2^{2j} \cdot (r, p_1) \cdot (r, q_1)$ при $j < \min(i, k) = i$ и $|Q_j(n)| = 0$ при $j \ge i$.

3. Теперь можем вычислить $|R_n|$:

$$|R_n| = |P(n)| + \sum_{j=0}^{i-1} |Q_j(n)| = (p_1, r)\cdot(q_1, r)\cdot(1 + \sum_{j=0}^{i} 2^{2j}) =$$

$$= (p_1, r)\cdot(q_1, r)\cdot(1 + \frac{2^{2i}-1}{3}) = (p_1, r)\cdot(q_1, r)\cdot\frac{2^{2i}+2}{3}.$$

Рассмотрим возможные случаи.

а) Пусть $i < k$. Тогда

$$(p_1, r) \le p_1 = \frac{p-1}{2^i},$$

$$(q_1, r) \le q_1 = \frac{q-1}{2^k} \le \frac{q-1}{2^{i+1}},$$

$$|R_n| \le \frac{(p-1)\cdot(q-1)}{2^{2i+1}}\cdot\frac{2^{2i}+2}{3} = \varphi(n)\cdot(\frac{1}{6} + \frac{1}{3\cdot 2^{2i}}) \le \varphi(n)\cdot(\frac{1}{6}+\frac{1}{12}) = \frac{\varphi(n)}{4}.$$

б) Пусть $i = k$. Докажем, что в этом случае невозможно одновременное выполнение двух равенств $(p_1, r) = p_1$ и $(q_1, r) = q_1$. В самом деле,

$$n - 1 = 2^s\cdot r = (2^i\cdot p_1 + 1)\cdot(2^i\cdot q_1 + 1) - 1 = 2^{2i}\cdot p_1\cdot q_1 + 2^i\cdot p_1 + 2^i\cdot q_1,$$

и из условия $p_1 | r$ следует $p_1 | q_1$, а из условия $q_1 | r$ следует $q_1 | p_1$. Понятно, что одновременно обе эти делимости не могут иметь места, так как иначе получим равенство $p_1 = q_1$, невозможное ввиду условия $p \ne q$.

Если $(p_1, r) < p_1$, то $(p_1, r) \le p_1/3$, поскольку p_1 и r нечётные. То же самое верно для (q_1, r), и хотя бы одно из неравенств $(p_1, r) < p_1$ или $(q_1, r) < q_1$ имеет место. Поэтому всегда выполняется неравенство $(p_1, r)\cdot(q_1, r) \le \frac{p_1\cdot q_1}{3}$. Теперь можем записать:

$$|R_n| \le \frac{p_1\cdot q_1}{3}\cdot\frac{2^{2i}+2}{3} = \frac{(p-1)\cdot(q-1)}{9}\cdot\frac{2^{2i}+2}{2^{2i}} = \varphi(n)\cdot\frac{1}{9}\cdot(1+\frac{1}{2^{2i-1}}) \le$$

$$\le \varphi(n)\cdot\frac{1}{9}\cdot(1+\frac{1}{2}) = \frac{\varphi(n)}{6} < \frac{\varphi(n)}{4}.$$

Теорема доказана.

Доказанная теорема позволит нам оценить вероятность ошибки основанного на проверке критерия Рабина теста Миллера–Рабина.

Тест Миллера–Рабина проверки числа на простоту

Вход: нечётное $n > 3$, параметр надёжности t.

Выход: ответ на вопрос: «n – простое?».

1. Представить n в виде $n = 2^s\cdot r + 1$, где r – нечётное.
2. Выполнить t раз:

2.1. Выбрать случайное b, $2 \le b \le n - 2$.

2.2. Вычислить $y = b^r \bmod n$.

2.3. Если $y \neq \pm 1$, выполнять:

2.3.1. Положить $j = 1$.

2.3.2. Пока $j < s$ и $y \neq -1 \pmod n$:

– положить $y = y^2 \bmod n$;

– если $y = 1$, выход, ответ: «составное»;

– увеличить j на 1.

2.3.3. Если $y \neq -1 \pmod n$, выход, ответ: «составное».

3. Ответ: «простое».

Ошибка в тесте Миллера – Рабина может возникнуть только в шаге 3, когда составное число будет принято за простое. Вероятность ошибки равна ε^t, где вероятность ошибки на одной итерации оценивается как $\varepsilon \leq \varphi(n)/(4n) < 1/4$.

Некоторые комментарии к тесту Миллера–Рабина:

1. В шаге 2.1 вместо генерации случайного числа b можно перебирать маленькие простые числа: $b = 2, 3, 5, 7, \ldots$ При этом, во-первых, не требуется трудоёмкой процедуры генерации случайного числа и, во-вторых, маленькое значение b упрощает шаг 2.2 при использовании дихотомического варианта возведения в степень «слева направо».

2. В диапазоне до $25 \cdot 10^9$ есть только одно составное число, сильно псевдопростое по четырём основаниям 2, 3, 5 и 7; это число 3215031751.

3. Доказан следующий факт: в предположении, что верна обобщённая гипотеза Римана, любое нечётное составное n не является строго псевдопростым хотя бы для одного основания $b < 2(\ln n)^2$. Это означает, что достаточно проверить основания только до этой границы, и можно будет с уверенностью констатировать простоту числа n. Иными словами, тест Миллера – Рабина превращается в детерминированный, к тому же полиномиальный алгоритм. Дело за малым – осталось доказать обобщённую гипотезу Римана...

4. В то же время известно следующее утверждение (Алфорд, Грэнвилл, Померанц): для любого конечного множества оснований существует бесконечно много составных чисел, строго псевдопростых по всем этим основаниям.

5. Существуют более тонкие оценки вероятности ошибки теста Миллера–Рабина, учитывающие размер проверяемого на простоту числа. Следующая таблица показывает зависимость параметра надёжности t, необходимого для достижения вероятности ошибки 2^{-80}, от k – размера числа в битах.

k	100	150	250	500	1000	≥1300
t	27	18	12	6	3	2

6. Сравним три изученных нами теста на простоту по двум характеристикам – сложности вычислений и точности (вероятности ошибки). Сравнительная характеристика алгоритмов даётся в следующей таблице:

Алгоритмы	Тест Ферма	Тест Соловея–Штрассена	Тест Миллера–Рабина
Вычисления, выполняемые на одной итерации	$a^{n-1} \bmod n$	$a^{\frac{n-1}{2}} \bmod n, \left(\frac{a}{n}\right)$	$a^{\frac{n-1}{2^k}}, k \geq 1$
Оценка вероятности ошибки на одной итерации	$\frac{\varphi(n)}{n}$	$\frac{\varphi(n)}{2n}$	$\frac{\varphi(n)}{4n}$

Таблица показывает, что тест Миллера–Рабина является более предпочтительным, чем тесты Ферма и Соловея–Штрассена, по обоим параметрам.

Связь теста Миллера–Рабина с задачей факторизации

В некоторых случаях, а именно в случае выбора в шаге 2.1 случайного b, которое является основанием Ферма, но не является основанием Рабина, тест Миллера–Рабина не только устанавливает составность проверяемого числа, но и может дать информацию для поиска его делителя. Изменим шаг 2.3.2 алгоритма следующим образом:

2.3.2. Пока $j < s$ и $y \neq -1 \pmod n$:

– положить $z = y^2 \bmod n$;

– если $z = 1$, выход, ответ: «составное»;

– увеличить j на 1, положить $y = z$.

Если на этом шаге алгоритм заканчивает работу, то имеем выполнение условия $y^2 = 1 \pmod n$ при $y \neq \pm 1 \pmod n$. Отсюда следует

$$n \mid y^2 - 1 = (y-1)\cdot(y+1), \text{ но } n \nmid y-1 \text{ и } n \nmid y+1.$$

Если p простое и $p \mid n$, то по основному свойству простых $p \mid y-1$ или $p \mid y+1$. Пусть, например, $p \mid y-1$. Тогда $p \mid (y-1, n)$, вследствие чего наибольший общий делитель $(y-1, n)$ является нетривиальным делителем n: он не равен 1 и не равен n.

Задача. Пусть $n = 3 \pmod 4$. Доказать: $R_n = E_n$.

Решение.

Включение $R_n \subseteq E_n$ доказано в теореме Селфриджа. Докажем обратное включение. По условию $n = 4k + 3 = 2q + 1$, где q – нечётное. Пусть $a \in E_n$. Это означает, что $\left(\frac{a}{n}\right) = a^{\frac{n-1}{2}} = a^q \bmod n$, откуда $a^q = \pm 1 \pmod n$ и $a \in R_n$.

В книге [9] (глава 2) можно найти ещё целый ряд интересных упражнений на тему «Псевдопростые числа»; настоятельно советуем читателю выполнить их.

2.4. Метод Люка

Этот метод так же, как и предыдущие, является вероятностным, но другого типа: простоту числа он устанавливает абсолютно точно, а вот ответ «составное» может быть неверным с некоторой ненулевой вероятностью. В основе метода лежит следующий критерий.

Критерий Люка Число n является простым, если и только если существует элемент, принадлежащий показателю $n-1$ по модулю n.

Доказательство.

Необходимость следует из общеизвестного факта теории чисел – существования первообразных корней по простому модулю.

Достаточность. Если существует элемент, принадлежащий показателю $n-1$ по модулю n, то по свойствам показателей выполняется $n-1 \mid \varphi(n)$, что ввиду неравенства $\varphi(n) \le n-1$ означает $\varphi(n) = n-1$. Последнее равносильно простоте числа n.

Метод Люка состоит в отыскании элемента, принадлежащего показателю $n-1$ по модулю n, т.е. элемента a, удовлетворяющего условиям:

1) $a^{n-1} = 1 \pmod n$ и

2) $a^{(n-1)/p} \ne 1 \pmod n$ для всех простых p, делящих $n-1$.

Если такой элемент будет найден, то простота числа n будет достоверно установлена; в противном случае утверждаем, но не гарантируем, что n есть число составное. Для проверки условия 2 нужно знать разложение $n-1$ на простые множители, и это является, конечно, самым слабым местом алгоритма. Реально метод Люка применим только для чисел специального вида (например, чисел Ферма) или специальным образом (с известным разложением $n-1$) сгенерированных чисел.

Метод Люка проверки числа на простоту

Вход: нечётное $n > 3$, параметр надёжности t, каноническое разложение $n-1 = p_1^{e_1} \cdot \ldots \cdot p_k^{e_k}$.

Выход: ответ на вопрос: «n – простое?».

1. Выполнить t раз:

1.1. Выбрать случайное a, $2 \le a \le n-2$.

1.2. Если $a^{n-1} \ne 1 \pmod n$, выход, ответ: «составное».

1.3. Если $a^{\frac{n-1}{p_i}} \ne 1 \pmod n$ для всех $i = 1, \ldots, k$, выход, ответ: «простое».

2. Ответ: «составное».

Комментарии к алгоритму:

1. Попутно с проверкой на простоту находим первообразный корень a.

2. Вероятность выбора в качестве a первообразного корня на каждой итерации в шаге 1 составляет

$$p = \frac{\varphi(n-1)}{n-1} = (1-\frac{1}{p_1})\cdot\ldots\cdot(1-\frac{1}{p_k}) \le \frac{1}{2}.$$

Таким образом, вероятность ошибки (невыбора первообразного корня на всех t итерациях) составляет $(1-p)^t \ge 2^{-t}$.

Известна модификация метода Люка, позволяющая снизить вероятность ошибки и основанная на следующей теореме.

Теорема (Бриллхарт, Лемер, Селфридж, 1975 г.)

Пусть $n > 2$ – нечётное и $n-1 = p_1^{e_1}\cdot\ldots\cdot p_k^{e_k}$. Тогда если для каждого $i = 1, \ldots, k$ найдётся a_i, удовлетворяющее условиям

$$a_i^{n-1} = 1 (\mathrm{mod}\, n)$$

и

$$a_i^{(n-1)/p_i} \ne 1 (\mathrm{mod}\, n),$$

то n – простое.

Доказательство.

Пусть n_i – показатель, которому принадлежит a_i по модулю n. Поскольку $a_i^{n-1} = 1 (\mathrm{mod}\, n)$, то по свойствам показателей $n_i \mid n-1$. Следовательно, $n_i = p_1^{s_1}\cdot\ldots\cdot p_k^{s_k}$, где $s_i \le e_i$ для $i = 1, \ldots, k$. С другой стороны, $n_i \nmid (n-1)/p_i = p_1^{e_1}\cdot\ldots\cdot p_i^{e_i-1}\cdot\ldots\cdot p_k^{e_k}$. Значит, $s_i = e_i$ и $p_i^{e_i} \mid n_i$. По свойствам показателей имеет место делимость $n_i \mid \varphi(n)$, а значит, и делимость $p_i^{e_i} \mid \varphi(n)$. Последнее верно для любого i и числа $p_i^{e_i}$ попарно просты, поэтому выполнено условие $n-1 \mid \varphi(n)$, возможное лишь в случае $\varphi(n) = n-1$ и простого n. Теорема доказана.

Способ модификации теста Люка с использованием доказанной теоремы очевиден. Как уже отмечалось, вероятность ошибки при этом снижается, поскольку появляется больше возможностей для доказательства простоты числа, однако исчезает приятный сопутствующий эффект алгоритма – первообразный корень теперь уже находится не всегда.

Пример. Пусть $n = 41$, $n-1 = 40 = 2^3 \cdot 5$.

Проверим второе условие теоремы для $a = 2$ и $a = 3$; равенство $a^{40} = 1 \pmod{41}$, очевидно, выполняется ввиду простоты числа 41.

$a = 2$. $2^8 = 256 = 10 \ne 1 \pmod{41}$,

$$2^{20} \bmod 41 = \left(\frac{2}{41}\right) = 1.$$

$\underline{a = 3}$. $3^8 = (3^4)^2 = (-1)^2 = 1 \pmod{41}$,

$$3^{20} \bmod 41 = \left(\frac{3}{41}\right) = \left(\frac{41}{3}\right) = \left(\frac{2}{3}\right) = -1 \neq 1.$$

Таким образом, значения $a = 2$ и $a = 3$ не являются первообразными корнями, но в совокупности доказывают простоту числа 41 ввиду теоремы Бриллхарта–Лемера–Селфриджа.

Критерий Люка породил целую группу тестов на простоту с обшим названием «$(n-1)$-методы», так как все они требуют знания (полного или частичного) разложения числа $n-1$ на простые множители. Существуют также и $(n+1)$-методы, использующие знание разложения числа $n+1$.

2.5. Числа Ферма

Прежде чем определить числа Ферма, докажем следующее утверждение.

Утверждение о простых вида $2^m + 1$

Если $p = 2^m + 1$ – простое, то $m = 2^n$.

Доказательство.

$2^m = -1 \pmod{p}$, следовательно, $O_p(2) \nmid m$.

$2^{2m} = 1 \pmod{p}$, значит, $O_p(2) \mid 2m$.

Из полученного заключаем, что $O_p(2) = 2r$, где $r \mid m$.

Далее, в силу того, что $2^{2r} = 1 \pmod{p}$ и p простое, имеем $2^r = -1 \pmod{p}$, или, что то же самое, $p = 2^m + 1 \mid 2^r + 1$. С учётом условия $r \mid m$ это возможно только при $r = m$. Итак, получили $O_p(2) = 2m$, и по свойствам показателей $2m \mid \varphi(p) = p - 1 = 2^m$. Следовательно, $m = 2^n$. Утверждение доказано.

Определение. Числами Ферма называются числа вида

$$F(n) = 2^{2^n} + 1.$$

Ввиду своей особой структуры числа Ферма легко подлаются проверке на простоту и сравнительно легко (по сравнению с произвольными числами того же размера) – факторизации. Все простые числа Ферма характеризуются следующей теоремой.

Теорема Пепина (1877 г.)

Число Ферма $F(n)$ простое, если и только если $3^{(F(n)-1)/2} = -1 \pmod{F(n)}$.

Доказательство.

Необходимость. В случае простоты $F(n)$ по критерию Эйлера запишем

$$3^{\frac{F(n)-1}{2}} = \left(\frac{3}{F(n)}\right) = \left(\frac{F(n)}{3}\right) = \left(\frac{2}{3}\right) = -1(\mathrm{mod}\, F(n)).$$

Достаточность. Заметим, что 2 – единственный простой делитель числа $F(n) - 1$. Поэтому простота числа $F(n)$ следует из критерия Люка, условия теоремы и того, что $3^{F(n)-1} = 1 \pmod{F(n)}$.

Теорема Пепина верна также в следующей формулировке.

Число Ферма $F(n)$, $n > 1$, простое, если и только если $5^{(F(n)-1)/2} = -1 \pmod{F(n)}$.

Доказательство аналогично. Необходимо только заметить, что $2^{2^n} = 4^{2^{n-1}} = (-1)^{2^{n-1}} = 1(\mathrm{mod}\, 5)$.

Утверждение о делителях чисел Ферма

Все простые делители $F(n)$, $n > 1$, имеют вид $2^{n+2} \cdot x + 1$.

Доказательство.

Пусть $q \mid F(n) = 2^{2^n} + 1$ и q – простое.

$2^{2^n} = -1(\mathrm{mod}\, q)$, значит, $O_q(2) \nmid 2^n$.

$2^{2^{n+1}} = 1(\mathrm{mod}\, q)$, откуда $O_q(2) \mid 2^{n+1}$.

Следовательно, $O_q(2) = 2^{n+1}$, и по свойствам показателей $2^{n+1} \mid \varphi(q) = q - 1$, откуда $q = 1 \pmod{2^{n+1}}$; в частности, $q = 1 \pmod 8$. Вычислим символ Лежандра: $\left(\frac{2}{q}\right) = 1 = 2^{\frac{q-1}{2}} (\mathrm{mod}\, q)$. Отсюда получаем $O_q(2) = 2^{n+1} \mid (q-1)/2$, и $q = 2^{n+2} \cdot x + 1$. Утверждение доказано.

Задача. Доказать простоту или найти делители чисел Ферма F(4), F(5).

Решение.

$F(4) = 2^{16} + 1$, его возможные простые делители имеют вид $d = 64x + 1$, и в случае составности числа F(4) среди них есть делитель, не превосходящий $[\sqrt{2^{16}+1}] = 256$. Будем поочерёдно перебирать возможные делители.

$x = 1$, $d = 65$ – не простое;

$x = 2$, $d = 129$ – не простое;

$x = 3$, $d = 193$ – простое число, и надо проверить условие $193 \mid 2^{16} + 1$. Для этого вычислим $2^{16} \bmod 193$:

$2^8 \bmod 193 = 63$, $2^{16} \bmod 193 = 63^2 \bmod 193 = 109 \neq -1$.

Значит, $193 \nmid 2^{16} + 1$.

$x = 4$, $d = 257 > 256$.

Вывод: F(4) – простое число.

Для числа F(5) читателю предлагается решить задачу самостоятельно.

2.6. Числа Мерсенна

Определение. Числами Мерсенна называются числа вида $M(n) = 2^n - 1$.

Особенности чисел Мерсенна:

1. Легко проверяется взаимная простота двух чисел Мерсенна, а именно: $(M(n), M(m)) = 1 \Leftrightarrow (n, m) = 1$.

2. Легко выполняется приведение по модулю $M(n)$: ввиду равенства $2^n = 1 \pmod{M(n)}$ для вычисления $x \bmod M(n)$ нужно разбить число x на группы по n двоичных разрядов и просуммировать числа, представленные этими группами.

3. Если n составное, то и $M(n)$ составное, что доказывается равенством $2^{ab} - 1 = (2^a - 1)\cdot(2^{a(b-1)} + 2^{a(b-2)} + \ldots + 2^a + 1)$.

Утверждение о делителях чисел Мерсенна

Если n – простое нечётное, то все простые делители числа Мерсенна $M(n)$ имеют вид $2\cdot n\cdot x + 1$.

Доказательство.

Пусть $q \mid 2^n - 1$ и q – простое. Ввиду равенства $2^n = 1 \pmod{q}$ имеет место делимость $O_q(2) \mid n$, откуда ввиду простоты n следует равенство $O_q(2) = n$. По свойствам показателей $n \mid q - 1$, значит, $q = n\cdot k + 1$, или, ввиду нечётности q и n, $q = 2\cdot n\cdot x + 1$.

Залача. Доказать простоту или найти делители чисел Мерсенна M(11) и M(13).

Решение.

M(11) = 2047, его возможные простые делители имеют вид $d = 22x + 1$, и в случае составности числа M(11) среди них есть делитель, не превосходящий $\sqrt{2047} = 45$.

$x = 1$, $d = 23$. Проверяем: $2047 = 23\cdot 89$.

Для числа M(13) читателю предлагается решить задачу самостоятельно.

2.7. Надёжные простые числа

К простым числам, используемым в криптографических приложениях, зачастую предъявляются дополнительные требования. Два примера таких простых чисел специального вида рассматриваются в этом и следующем разделах.

Во многих криптографических системах вычисления выполняются в группе простого порядка Z_p. Например, в схеме открытого распределения ключей Диффи–Хеллмана параметрами системы являются простой модуль p и первообразный корень g, порождающий всю группу $Z_p^* = \{1, \ldots, p-1\}$, закрытым ключом пользователя А служит $x \in Z_p^*$, его открытый ключ – элемент $g^x \bmod p$. На другом конце пользователь В получает $g^x \bmod p$ и возводит его в свою закрытую степень y. То же самое проделывает А, и в результате оба пользователя А и В имеют общий секретный ключ $k = g^{xy} \bmod p$. Стойкость системы основана на сложности задачи дискретного логарифмирования. Однако в случае неудачного выбора x можем получить элемент g^x малого порядка; кроме того, в процесс передачи значения g^x может вмешаться злоумышленник и подменить наше стойкое значение на некоторый элемент, опять же имеющий малый порядок. Поскольку $k \in \langle g^x \rangle$, то в случае, если порядок этой подгруппы небольшой, злоумышленник может найти секретный ключ k простым перебором. (Через $\langle a \rangle$ обозначается группа, порождённая элементом a: $\langle a \rangle = \{a, a^2, \ldots, a^h\}$, где h – порядок элемента a.)

Таким образом, пользователи системы Диффи–Хеллмана должны всё время заботиться о том, чтобы полученные ими числа не порождали подгрупп с малым количеством элементов. Одно из решений проблемы состоит в выборе числа p так, чтобы группа Z_p^* не имела подгрупп небольшого порядка. Но $|Z_p^*| = p - 1$, и эта группа имеет подгруппы порядка d для всех $d \mid p-1$. Проблема состоит в том, что число $p - 1$ чётное, значит, в Z_p^* всегда присутствует подгруппа порядка 2. За исключением этой подгруппы, наличия других подгрупп малых порядков можно избежать, потребовав, чтобы p было надёжным простым числом.

Определение. Простое число p вида $p = 2q + 1$, где q – также простое число, называется *надёжным простым числом*.

Группа Z_p^* для надёжного $p = 2q + 1$ имеет 4 подгруппы: две тривиальных порядков 1 и $p - 1$, подгруппу порядка 2, содержащую элементы 1 и $p - 1$, и подгруппу порядка q. Эта последняя есть не что иное, как группа *квадратичных вычетов* по модулю p, обозначаемая $Q(p)$.

Во многих криптосистемах (например, схеме неотрицаемой подписи Шаума–ван Антверпена, протоколах идентификации Шнорра, Окамото) используются надёжные простые числа и вычисления ведутся в группе $Q(p)$. Отметим преимущества использования этой группы по сравнению с группой Z_p^*.

1. Группа $Q(p)$ не имеет нетривиальных подгрупп, т. е. порядок всех её неединичных элементов одинаков и равен q.

2. При работе в группе $Z_p^* = \langle g \rangle$ в качестве ключей часто используются x – закрытый ключ и g^x – открытый. При этом злоумышленник легко узнает последний бит x, просто вычислив символ Лежандра $\left(\frac{g^x}{p}\right)$, потому что g^x является квадратичным вычетом по модулю p, если и только если x чётный.

3. Легко выполняется поиск порождающего элемента группы $Q(p)$ – достаточно взять любой элемент $a \in \{2, \ldots, p-2\}$ и вычислить $a^2 \bmod p$.

Генерация надёжного простого числа может быть основана на следующей теореме.

Теорема Чебышёва

Пусть p, q – простые нечётные и $p = 2q + 1$. Тогда:

1) если $q = 1 \pmod 4$, то 2 – первообразный корень по модулю p;
2) если $q = 3 \pmod 4$, то -2 – первообразный корень по модулю p.

Доказательство.

Простыми делителями $(p-1)$ – порядка первообразного корня – являются числа 2 и q. В этом случае, по теореме о первообразных корнях, для того чтобы g являлся первообразным корнем по модулю p, достаточно выполнения условий $g^2 \neq 1 \pmod p$ и $g^q \neq 1 \pmod p$. Для $g = \pm 2$ получим $g^2 = 4 \neq 1 \pmod p$, так как $p > 3$. Осталось проверить второе условие.

1) Пусть $q = 1 \pmod 4$. Тогда $p = 2 \cdot (4k + 1) + 1 = 8k + 3$. В этом случае

$$2^q \bmod p = 2^{\frac{p-1}{2}} \bmod p = \left(\frac{2}{p}\right) = -1.$$

2) Пусть $q = 3 \pmod 4$. Тогда $p = 2 \cdot (4k + 3) + 1 = 8k + 7$. Аналогично вычисляем

$$(-2)^q \bmod p = (-2)^{\frac{p-1}{2}} \bmod p = \left(\frac{-2}{p}\right) = \left(\frac{-1}{p}\right) \cdot \left(\frac{2}{p}\right) = -1.$$

Теорема доказана.

В соответствии с теоремой один шаг генерации надёжного простого числа состоит в следующем. Пусть простое q построено. Вычислим $p = 2q + 1$ и проверим условие $g^q = -1 \pmod p$ для $g = 2$ при $q = 1 \pmod 4$ и для $g = -2$ при $q = 3 \pmod 4$. Если условие выполнено, то надёжное простое p получено.

2.8. Сильные простые числа

Одно из требований к параметрам криптосистемы RSA состоит в том, что множители p и q модуля RSA должны быть не просто *простыми*, а *сильными простыми* числами.

Определение. Простое p называется *сильным простым числом*, если существуют большие простые числа r, s и t такие, что:

1) $r \mid p-1$;
2) $s \mid p+1$;
3) $t \mid r-1$.

В этом определении условия 1 и 2 противостоят атакам факторизации на модуль с множителем p – соответственно $(n-1)$- и $(n+1)$-методам, условие 3 противостоит, в частности, циклической атаке на криптосистему RSA.

Циклическая атака на RSA

Вспомним: параметром криптосистемы RSA является модуль $n = p \cdot q$, где p, q – различные простые, открытым текстом является $x \in Z_n$, и шифртекст получается возведением в степень по модулю: $y = x^e \bmod n$. Разные сообщения шифруются, естественно, в разные шифртексты, т. е. алгоритм шифрования при заданном ключе (e, n) представляет собой подстановку на множестве Z_n. Циклическая атака ставит целью чтение открытого текста без закрытого ключа и состоит в следующем. Будем последовательно получать $y^e, y^{e^2}, \ldots, y^{e^{k-1}}, y^{e^k}$ (все вычисления ведутся по модулю n), пока не выполнится условие $y^{e^k} = y$; тогда предпоследнее значение $y^{e^{k-1}}$ – это есть искомый открытый текст x. Требуемое условие обязательно рано или поздно выполнится, поскольку подстановки образуют конечную группу, в которой порядок каждого элемента конечен. Понятно, что на практике циклическая атака осуществима в случае, если значение k – порядка подстановки шифрования – небольшое. Установим связь значения k с условием 3 из определения сильного простого числа. Пусть множитель p модуля RSA есть сильное простое. Последовательно запишем

$$y^{e^k} = y (\bmod n),$$

$$y^{e^k} = y (\bmod p),$$

$$e^k \cdot \log y = \log y \ (\bmod p-1),$$

$$e^k \cdot \log y = \log y \ (\bmod r).$$

С большой вероятностью, а именно с вероятностью $(1 - 1/r)$, выполнено условие $(\log y, r) = 1$. В этом случае последнее сравнение равносильно следующему:

$$e^k = 1 \pmod r, \text{ откуда}$$
$$k \cdot \log e = 0 \pmod{r-1},$$
$$k \cdot \log e = 0 \pmod t,$$

и в случае $(\log e, t) = 1$, что выполняется с большой вероятностью $(1 - 1/t)$, получаем

$$k = 0 \pmod t, \text{ что влечёт } k \geq t.$$

Вывод: при большом t с большой вероятностью (т. е. для большинства значений e) порядок подстановки $y = x^e \bmod n$ большой, что, в свою очередь, делает неосуществимой циклическую атаку.

Известен следующий метод генерации сильных простых чисел. Метод использует теорему Дирихле об арифметической прогрессии, которая гласит: для любой пары взаимно-простых чисел a и b существует бесконечно много простых вида $a \cdot n + b$.

Метод Гордона генерации сильного простого числа (1984 г.)

1. Генерируются s и t – большие простые числа примерно одинакового размера.
2. Выбирается целое случайное i_0, находится первое простое число в прогрессии $r = 2 \cdot i \cdot t + 1$, $i = i_0, i_0 + 1, \ldots$
3. Вычисляется $p_0 = 2 \cdot (s^{r-2} \bmod r) \cdot s - 1$.
4. Выбирается целое случайное j_0, находится первое простое число в прогрессии $p = 2 \cdot j \cdot r \cdot s + p_0$, $j = j_0, j_0 + 1, \ldots$
5. Ответ: p.

Обоснование метода

Докажем, что для построенного p выполнены условия 1 – 3 из определения сильного простого числа.

По построению, в шаге 2 имеет место делимость $t \mid r - 1$.

В шаге 4 выполняются условия $p = p_0 \pmod r$ и $p = p_0 \pmod s$; в свою очередь, в шаге 3 имеем: $s \mid p_0 + 1$ (это проверяется непосредственно) и $r \mid p_0 - 1$ (это верно ввиду $(s^{r-2} \bmod r) \cdot s = s^{r-1} = 1 \pmod r$). Следовательно, $s \mid p + 1$ и $r \mid p - 1$.

2.9. Метод Маурера

Метод Маурера (1989 г.) является исторически первым методом генерации доказуемо простого числа. Он основан на следующей теореме.

Теорема Поклингтона (1914 г.)

Пусть для целого $n \geq 3$ выполняется $n = RF + 1$ и известно разложение F на простые множители: $F = \prod_{j=1}^{t} q_j^{e_j}$. Тогда если существует целое a, для которого выполнены условия:

1) $a^{n-1} \equiv 1 \pmod{n}$ и

2) для каждого j, $1 \leq j \leq t$, имеет место $(a^{(n-1)/q_j} - 1, n) = 1$,

то любой простой делитель числа n сравним с 1 по модулю F.

Доказательство.

Пусть p простое и $p \mid n$. Тогда из первого условия можно записать: $a^{n-1} \equiv 1 \pmod{p}$, откуда следует, что $O_p(a) \mid n - 1$. Из второго условия: $a^{\frac{n-1}{q_j}} \not\equiv 1 \pmod{p}$, откуда $O_p(a) \nmid \frac{n-1}{q_j}$. Значит, имеет место делимость $q_j^{e_j} \mid O_p(a) \mid p - 1$, а так как это справедливо для любого j, то $F \mid p - 1$, или $p \equiv 1 \pmod{F}$. Теорема доказана.

Следствие. Если, в условиях теоремы Поклингтона, $R < F$, то n – простое.

Доказательство от противного. Предположим, что n – составное. Тогда сушествуют простые p_1, p_2, делящие n, и $p_1 \cdot p_2 \leq n$. Так как по теореме $p_1 \equiv 1 \pmod{F}$, то $p_1 \geq F + 1$; аналогично $p_2 \geq F + 1$. Из условия $R < F$ следует неравенство $n < F^2 + 1$. Запишем

$$n < F^2 + 1 < (F + 1)^2 \leq p_1 \cdot p_2 \leq n.$$

Полученное противоречие доказывает следствие.

Опишем схему работы алгоритма Маурера. Пусть нужно построить простое число p длины t битов. Для этого строится цепочка простых чисел $p_s < p_{s-1} < \ldots < p_0 = p$, где каждое p_i имеет длину t_i. Полагаем $t_0 = t$, а длина t_{i+1} выбирается на каждом шаге рекурсии случайно из диапазона $[t_i/2] + 1 \leq t_{i+1} \leq t_i - 20$, если $t_i > 40$, и $t_{i+1} = [t_i/2] + 1$, если $t_i \leq 40$. В качестве p_s выбирается любое простое длины $t_s \leq 20$ битов. Каждое следующее простое получается из предыдущего в виде $p_i = p_{i+1} \cdot R + 1$, где чётное R выбирается случайно из диапазона, обеспечивающего необходимую длину p_i. Затем для $n = p_i$ и некоторого случайного a проверяются условия теоремы Поклингтона; если они выполнены, то очередное простое построено. При этом выполнение условия следствия $R < F = p_{i+1}$ гарантируется надлежащим выбором длины числа p_{i+1}.

Заметим, что алгоритм Маурера всегда в качестве результата выдаёт простое число. С другой стороны, поскольку алгоритм не занимается по-

иском оснований a, удовлетворяющих условиям теоремы Поклингтона, а лишь проверяет эти условия для одного случайного основания, то какие-то простые числа могут быть им не распознаны, и перебор будет продолжен, пока не будет построено число, простоту которого удастся доказать. Этот недостаток частично преодолевается в следующем алгоритме, основанном на более слабой теореме и, следовательно, отвергающем меньшее количество проверяемых чисел (как простых, так и составных).

2.10. Генерация простых чисел в Российском стандарте выработки электронной цифровой подписи ГОСТ Р34.10 – 94

Метод основан на теореме Диемитко.

Теорема Диемитко (1988 г.)

Пусть $n = qR + 1$, где q – простое. Тогда если существует целое a такое, что:

1) $a^{n-1} = 1 \pmod n$ и

2) $a^{(n-1)/q} \neq 1 \pmod n$,

то найдется простой делитель p числа n, для которого верно условие $p = 1 \pmod q$.

Доказательство.

Пусть $n = p_1^{e_1} \cdot \ldots \cdot p_k^{e_k}$. Из первого условия можно записать: $a^{n-1} = 1 \pmod{p_i^{e_i}}$ для любого $i = 1, \ldots, k$, откуда следует, что $O_{p_i^{e_i}}(a) \mid n-1$. Из второго условия следует, что найдётся j, для которого $a^{\frac{n-1}{q}} \neq 1 \pmod{p_j^{e_j}}$. Значит, $O_{p_j^{e_j}}(a) \nmid \frac{n-1}{q}$. Следовательно, имеет место делимость $q \mid O_{p_j^{e_j}}(a) \mid p_j^{e_j-1} \cdot (p_j - 1)$. Но в силу условия $(q, n) = 1$ выполнено также $(q, p_j^{e_j-1}) = 1$, поэтому $q \mid p_j - 1$, или $p_j = 1 \pmod q$. Теорема доказана.

Следствие. Если, в условиях теоремы Диемитко, R чётное и $R < 4(q + 1)$, то n – простое.

Доказательство.

Предположим, что $n = qR + 1$ – составное, p_j – его делитель такой, что $p_j = 1 \pmod q$, и $n = p_j \cdot Q$. Из условий $p_j = 1 \pmod q$ и $n = 1 \pmod q$ следует $Q = 1 \pmod q$; из чётности R следует нечётность n, а значит, нечётность p_j

и Q, что влечёт выполнение неравенств $p_j \geq 1 + 2q$ и $Q \geq 1 + 2q$. Можем записать:

$$n = p_j \cdot Q \geq (1 + 2q)^2 = q \cdot 4(q + 1) + 1 > q \cdot R + 1 = n.$$

Полученное противоречие доказывает следствие.

В рассматриваемом стандарте описаны процедуры B и B′ (отличающиеся друг от друга только используемыми датчиками случайных чисел) построения простого числа p длины t битов, $1021 \leq t \leq 1024$, такого, что число $(p - 1)$ имеет простой делитель q длины s битов, $255 \leq s \leq 256$. Эти процедуры, в свою очередь, используют процедуры A и A′ соответственно, позволяющие для любого целого $t > 16$ (процедура A) и $t > 32$ (процедура A′) строить простые числа p длины t битов с простым делителем q числа $(p - 1)$ длины $s = [t/2]$ битов. Всюду далее под *алгоритмом ГОСТ* будем понимать процедуру A.

Сначала строится последовательность длин $t_0, t_1, \ldots, t_s$ по правилам $t_0 = t$, $t_{i+1} = [t_i/2]$ для $i = 0, 1, \ldots, s - 1$, $t_s < 17$; в качестве p_s выбирается наименьшее простое длины t_s битов. Затем строится цепочка простых чисел $p_s < p_{s-1} < \ldots < p_0 = p$, где каждое p_i имеет длину t_i и каждое следующее простое получается из предыдущего в виде $p_i = p_{i+1} \cdot R + 1$. В качестве R выбирается случайное чётное число из диапазона $\frac{2^{t-1}}{q} < R < \frac{2^t - 1}{q}$. Проверяются условия теоремы Диемитко для построенного $n = p_i$ и фиксированного основания $a = 2$. Если условия выполнены, то очередное простое получено; в противном случае происходит увеличение значения R на 2 и проверка на простоту следующего кандидата $n = p_{i+1} \cdot R + 1$ до тех пор, пока длина числа n не превысит заданную. В этом последнем случае генерируется новое начальное значение R, и все повторяется сначала.

В алгоритме используются случайные числа в интервале (0, 1). Они получаются так. Выбираются случайные параметры y_0 и c (16-разрядные для процедур A и B и 32-разрядные для процедур A′ и B′). Строится последовательность случайных величин $\{y_i\}$ линейным конгруэнтным методом по правилам:

$$y_{i+1} = (19381 y_i + c) \bmod 2^{16} \text{ (процедуры A и B)},$$
$$y_{i+1} = (97781173 y_i + c) \bmod 2^{32} \text{ (процедуры A′ и B′)}.$$

Для получения случайного $\xi \in (0, 1)$ берутся очередные $r = [t/16]$ элементов этой последовательности (здесь t – длина в битах строящегося числа p), из них конструируется число $Y = \sum_{i=0}^{r-1} y_{k+i} \cdot 2^{16i}$ (на первом шаге $k = 0$,

после вычисления очередного Y полагается $k = k + r$) и вычисляется случайная величина $\xi = \frac{Y}{2^{16r}}$. Не отвлекаясь далее на детали генерации случайных величин, опишем один шаг алгоритма ГОСТ.

Алгоритм ГОСТ

Вход: t – требуемая длина числа в битах, q – простое число длины $[t/2]$ битов.

Выход: p – простое число длины t битов.

1. Вычислить $N = \left\lceil \frac{2^{t-1}}{q} \right\rceil + \left\lceil \frac{2^{t-1}}{q} \cdot \xi \right\rceil$. Если N нечётно, увеличить его на 1.
2. Положить $u = 0$.
3. Вычислить $p = q(N + u) + 1$.
4. Если $p > 2^t$, то переход к п. 1.
5. Если $2^{p-1} = 1 \pmod p$ и $2^{N+u} \neq 1 \pmod p$, выход, ответ: p. Иначе положить $u = u + 2$ и перейти к п. 3.

Проанализируем алгоритм ГОСТ с целью ответить на вопросы: всегда ли результатом работы алгоритма являются простые числа? и распознает ли алгоритм все простые числа?

Верхнюю границу для R в следствии из теоремы Диемитко, равную $4(q + 1) - 1$, увеличить нельзя, так как уже при $R = 4(q + 1)$ имеем $n = qR + 1 = 4q^2 + 4q + 1 = (2q + 1)^2$ – составное, но условия теоремы могут быть выполнены для некоторого a. Однако по процедуре, описанной в алгоритме ГОСТ, такое R может быть построено. Например, для $q = 5$ можем получить $R = 24$, соответственно $n = 121 = 11^2$; при $a = 3$ будем иметь $3^{120} = 1 \pmod{121}$, $3^{24} = 81 \neq 1 \pmod{121}$ – выполнение условий теоремы Диемитко и вывод о простоте числа 121. В литературе приводятся и другие подобные примеры, для чисел большего размера. В связи с этим делается вывод о том, что результатом работы алгоритма ГОСТ не всегда является простое число. Вместе с тем отмечается, что простые q, для которых условия теоремы выполняются при $R = 4(q + 1)$ и $a = 2$, неизвестны. Возможно, для «реабилитации» алгоритма ГОСТ требуется более глубокий теоретический анализ случая $a = 2$.

Второй обсуждаемый вопрос – распознаёт ли алгоритм все простые числа – имеет большое значение при оценке его эффективности, поскольку основные временные затраты алгоритма связаны с перебором большого количества чисел, прежде чем будет построено число, простоту которого удастся доказать. Поскольку алгоритм проверяет условия теоремы Диемитко для одного конкретного основания, то какие-то простые числа могут быть им не распознаны.

Студентом Артёмом Беловым был проведен следующий компьютерный эксперимент [3]. Все числа, порождаемые в ходе работы алгоритма (как отвергаемые, так и принимаемые им в качестве простых), дополнительно проверялись на простоту вероятностным тестом Миллера – Рабина. Было сгенерировано 100000 простых чисел, размер которых выбирался случайно в диапазоне от 512 до 1024 битов; это значит, что в процессе работы алгоритм построил примерно 10^6 простых различной длины. При этом ни одно из построенных чисел не было определено тестом Миллера–Рабина как составное. Было обнаружено всего 3 простых числа, не прошедших проверку условий теоремы Диемитко. Это следующие числа:

1) $n = 69431 = 1 + 530 \cdot 131$, $q = 131$ – встретилось 188 раз;
2) $n = 1326467 = 1 + 2546 \cdot 521$, $q = 521$ – 82 раза;
3) $n = 8540515853 = 1 + 1504 \cdot 32771$, $q = 32771$ – всего 3 раза.

Для каждой пары условие теоремы Диемитко выполняется для основания $a = 3$. Заметим, что все найденные такие числа имеют небольшие размеры, т. е. ошибки возникали только в начале работы алгоритма, когда генерация очередного кандидата в простые происходит быстро. Это говорит о том, что их влияние на эффективность алгоритма не критично.

Аналогичная программа, проверяющая алгоритм Маурера, выявила 19 ошибочно отброшенных чисел. Различных из них – 14, а именно:

1) $n = 2033839$, $q = 1097$;
2) $n = 1256303$, $q = 1399$;
3) $n = 1101811$, $q = 1933$;
4) $n = 1681003$, $q = 1987$;
5) $n = 3033169$, $q = 2179$;
6) $n = 5684759$, $q = 2309$;
7) $n = 2099863$, $q = 2713$;
8) $n = 3188767$, $q = 3301$;
9) $n = 7278269$, $q = 3947$;
10) $n = 15358129$, $q = 6037$;
11) $n = 21993703$, $q = 6581$;
12) $n = 13664473$, $q = 7207$;
13) $n = 45592577$, $q = 11131$;
14) $n = 73984367$, $q = 12401$.

Для всех перечисленных пар чисел условия теоремы Поклингтона выполняются для основания $a = 3$.

Большее количество *различных* простых чисел, пропущенных алгоритмом Маурера, может быть объяснено более жесткими условиями теоремы Поклингтона. Тот факт, что *общее* количество ошибок в алгоритме ГОСТ больше, объясняется меньшей свободой выбора чисел в нем, чем в алгоритме Маурера.

3. МЕТОДЫ ФАКТОРИЗАЦИИ

Введение

Задача факторизации ставится так: для заданного числа n найти его каноническое разложение на простые сомножители

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}.$$

На сложности этой задачи основана стойкость многих криптосистем с открытым ключом, например RSA, Рабина и др. На настоящий момент не найдено полиномиальных алгоритмов решения этой задачи и не доказано, что их не существует. Некоторые предварительные замечания:

1. Проверка числа на простоту выполняется проще, чем факторизация. Поэтому прежде чем факторизовать число, с помощью какого-либо теста убеждаются, что оно составное.

2. Наряду с полной факторизацией числа рассматривается задача его расщепления (2-факторизация, splitting): представление n в виде $n = a \cdot b$, где a, b – нетривиальные делители, не обязательно простые; $1 < a, b < n$. Понятно, что рекурсивное применение алгоритма 2-факторизации приводит в конце концов к полной факторизации числа.

3. Существует эффективный способ проверки, является ли n степенью какого-либо числа, т.е. представимо ли оно в виде $n = x^k$ для целых $x, k > 1$. Для этого перебираем все простые k в диапазоне $2 \le k \le [\log_2 n]$ и для каждого из них пытаемся найти x, удовлетворяющий условию $n = x^k$, извлекая корень k-й степени из n или применяя метод двоичного поиска x в интервале $]2, 2^{[(\log n)/k]+1}[$.

Все методы факторизации можно разделить на методы специального назначения, эффективные для чисел, имеющих делители особого вида, и методы общего назначения, работоспособность которых не зависит от вида числа. Первые имеют экспоненциальную сложность $O(c^k)$, где c – константа, k – размер факторизуемого числа, вторые – субэкспоненциальную $O(c^{(\log k)^2})$. По другому критерию методы делятся на детерминированные (всегда выдающие ответ) и вероятностные (типа Las-Vegas – ответ выдаётся не всегда, но если выдаётся, то он стопроцентно верен).

3.1. Метод пробных делений

Метод относится к методам спецназначения, которые хорошо ищут маленькие делители числа. Чаще всего применяется в качестве предварительного этапа перед более сложными методами факторизации, если речь идёт о разложении произвольного числа (понятно, что нет смысла применять этот метод для факторизации, скажем, модуля RSA). Его эффективность основана на том, что большинство чисел имеет маленькие делители; так, каждое второе делится на 2, каждое третье – на 3, и т.д. Для применения метода строится последовательность пробных делителей $d_1 < d_2 < \ldots < d_s$. Для гарантии полного разложения числа n должны выполняться условия: $\{d_1, d_2, \ldots, d_s\}$ содержит все последовательные простые до границы $d_s > \sqrt{n}$, причём достаточно только одного пробного делителя, удовлетворяющего этому неравенству, в силу того, что всякое составное число n имеет делитель, не превосходящий $\sqrt{n}$. Способы построения последовательности пробных делителей мы обсудим позднее.

Алгоритм пробных делений

Вход: число $n > 1$, последовательность пробных делителей $\{d_1, d_2, \ldots, d_s\}$.

Выход: представление n в виде $n = p_1 \cdot p_2 \cdot \ldots \cdot p_t$, где p_i – простые, не обязательно различные.

1. Положить $t = 0$, $k = 1$.
2. Если $n = 1$, выход.
3. Поделить n на d_k с остатком: $n = d_k \cdot q + r$.
4. Если $r = 0$, увеличить t на 1, положить $p_t = d_k$, $n = q$ и перейти к п.2.
5. Если $q > d_k$, увеличить k на 1 и перейти к п.3.
6. Иначе увеличить t на 1, положить $p_t = n$ и выход.

Обоснование шага 6 алгоритма

Нужно доказать, что в шаге 6 установлено, что n – простое. В самом деле, имеем $n = d_k \cdot q + r$, где $r < d_k$, $q \le d_k$. Это означает, что $n < d_k^2 + d_k = d_k \cdot (d_k + 1)$. С другой стороны, n не делится ни на одно простое, меньшее d_k, поэтому в случае составности n выполнено неравенство $n \ge (d_k + 1)^2$, что противоречит предыдущему.

Оценка сложности алгоритма

В худшем случае, когда факторизуемое число n оказывается простым, в алгоритме приходится выполнить порядка $\sqrt{n}$ делений, что даёт экспоненциальную от размера n оценку сложности.

Замечания по реализации:

1. В качестве последовательности $\{d_1, d_2, \ldots\}$ в случае нечётного n можно взять такую последовательность: {3, 5, 7, 11, 13, 17, 19, 23, 25, ...}. Она строится по закону: $d_1 = 3$, $d_2 = 5$, а далее шаги увеличения делителей чередуются: $d_3 = d_2 + 2$, $d_4 = d_3 + 4$, $d_5 = d_4 + 2$, $d_6 = d_5 + 4$ и т. д. Это позволяет исключить из последовательности числа, кратные 3, кроме самого числа 3, но, конечно, не гарантирует простоту всех пробных делителей. Можно сформулировать закон построения этой последовательности по-другому: $d_1 = 3$, $d_2 = 5$, $d_3 = 7$, $d_i = d_{i-2} + 6$ для $i > 3$. Небольшим усложнением закона построения последовательности можно добиться исключения из неё чисел, кратных 5. Естественно, нет смысла строить и хранить последовательность пробных делителей заранее; она получается в процессе работы алгоритма.

2. Для практического применения метода пробных делений приходится задавать ограничение B на величину пробных делителей. Граница B подбирается экспериментально, и в случае небольшой границы можно в качестве последовательности $\{d_1, d_2, \ldots, d_s\}$ взять заранее построенную таблицу простых чисел, меньших B.

3. В том случае, когда делители d_i ограничены не величиной $\sqrt{n}$, а искусственной границей B, метод пробных делений не гарантирует полной факторизации числа. Поэтому функция, реализующая его, должна возвращать следующие параметры:

– массив найденных простых делителей;

– признак того, разложено число полностью или нет;

– в случае неполного разложения – неразложенный остаток, с тем чтобы можно было атаковать его другими методами.

3.2. Метод Олвея

Главный недостаток предыдущего метода – многократное выполнение дорогостоящей операции деления. В 1952 г. Дж. Олвей (Alway) предложил модификацию, которую можно назвать «методом пробных делений без деления». Идея Олвея состоит в замене операции деления серией аддитивных операций, и в случае, если делители d_i достаточно велики, количество этих аддитивных операций оказывается небольшим.

Один из ключевых моментов метода пробных делений – деление n на d_k с остатком, т. е. представление его в виде $n = d_k \cdot q_k + r_k$. Если последовательность $\{d_k\}$ образует арифметическую прогрессию и делители достаточно велики по отношению к n, то, как мы покажем далее, последова-

тельность неполных частных $\{q_k\}$ образует «приблизительно арифметическую» прогрессию. Тогда остатки $\{r_k\}$ имеют примерно квадратичное поведение, что позволяет «угадывать» их без деления. Такова общая идея; докажем всё строго.

Пусть $\{d_k\}$ – арифметическая прогрессия нечётных чисел с шагом 2. Рассмотрим поведение последовательности $\{q_k\}$. Обозначим $\Delta q_k = q_k - q_{k-1}$ и оценим «разность разностей» $\Delta q_{k+1} - \Delta q_k = q_{k+1} - 2q_k + q_{k-1}$. Поскольку $q_k = \left[\frac{n}{d_k}\right]$, $d_{k+1} = d_k + 2$ и $d_{k-1} = d_k - 2$, то верны неравенства

$$\frac{n}{d_k} - 1 < q_k \le \frac{n}{d_k},$$

$$\frac{n}{d_k + 2} - 1 < q_{k+1} \le \frac{n}{d_k + 2},$$

$$\frac{n}{d_k - 2} - 1 < q_{k-1} \le \frac{n}{d_k - 2}.$$

Домножим первое неравенство на (-2), не забыв сменить знаки неравенства на противоположные, и просуммируем все три неравенства. Получим

$$C - 2 < \Delta q_{k+1} - \Delta q_k < C + 2,$$

где $$C = \frac{n}{d_k + 2} + \frac{n}{d_k - 2} - 2\frac{n}{d_k} = \frac{8n}{d_k \cdot (d_k^2 - 4)}.$$

Легко убедиться, что при $d_k \ge 3$ выполняется неравенство

$$0 < C \le \frac{8n}{(d_k - 1)^3}.$$

В свою очередь, верхняя граница для C влечёт оценку $C \le 1$ при выполнении условия $d_k \ge 2\sqrt[3]{n} + 1$. В этом случае (а мы будем далее предполагать только этот случай) получим

$$\Delta q_{k+1} - \Delta q_k \in \{-1, 0, 1, 2\}. \qquad (*)$$

По аналогии обозначим $\Delta r_k = r_k - r_{k-1}$ и оценим величину $\Delta r_{k+1} - \Delta r_k$:

$$\Delta r_{k+1} - \Delta r_k = r_{k+1} - 2r_k + r_{k-1} =$$

$$= n - d_{k+1} \cdot q_{k+1} - 2n + 2(d_{k+1} - 2) \cdot q_k + n - (d_{k+1} - 4) \cdot q_{k-1} =$$

$$= - d_{k+1} \cdot q_{k+1} + 2d_{k+1} \cdot q_k - 4q_k - d_{k+1} \cdot q_{k-1} + 4q_{k-1} = = - 4\Delta q_k - d_{k+1} \cdot (\Delta q_{k+1} - \Delta q_k).$$

Отсюда получаем два важных следствия:

1) $r_{k+1} = 2r_k - r_{k-1} - 4\Delta q_k \pmod{d_{k+1}}$,

2) в силу условия (*) для получения истинного значения r_{k+1} величина $2r_k - r_{k-1} - 4\Delta q_k$ подлежит лишь незначительной корректировке: не более чем двукратному уменьшению на d_{k+1} или, возможно, однократному увеличению на d_{k+1}.

Идея алгоритма Олвея состоит в «угадывании» по полученной формуле следующего значения остатка на основании двух предыдущих значений (на старте алгоритма всё же придётся дважды выполнить операцию деления) и корректировке его в случае непопадания в нужный диапазон.

Алгоритм Олвея нахождения нетривиального делителя числа

Вход: нечётное n, нечётное $d \geq 2\sqrt[3]{n}+1$.

Выход: делитель f числа n, удовлетворяющий условию $d < f \leq \sqrt{n}$, либо ответ «Делителя в указанном диапазоне нет».

Обозначения: r_1, r_2 соответствуют r_k и r_{k-1}, $q = -4\Delta q_k$.

1. Вычислить $r_1 = n \bmod d$, $r_2 = n \bmod (d-2)$,
   $q = 4\cdot([n/(d-2)] - [n/d])$, $s = [\sqrt{n}]$.
2. Увеличить d на 2. Если $d > s$, выход с ответом «Делителя нет».
3. Положить $r = 2r_1 - r_2 + q$, $r_2 = r_1$, $r_1 = r$.
4. Если $r_1 < 0$, увеличить r_1 на d и увеличить q на 4.
5. Пока $r_1 \geq d$, уменьшать r_1 на d и q на 4.
6. Если $r_1 = 0$, выход, ответ: d, иначе переход к п.2.

Замечания:

1. Условие $d \geq 2\sqrt[3]{n}+1$ на входе алгоритма влияет только на его эффективность: если условие выполнено, то по доказанному ранее в шаге 5 потребуется не более двух вычитаний; если же условием пренебречь, то алгоритм будет работать (в принципе), но количество вычитаний может оказаться неприемлемо большим.

2. Алгоритм относится к классу методов специального назначения, так как успешно ищет только большие делители числа.

Пример. Пусть $n = 575$, $d = 17$.

$575 = 17\cdot33 + 14 = 15\cdot38 + 5$.

d	r_1	r_2	q
17	14	5	20
19	43	14	
	24		16
	5		12
21	8	5	
23	23	8	
	0		8

Ответ: 23.

3.3. Метод Ферма

Этот метод также направлен на поиск большого (а именно ближайшего к $\sqrt{n}$) делителя числа n. Идея состоит в представлении факторизуемого числа в виде разности двух квадратов:

$$n = x^2 - y^2 = (x - y)(x + y).$$

Докажем, что для нечётного n такие x и y всегда существуют. В самом деле, пусть $n = a \cdot b$:

$$n = a \cdot b = \underbrace{\left(\frac{a+b}{2}\right)^2}_{x} - \underbrace{\left(\frac{a-b}{2}\right)^2}_{y};$$

здесь выражения в скобках целые ввиду нечётности a и b. Рассмотрим крайние случаи:

1) n – простое, т. е. $a = n$, $b = 1$. В этом случае $x = \frac{n+1}{2}$.

2) n есть полный квадрат, т. е. $a = b = \sqrt{n} = x$.

Алгоритм Ферма состоит в опробовании всех x от $[\sqrt{n}\,]$ до $\frac{n+1}{2}$.

Алгоритм Ферма поиска ближайших к $\sqrt{n}$ делителей числа n

Вход: n – нечётное.

Выход: a, b – делители n, или ответ: «n – простое».

1. Вычислить $x = [\sqrt{n}\,]$. Если $x^2 = n$, выход, ответ: $a = b = x$.
2. Увеличить x на 1.
3. Если $x = \frac{n+1}{2}$, выход, ответ: «n простое». Иначе вычислить $z = x^2 - n$, $y = [\sqrt{z}\,]$.
4. Если $y^2 = z$, выход, ответ: $a = x + y$, $b = x - y$. Иначе переход к п.2.

Замечание по реализации

В шаге 3 алгоритма z можно вычислить без возведения в квадрат, увеличивая предыдущее значение z на $2x - 1$ (имея в виду $x^2 = (x - 1)^2 + 2x - 1$).

Пример. Пусть $n = 111$.

x	$z = x^2 - n$	$y = [\sqrt{z}\,]$
11	10	3
12	33	5
13	58	7

14	85	8
15	114	10
16	145	12
17	178	13
18	213	14
19	250	15
20	289	17

Ответ: $111 = (20 + 17)(20 - 17) = 37 \cdot 3$.

3.4. Метод решета

Другое название метода – метод Ферма с просеиванием – отражает тот факт, что метод является модификацией метода Ферма. Идея просеивания состоит в следующем. Один из этапов метода Ферма – процедура проверки, является ли $z = x^2 - n$ полным квадратом. Заметим, что полный квадрат есть квадратичный вычет по любому модулю. Для иллюстрации идеи обратимся к последнему примеру. Легко установить, что все числа – полные квадраты имеют в десятичной системе счисления последнюю цифру из множества {0, 1, 4, 9, 6, 5}. Поэтому значения z, равные 33, 58, 178, 213, бесперспективны, и y для них можно не вычислять. Пойдём ещё дальше – установим бесперспективность значений x, равных 12, 13, 17, 18, и не будем для них вычислять z. Понятно, что выбор модуля, равного 10, объясняется лишь нашей привычкой к десятичной системе счисления, и совсем не обязательно рассматривать вычеты именно по этому модулю. Более того, можно взять не один, а несколько модулей и просеивать значения x сразу через несколько «решёт». Для того же примера выберем совокупность модулей {4, 5}.

mod 4 $111 \bmod 4 = 3$

x	0	1	2	3
x^2	0	1	0	1
$x^2 - n$	1	2	1	2

Из таблицы видно, что нет смысла рассматривать значения x, равные 1 или 3 по модулю 4. Для нашего примера это означает, что можно исключить все нечётные x.

mod 5 $111 \bmod 5 = 1$

x	0	1	2	3	4
x^2	0	1	4	4	1
$x^2 - n$	4	0	3	3	0

Здесь исключению подлежат x, равные 2 или 3 по модулю 5. Дополнительно к ранее исключенным пропускаем значения x, равные 12 и 18. Итого останется всего 3 значения x, которые надо будет проверить: 14, 16 и 20. Таблица иллюстрирует сказанное.

mod 5	mod 4	mod 10	x	$z = x^2 - n$	$y = [\sqrt{z}\,]$
	X		11	10	3
X		X	12	33	5
X	X	X	13	58	7
			14	85	8
	X		15	114	10
			16	145	12
X	X	X	17	178	13
X		X	18	213	14
	X		19	250	15
			20	289	17

Алгоритм решета поиска ближайших к $\sqrt{n}$ делителей числа n

Вход: n – нечётное.

Выход: a, b – делители n, или ответ: «n – простое».

1. Выбрать совокупность попарно простых модулей $M = \{m_1, \ldots, m_r\}$ и построить таблицу просеивания S с элементами s_{ij}, $i = 1, \ldots, r$, $j = 0, \ldots, m_i - 1$:

$$s_{ij} = \begin{cases} 1, \text{ если } j^2 - n \in Q(m_i) \text{ или } j^2 - n = 0 (\mathrm{mod}\, m_i), \\ 0 \text{ иначе.} \end{cases}$$

2. Вычислить $x = [\sqrt{n}\,]$. Если $x^2 = n$, выход, ответ: $a = b = x$.
3. Увеличить x на 1 и вычислить $k_i = x \bmod m_i$, $i = 1, \ldots, r$.
4. Просеивание: если $s_{ik_i} = 1$ для всех $i = 1, \ldots, r$, вычислить $z = x^2 - n$, $y = [\sqrt{z}\,]$, и если $y^2 = z$, выход, ответ: $a = x + y$, $b = x - y$.
5. Увеличить x на 1. Если $x = \dfrac{n+1}{2}$, выход, ответ: «n – простое». Иначе положить $k_i = (k_i + 1) \bmod m_i$ для $i = 1, \ldots, r$, перейти к п. 4.

Замечания:

1. Если выбирать модули m_i простыми, то проверка принадлежности числа множеству квадратичных вычетов сводится к вычислению символа Лежандра.

2. Попарная простота модулей даёт тот эффект, что все «решёта» работают независимо друг от друга.

3. Если все m_i – простые, то примерно половина элементов в каждой строке таблицы S – нули. С учётом замечания 2 это приводит к тому, что

в процессе работы алгоритма понадобится вычислять z для одного из примерно 2^r значений x.

Пример. Пусть $n = 1111$, $M = \{3, 4, 5, 7\}$.

$[\sqrt{1111}] = 33$, $1111 = 1 \pmod 3$, $1111 = 3 \pmod 4$, $1111 = 1 \pmod 5$, $1111 = 5 \pmod 7$.

Таблица просеивания S:

$m_i \backslash j$	0	1	2	3	4	5	6
3	0	1	1				
4	1	0	1	0			
5	1	1	0	0	1		
7	1	0	0	1	1	0	0

С учётом второй строки таблицы просеивания будем рассматривать только чётные значения x и, соответственно, увеличивать остатки k_i не на 1, а на 2. В примере выделены первые значения k_i, по которым не прошло просеивание (т. е. $s_{ik_i} = 0$).

x	k_i				$z = x^2 - n$	$y = [\sqrt{z}]$
34	1	2	4	**6**		
36	**0**	0	1	1		
38	2	2	**3**	3		
40	1	0	0	**5**		
42	**0**	2	2	0		
44	2	0	4	**2**		
46	1	2	1	4	1005	31
48	**0**	0	3	6		
50	2	2	0	**1**		
52	1	0	**2**	3		
54	**0**	2	4	5		
56	2	0	1	0	2025	45

Ответ: $1111 = (56 + 45)(56 - 45) = 101 \cdot 11$.

3.5. ρ-Метод Полларда

Этот метод является вероятностным типа Las-Vegas, т.е. иногда он заканчивает свою работу безрезультатно. Сначала познакомимся с методом определения длины периода последовательности, который используется в ρ-методе Полларда.

3.5.1. Метод Флойда вычисления длины периода

Пусть задано некоторое конечное множество S, $|S| = m$, x_0 – случайный элемент из S, и случайная функция f: $S \to S$. Рассмотрим последователь-

ность $\langle x\rangle = x_0, x_1, x_2, \ldots$, которая получается по правилу $x_{i+1} = f(x_i)$ для $i = 0, 1, \ldots$ В силу конечности множества S последовательность $\langle x\rangle$ периодическая с матожиданием длины предпериода $\lambda = \sqrt{\pi m/8}$ и длины периода $\mu = \sqrt{\pi m/8}$. Очевидный способ нахождения длины периода – вычислять и хранить значения $x_0, x_1, x_2, \ldots$ до первого совпадения. В среднем придётся вычислить $\lambda + \mu = \sqrt{\pi m/2}$ значений; таким образом, требование по памяти и по времени работы алгоритма составляет $O(\sqrt{m})$.

Метод Флойда позволяет обойтись без хранения всех значений x_i. Идея его состоит в следующем. Начиная с пары (x_0, x_0), вычисляем (x_i, x_{2i}), $i = 1, 2, \ldots$, из предыдущей пары (x_{i-1}, x_{2i-2}) следующим образом: $x_i = f(x_{i-1})$, $x_{2i} = f(f(x_{2i-2}))$. Вычисления продолжаем до тех пор, пока не произойдёт совпадение $x_i = x_{2i}$; а это случится обязательно, поскольку расстояние между x_i и x_{2i} всякий раз увеличивается на 1. Первое совпадение наступит при $i = \mu(1 + [\lambda/\mu])$ – наименьшем индексе, кратном μ и большем, чем λ. Поэтому временная сложность алгоритма та же – $O(\sqrt{m})$, но по сравнению с тривиальным методом существенно меньше требования по памяти – в каждый момент времени храним только два значения из последовательности.

Пример. Пусть $f(x) = (x^2 + 1) \bmod 10$, $x_0 = 3$.

i	0	1	2	3	4	5	6
x_i	3	0	1	2	5	6	7
x_{2i}	3	1	5	7	1	5	7

Ответ: период последовательности – делитель числа 6; в данном случае он равен 6.

3.5.2. ρ-Метод Полларда поиска нетривиального делителя числа

Пусть дано число n и нужно найти его нетривиальный делитель d. Рассмотрим функцию $f: Z_n \to Z_n$, $f(x) = (x^2 + 1) \bmod n$, и построим последовательность $\langle x\rangle = x_0, x_1, x_2, \ldots, x_{i+1} = f(x_i)$. Эта последовательность периодическая.

Рассмотрим (виртуально) другую последовательность $\langle c\rangle = c_0, c_1, c_2, \ldots$, где $c_i = x_i \bmod d$. Последовательность также периодическая, причём поскольку $|Z_n| < |Z_d|$, то с большой вероятностью период последовательности $\langle c\rangle$ меньше, чем период $\langle x\rangle$. Это значит, что найдутся такие i, j, что $c_i = c_j$, но $x_i \neq x_j$. Или, что то же самое: $x_i \neq x_j \bmod n$ и $x_i = x_j \bmod d$. Перепишем эти условия ещё раз в виде $n \nmid x_i - x_j$ и $d \mid x_i - x_j$.

Теперь стало совсем очевидным, что наибольший общий делитель $(x_i - x_j, n)$ не равен 1 и не равен n, т.е. является искомым нетривиальным делителем числа n. Осталось лишь добавить, что период последовательности $<c>$ ищется методом Флойда.

Алгоритм поиска нетривиального делителя числа

Вход: составное n.

Выход: d – нетривиальный делитель n, или отказ.

1. Положить $a = 2$, $b = 2$.
2. Положить $a = f(a)$, $b = f(f(b))$ для $f(x) = (x^2 + 1) \bmod n$.
3. Если $a = b$, выход без результата.
4. Вычислить $d = (|a - b|, n)$. Если $d = 1$, переход к п. 2, иначе выход, ответ: d.

Пример. Пусть $n = 55$.

a	b	d
2	2	
5	26	$(21, 55) = 1$
26	15	$(11, 55) = 11$

Ответ: 11.

Замечания:

1. В шаге 3 вместо отказа можно вернуться в п. 2 и сменить функцию (например, на $f(x) = (x^2 + c) \bmod n$ для некоторого c).

2. Алгоритм имеет сложность порядка $O(\sqrt{d}) = O(n^{1/4})$ операций.

3. Вероятность успеха алгоритма тем выше, чем меньше d. Поэтому алгоритм относится к классу методов специального назначения – хорошо ищет маленькие делители.

3.6. ($p-1$)-Метод Полларда

Этот метод также относится к классу методов спецназначения и является вероятностным типа Las-Vegas.

Определение. Пусть B – целое. Число n называется *B-гладким*, если все его простые делители не превосходят B.

($p - 1$)-Метод Полларда позволяет, исходя из некоторой границы гладкости B, найти делитель p числа n такой, что $p - 1$ – B-гладкое.

Пусть $\{q_1, \ldots, q_k\}$ – множество всех простых, не превосходящих B. Для каждого q_i, $i = 1, \ldots, k$, найдём наибольшее значение e_i такое, что $q_i^{e_i} \le n$, т. е. $e_i = [\log n / \log q_i]$. Вычислим $Q = \prod_{i=1}^{k} q_i^{e_i}$; заметим, что если p – простой делитель n и $(p - 1)$ – B-гладкое, то $p - 1 | Q$. Выберем произ-

вольное a, взаимно простое с p. Тогда по теореме Ферма $a^{p-1} = 1 \pmod p$, откуда $a^Q = 1 \pmod p$, т. е. $p|a^Q - 1$ и, значит, $p|(a^Q - 1, n)$. Если к тому же выполнено неравенство $(a^Q - 1, n) \neq n$, то имеем нетривиальный делитель n, иначе алгоритм заканчивает работу безрезультатно.

Алгоритм поиска нетривиального делителя числа

Вход: составное n.

Выход: d – нетривиальный делитель n, или отказ.

1. Выбрать границу гладкости B.
2. Выбрать случайное $2 \le a \le n - 2$, вычислить $d = (a, n)$. Если $d > 1$, выход, ответ: d.
3. Для всех простых $q \le B$

   3.1. Вычислить $e = [\log n / \log q]$.

   3.2. Положить $a = a^{q^e} \bmod n$.
4. Если $a = 1$, отказ, иначе найти $d = (a - 1, n)$.
5. Если $d = 1$, отказ, иначе ответ: d.

Замечания:

1. Отказы алгоритма в п. 4 и в п. 5 имеют разную природу. Равенство $d = 1$ в п. 5 означает, что число n не имеет простых делителей p таких, что $p - 1$ является B-гладким, и для успеха алгоритма необходимо увеличивать границу гладкости. В случае же равенства $a = 1$ в п. 4 отказ не катастрофический; можно вернуться в п. 2 и попробовать выбрать другое значение a.

2. Сложность алгоритма $O(B \cdot \log n / \log B)$ операций умножения.

3. На практике граница гладкости B выбирается из требований быстродействия; в литературе рекомендуется значение $B \approx 10^5 - 10^6$.

4. Для успешного противостояния этому методу все простые делители p факторизуемого числа должны быть таковы, чтобы $p - 1$ не являлось B-гладким. Отсюда возникло требование 1 в понятии сильного простого числа (см. раздел 2.8).

Пример. Пусть $n = 77$.

1. $B = 2$.
2. $a = 2$.
3. Для $q = 2$ вычисляем $e = [\log_2 77] = 6$, $a = 2^{64} \bmod 77 = 16$.
4. $d = (15, 77) = 1$ – отказ; мала граница гладкости B (и в самом деле, оба простых делителя числа 77 (7 и 11) таковы, что соответственно 6 и 10 не являются 2-гладкими).

Увеличим границу гладкости до 3.

3. Для $q = 3$ получим $e = [\log_3 77] = 3$, $16^{27} \bmod 77 = 36$.
4. $d = (35, 77) = 7$ – ответ.

3.7. Методы случайных квадратов

Задача: найти нетривиальный делитель числа n. Общая идея методов этого класса состоит в отыскании x, y таких, что

$$x^2 = y^2 \pmod n, \quad x \neq \pm y \pmod n.$$

Тогда

$$(x - y)(x + y) = 0 \pmod n,$$
$$(x - y) \neq 0 \pmod n,$$
$$(x + y) \neq 0 \pmod n.$$

В этом случае $d_1 = (x - y, n)$ и $d_2 = (x + y, n)$ – нетривиальные делители n, так как если p – простой делитель n, то $p|(x - y)(x + y)$ и, по основному свойству простых, $p|(x - y)$ или $p|(x + y)$. Поэтому $d_1 \neq 1$, $d_2 \neq 1$. С другой стороны, $d_1 \neq n$ и $d_2 \neq n$ в силу условий $n \nmid x - y$, $n \nmid x + y$.

Утверждение. Если n – нечётное составное не являющееся степенью число, x и y – случайные числа, такие, что $x^2 = y^2 \pmod n$, то вероятность того, что $x \neq \pm y \pmod n$, не меньше ½.

Доказательство.

Пусть каноническое разложение числа n есть $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$, y – случайное. Тогда сравнение $x^2 = y^2 \pmod n$, рассматриваемое относительно x, имеет 2^k решений, из них два тривиальных: $x = y$, $x = -y$. Таким образом, вероятность того, что $x \neq \pm y \pmod n$, составляет $\dfrac{2^k - 2}{2^k}$, что больше или равно ½ при $k \geq 2$.

3.7.1. Метод Диксона

Итак, наша цель – найти такие x и y, что $x^2 = y^2 \pmod n$. Выберем некоторое случайное a_1, вычислим $b_1 = a_1^2 \bmod n$ и найдём разложение b_1 на простые множители: $b_1 = \prod_{j=1}^{k} p_j^{e_j}$. Если все степени e_j чётные, то положим $x = a_1$, $y = \prod_{j=1}^{k} p_j^{e_j/2}$. В противном случае выберем другое случайное a_2, вычислим $b_2 = a_2^2 \bmod n$ и найдём разложение $b_2 = \prod_{j=1}^{k} p_j^{s_j}$. Если получится, что все суммы $(e_j + s_j)$ чётные, возьмём $x = a_1 \cdot a_2$, $y = \prod_{j=1}^{k} p_j^{(e_j + s_j)/2}$.

В общем случае поиск подходящих x, y сводится к решению системы сравнений по модулю 2. И, конечно же, при поиске разложения b_i не решается задача факторизации; разложение ищется в некоторой фиксированной факторной базе.

Алгоритм Диксона поиска нетривиального делителя числа

Вход: нечётное составное n, не являющееся степенью.

Выход: d – нетривиальный делитель n.

1. Выбрать факторную базу $S = \{p_1, \ldots, p_k\}$ – первые k простых чисел. Положить $t = k + 1$.

2. Построить t пар чисел (выбирая a_i случайно) a_i, b_i таких, что $b_i = a_i^2 \bmod n$ и b_i является p_k-гладким, т. е. существует разложение $b_i = \prod_{j=1}^{k} p_j^{e_{ij}}$.

3. Обозначим через V матрицу размера $t \times k$ с элементами $v_{ij} = e_{ij} \bmod 2$, через c – вектор булевых переменных длины t, и составим систему линейных однородных уравнений

$$c \cdot V = 0. \qquad (1)$$

Система содержит k уравнений и $t > k$ неизвестных, поэтому решение всегда существует.

4. Пусть $c = (c_1 \ldots c_t)$ – решение системы (1). Обозначим $I = \{i: c_i = 1\}$ – множество индексов, соответствующих единичным компонентам вектора c, и вычислим $b = \prod_{i \in I} b_i$. Получим $b = \prod_{i \in I} \prod_{j=1}^{k} p_j^{e_{ij}} = \prod_{j=1}^{k} p_j^{\sum_{i \in I} e_{ij}} = \prod_{j=1}^{k} p_j^{\sum_{i=1}^{t} c_i e_{ij}}$ – полный квадрат, поскольку $\sum_{i=1}^{t} c_i \cdot e_{ij} = \sum_{i=1}^{t} c_i \cdot v_{ij} \pmod 2 = 0$.

5. Положим $x = \prod_{i \in I} a_i \bmod n$, $y = \sqrt{b}$. По построению $x^2 = y^2 \pmod n$.

6. Если $x \neq \pm y \pmod n$, то ответ: $d = (x + y, n)$. Иначе возвращаемся в п. 4 и выбираем другое решение системы (1). Если все решения перебрали и не получили результата, то увеличиваем t на 1, строим ещё одну пару чисел a_t, b_t, удовлетворяющую требованиям п. 2, и возвращаемся в п. 3.

Пример. Пусть $n = 143$.

1. Выберем $S = \{2, 3, 5\}$. Тогда $t = 4$.
2. $17^2 = 3 \pmod{143}$, $3 = 2^0 \cdot 3^1 \cdot 5^0$;
   $19^2 = 75 \pmod{143}$, $75 = 2^0 \cdot 3^1 \cdot 5^2$;
   $21^2 = 12 \pmod{143}$, $12 = 2^2 \cdot 3^1 \cdot 5^0$;

$23^2 = 100 \pmod{143}$, $100 = 2^2 \cdot 3^0 \cdot 5^2$.

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

4. Выберем решение $c = [0\ 1\ 1\ 0]$; таким образом, $I = \{2, 3\}$, $b = 2^2 \cdot 3^2 \cdot 5^2$.
5. $x = 19 \cdot 21 \bmod 143 = 113$, $y = 2 \cdot 3 \cdot 5 = 30$.
6. $x = -y$; возврат к п. 4.
4. Выберем решение $c = [1\ 1\ 0\ 0]$; получим: $I = \{1, 2\}$, $b = 2^0 \cdot 3^2 \cdot 5^2$.
5. $x = 17 \cdot 19 \bmod 143 = 37$, $y = 3 \cdot 5 = 15$.
6. $d = (52, 143) = 13$ – ответ.

3.7.2. Метод квадратичного решета

Этот метод отличается от метода Диксона только в шагах 1, 2, на которых более направленно выбираются факторная база S и пары a_i, b_i так, что число b_i с большей вероятностью разложимо в базе S.

Положим $m = [\sqrt{n}\,]$ и рассмотрим многочлен $g(x) = (x + m)^2 - n$. Заметим, что $|g(x)|$ тем меньше, чем меньше $|x|$. Будем в шаге 2 выбирать $a_i = x + m$ для малых (по абсолютной величине) значений x и полагать $b_i = g(x)$ в расчёте на разложимость (с большой вероятностью) последнего в базе S в силу его малости. Для разложения отрицательных величин включим в S значение -1.

Кроме того, заметим следующее. По построению $b_i = (x + m)^2 - n$ и, если p простое и $p | b_i$, то $(x + m)^2 = n \pmod p$, т. е. n является квадратичным вычетом по модулю p. Значит, в факторную базу S имеет смысл включать только такие простые p, для которых $\left(\frac{n}{p}\right) = 1$.

Алгоритм квадратичного решета поиска нетривиального делителя числа

Вход: нечётное составное n, не являющееся степенью.

Выход: d – нетривиальный делитель n.

1. Выбрать факторную базу $S = \{p_1, \ldots, p_k\}$, где $p_1 = -1$, $p_2 = 2$, p_j для $j = 3, \ldots, k$ – первые простые, начиная с 3, такие, что $\left(\frac{n}{p_j}\right) = 1$. Положить $t = k + 1$. Вычислить $m = [\sqrt{n}\,]$.
2. Построить t пар чисел a_i, b_i, полагая $a_i = x + m$ для $x = 0, \pm1, \pm2, \ldots$ и $b_i = (x + m)^2 - n$, таких, что существует разложение $b_i = \prod_{j=1}^{k} p_j^{e_{ij}}$.

Шаги 3 – 6 те же, что в алгоритме Диксона.

Пример. Пусть $n = 91$.

1. Возьмём $k = 4$. Вычислим $\left(\frac{91}{3}\right) = \left(\frac{1}{3}\right) = 1$; $\left(\frac{91}{5}\right) = \left(\frac{1}{5}\right) = 1$. Следовательно, $S = \{-1, 2, 3, 5\}$. Вычислим $m = [\sqrt{91}\,] = 9$.
2.

x	$a = x + m$	$b = a^2 - n$	e_i
0	9	−10	1 1 0 1
1	10	9	0 0 2 0
−1	8	−27	1 0 3 0
2	11	30	0 1 1 1
−2	7	−42	–
3	12	53	–
−3	6	−55	–
4	13	78	–
−4	5	−66	–
5	14	105	–
−5	4	−75	1 0 1 2

3. $V = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$.
4. Выберем решение $c = [1\ \ 0\ \ 1\ \ 1\ \ 0]$; таким образом, $I = \{1, 3, 4\}$, $b = (-1)^2 \cdot 2^2 \cdot 3^4 \cdot 5^2$.
1. $x = 9 \cdot 8 \cdot 11 \bmod 91 = 64$, $y = (-1) \cdot 2 \cdot 3^2 \cdot 5 = -90$.
2. $d = (-26, 91) = 13$ – ответ.

3.7.3. Метод цепных дробей

Этот алгоритм был опубликован Дж. Бриллхартом и М.А. Моррисоном в 1975 г. Цель, как и прежде, повысить вероятность разложимости значений $b_i = a_i^2 \bmod n$ в факторной базе S за счёт их «малости». В методе цепных дробей в качестве a_i выбираются числители подходящих дробей, получающихся при разложении значения $\sqrt{n}$ в цепную дробь. Докажем следующую теорему.

Теорема. Пусть $x > 1$ – вещественное число, $\left\{\frac{P_i}{Q_i} : i = 1,2,\ldots\right\}$ – последовательность его подходящих дробей. Тогда $|P_i^2 - x^2 Q_i^2| < 2x$ для любого i.

Доказательство. По свойствам подходящих дробей, число x лежит между соседними подходящими дробями $\frac{P_i}{Q_i}$ и $\frac{P_{i+1}}{Q_{i+1}}$, и $\left|\frac{P_i}{Q_i} - \frac{P_{i+1}}{Q_{i+1}}\right| = \frac{1}{Q_i Q_{i+1}}$. Тогда можем записать:

$$|P_i^2 - x^2 Q_i^2| = Q_i^2 \cdot \left|\frac{P_i}{Q_i} - x\right| \cdot \left|\frac{P_i}{Q_i} + x\right| < Q_i^2 \cdot \frac{1}{Q_i Q_{i+1}} \cdot \left(2x + \frac{1}{Q_i Q_{i+1}}\right) =$$

$$= 2x \cdot \left(\frac{Q_i}{Q_{i+1}} + \frac{1}{2x \cdot Q_{i+1}^2}\right) < 2x \cdot \left(\frac{Q_i}{Q_{i+1}} + \frac{1}{Q_{i+1}}\right) = 2x \cdot \frac{Q_i + 1}{Q_{i+1}} \le 2x;$$

здесь последнее неравенство имеет место ввиду $Q_i < Q_{i+1}$. Теорема доказана.

Следствие. Пусть $n > 16$ и $\frac{P_i}{Q_i}$ – подходящая дробь для $\sqrt{n}$. Тогда минимальный по абсолютной величине вычет $P_i^2 \bmod n$ равен $P_i^2 - nQ_i^2$ и не превосходит $2\sqrt{n}$.

Доказательство. Очевидно, что $P_i^2 = P_i^2 - nQ_i^2 \pmod n$. Положив в теореме $x = \sqrt{n}$, получим $|P_i^2 - nQ_i^2| < 2\sqrt{n} < \frac{n}{2}$, что означает, что $P_i^2 - nQ_i^2$ – минимальный по абсолютной величине вычет $P_i^2 \bmod n$.

Алгоритм поиска нетривиального делителя числа

Вход: нечётное составное n, не являющееся степенью.

Выход: d – нетривиальный делитель n.

1. Выбрать факторную базу $S = \{p_1, \ldots, p_k\}$, где $p_1 = -1$, $p_2 = 2$, p_j для $j = 3, \ldots, k$ – первые простые, начиная с 3, такие, что $\left(\frac{n}{p_j}\right) = 1$. Положить $t = k + 1$.
2. Построить t пар чисел a_i, b_i, полагая $a_i = P_j \bmod n$ и $b_i = P_j^2 - nQ_j^2$, где $\frac{P_j}{Q_j}$ – подходящие дроби для числа $\sqrt{n}$, таких, что существует разложение $b_i = \prod_{j=1}^{k} p_j^{e_{ij}}$.

Шаги 3 – 6 те же, что в алгоритме Диксона.

Пример тот же, что в предыдущем разделе: $n = 91$.

1. Возьмём $k = 4$, $S = \{-1, 2, 3, 5\}$.
2. Подходящие дроби для $\sqrt{91}$: $\frac{9}{1}, \frac{10}{1}, \frac{19}{2}, \frac{105}{11}, \frac{124}{13}, \frac{725}{76}, \ldots$.

Строим таблицу:

$a = P_j \bmod n$	$b = P_j^2 - nQ_j^2$	e_j
9	−10	1 1 0 1
10	9	0 0 2 0
19	−3	1 0 1 0
14	14	–
33	−3	1 0 1 0
88	9	0 0 2 0

3. $V = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$.
4. Выберем решение $c = [0\ 0\ 1\ 1\ 0]$; таким образом, $I = \{3, 4\}$, $b = (-1)^2 \cdot 3^2$.
5. $x = 19 \cdot 33 \bmod 91 = 81$, $y = (-1) \cdot 3 = -3$.
6. $d = (78, 91) = 13$ – ответ.

4. МЕТОДЫ ДИСКРЕТНОГО ЛОГАРИФМИРОВАНИЯ

Введение

Задача дискретного логарифмирования – ещё одна (наряду с факторизацией) задача, на сложности которой основана стойкость ряда криптосистем, самые известные из которых – схема распределения ключей Диффи–Хеллмана и криптосистема Эль-Гамаля. Состояние дел относительно её сложности такое же, как и для задачи факторизации – на сегодня не найдено полиномиальных алгоритмов её решения, но и не доказано, что их не существует.

Пусть G – конечная *циклическая группа* с порождающим элементом g ($G = \langle g \rangle$), $|G| = n$ и a – произвольный элемент из G. Тогда существует единственное целое значение x, $0 \le x < n$, такое, что $a = g^x$. Это значение x называется *дискретным логарифмом a по основанию g*, обозначается $x = \log_g a$.

Сложность задачи дискретного логарифмирования

Почему дискретное логарифмирование, в отличие от логарифмирования в бесконечной группе, является сложной задачей? Ведь найти «обычный» логарифм можно с помощью простейшего калькулятора, стандартной функции любого языка программирования, логарифмической линейки, в конце концов. Что принципиально меняется, когда мы переходим в дискретное множество? Дело в том, что при этом функция возведения в степень перестаёт быть монотонной, и не работает, скажем, такой приём, как метод двоичного поиска. В силу дискретности множества неприменимы разложения в ряды и другие методы непрерывной математики.

Второй вопрос. Почему функция возведения в степень остаётся простой, даже при возведении в степень по модулю, а обратная к ней функция – поиска логарифма – сложна? Казалось бы, можно, начав с 1, последовательно умножать результат на элемент g и сравнивать его с требуемым элементом a – по сравнению с возведением в степень добавляется только элементарная операция сравнения на каждом шаге. Однако такое «возведение в степень» имеет экспоненциальную сложность; особенность же быстрого алгоритма возведения в степень (см. раздел 1.2) в том, что в нём вычисляются не все возможные степени основания, а только некоторые из них.

4.1. Алгоритм Гельфонда

Алгоритм предложен советским математиком А.О. Гельфондом в 1962 г.; в англоязычной литературе известен под названием «baby-step giant-step algorithm» («шаги ребёнка – шаги великана»), отражающим технику работы этого алгоритма.

Алгоритм Гельфонда

Вход: $G = \langle g \rangle$ – циклическая группа, $|G| = n$, $a \in G$.

Выход: $x = \log_g a$.

1. Вычислить $h = [\sqrt{n}\,] + 1$.
2. Вычислить $b = g^h$.
3. Построить таблицы
   $\{b^u: u = 1, \ldots, h\}$ («шаги великана») и
   $\{a \cdot g^v: v = 1, \ldots, h\}$ («шаги ребёнка»).
4. Найти в таблицах одинаковые элементы
   $b^u = g^{hu} = a \cdot g^v$,
   и вычислить отсюда $x = hu - v \bmod n$.
5. Ответ: x.

Обоснование алгоритма

Нужно доказать, что одинаковые элементы в таблицах обязательно найдутся, т.е. найдутся такие u, v, что $g^{hu} = a \cdot g^v = g^{x+v}$. Это равенство имеет место в случае $hu = x + v \pmod n$, или $x = hu - v \pmod n$. При $1 \le u \le h$, $1 \le v \le h$ выполнено неравенство $0 \le hu - v \le h^2 - 1$. Для разных пар (u_1, v_1) и (u_2, v_2) имеем $hu_1 - v_1 \ne hu_2 - v_2$, так как в противном случае получилось бы $h(u_1 - u_2) = v_1 - v_2$ и $h \mid v_1 - v_2$, что при указанном выборе v_1, v_2 возможно только в случае $v_1 = v_2$, и, следовательно, $u_1 = u_2$. Таким образом, выражения $hu - v$ принимают все значения в диапазоне от 0 до $h^2 - 1$, который, в силу того, что $h^2 > n$, содержит все возможные значения x – от 0 до $n - 1$. Значит, для некоторых u и v имеет место равенство $x = hu - v \pmod n$.

Оценка сложности алгоритма

Шаг 2 выполняется за $\log h$ умножений.

В шаге 3 построение таблиц потребует $2h$ умножений.

И, как это ни покажется странным, самый трудоёмкий этап алгоритма – это шаг 4, поиск одинаковых элементов в таблицах. Рассмотрим различные способы решения этой задачи и оценим сложность каждого их них.

1. Поиск с помощью сравнения «каждого с каждым» потребует h^2 сравнений. Поскольку $h^2 > n$ – количества возможных значений логарифма, то этот способ не лучше полного перебора.

2. Совместная сортировка двух таблиц по возрастанию (или убыванию) значений b^u и $a \cdot g^v$ поставит одинаковые значения рядом. Быстрые алгоритмы сортировки (Шелла, пирамиды, Хоара) имеют сложность $O(n \log n)$ операций, где n – длина массива. Таким образом, примерно за $2h \log(2h)$ операций можно отсортировать две таблицы, и еще порядка $2h$ сравнений понадобится для поиска рядом стоящих одинаковых элементов.

3. Можно отсортировать только одну таблицу, выполнив для этого $O(h \log h)$ операций, а затем элементы второй таблицы искать в упорядоченной первой методом двоичного поиска. Сложность двоичного поиска элемента в массиве длины h равна $O(\log h)$, и искать (в худшем случае) придётся h элементов. Итого сложность всего процесса составляет около $2h \log h$ операций. Преимущество этого метода ещё и в том, что нет необходимости хранить обе таблицы; элементы второй из них можно вычислять поочерёдно.

Пример. Пусть $G = Z_{23}^* = \langle 5 \rangle$, $a = 3$.

1. $h = [\sqrt{22}\,] + 1 = 5$.
2. $b = 5^5 \bmod 23 = 20$.
3.

u, v	1	2	3	4	5
20^u	20	9	19	12	10
$3 \cdot 5^v$	15	6	7	12	14

1. $x = 5 \cdot 4 - 4 = 16$.
2. Ответ: 16.

4.2. ρ-Метод Полларда

Также как и одноимённый алгоритм факторизации, ρ-метод Полларда дискретного логарифмирования основан на методе Флойда определения длины периода периодической последовательности (см. раздел 3.5.1). Метод является вероятностным типа Las-Vegas (т. е. возможен безрезультатный выход из алгоритма) и имеет ту же сложность по времени, что и предыдущий метод, но существенно экономичнее по объёму используемой памяти.

Пусть $G = \langle g \rangle$ – циклическая группа, $|G| = n$, $a \in G$. Разобьём группу G на три примерно равных по мощности подмножества S_1, S_2, S_3 так, чтобы $1 \notin S_2$. Зададим $x_0 = 1$ и построим последовательность $x_0, x_1, x_2, \ldots$ по правилу $x_{i+1} = f(x_i)$, $i = 0, 1, \ldots$, где псевдослучайная функция $f: G \to G$ определяется формулами:

$$f(x)=\begin{cases} a\cdot x, \text{если } x\in S_1,\\ x^2, \text{если } x\in S_2,\\ g\cdot x, \text{если } x\in S_3. \end{cases} \quad (1)$$

Видно, что $x_i = g^{y_i}\cdot a^{b_i}$ для $i = 0, 1, \ldots$, где $y_0 = b_0 = 0$ и, в соответствии с формулами (1), значения y_i, b_i вычисляются по формулам:

$$y_{i+1}=\begin{cases} y_i, \text{если } x_i\in S_1,\\ 2y_i \bmod n, \text{если } x_i\in S_2,\\ (y_i+1)\bmod n, \text{если } x_i\in S_3, \end{cases} \quad (2)$$

$$b_{i+1}=\begin{cases} (b_i+1)\bmod n, \text{если } x_i\in S_1,\\ 2b_i \bmod n, \text{если } x_i\in S_2,\\ b_i, \text{если } x_i\in S_3. \end{cases} \quad (3)$$

Введём функцию $F: G\times Z_n\times Z_n \to G\times Z_n\times Z_n$, которая вычисляет $(x_{i+1}, y_{i+1}, b_{i+1}) = F(x_i, y_i, b_i)$ по формулам (1), (2), (3). Методом Флойда найдём совпадающие значения $x_i = x_{2i}$, откуда

$$g^{y_i}\cdot a^{b_i} = g^{y_{2i}}\cdot a^{b_{2i}},$$

$$a^{b_i-b_{2i}} = g^{y_{2i}-y_i},$$

$$(b_i - b_{2i})\cdot \log_g a = y_{2i} - y_i \pmod n.$$

Решаем последнее линейное сравнение относительно $\log_g a$. Оно имеет $d = (b_i - b_{2i}, n)$ решений $z_0, \ldots, z_{d-1}$, нужное из которых определим перебором, возводя g в степени z_i. Понятно, что в случае $d = n$, т. е. $b_i = b_{2i}$, такой метод ничем не лучше полного перебора, и алгоритм выдаёт отказ.

Алгоритм дискретного логарифмирования (ρ-метод Полларда)

Вход: $G = \langle g\rangle$ – циклическая группа, $|G| = n$, $a\in G$.

Выход: $\log_g a$, или отказ.

1. Положить $x_1 = x_2 = 1\in G$, $y_1 = y_2 = b_1 = b_2 = 0\in Z$.
2. Положить $(x_1, y_1, b_1) = F(x_1, y_1, b_1)$ и $(x_2, y_2, b_2) = F(F(x_2, y_2, b_2))$.
3. Если $x_1 \neq x_2$, переход к п. 2.
4. Вычислить $r = (b_1 - b_2) \bmod n$. Если $r = 0$, выдать отказ и закончить работу.
5. Найти $d = (r, n)$ решений $z_0, \ldots, z_{d-1}$ сравнения $rx = a_2 - a_1 \pmod n$.
6. Найти значение $i\in\{0, 1, \ldots, d-1\}$, для которого выполняется условие $g^{z_i} = a$.
7. Ответ: $\log_g a = z_i$.

Замечания по алгоритму:

1. Алгоритм лучше всего применим для логарифмирования в группах простого порядка, так как в этом случае вероятность отказа мала, а в случае неотказа $d = 1$, и решение находится без перебора в шаге 6.

2. Отказов алгоритма можно совсем избежать, если в шаге 4 в случае $r = 0$ выполнить следующие действия: сгенерировать случайные $y_1, b_1 \in Z_n$; вычислить $x_1 = g^{y_1} \cdot a^{b_1}$; положить $x_2 = x_1$, $y_2 = y_1$, $b_2 = b_1$ и перейти к п. 2.

Пример. Пусть $G = Z_{29}^* = \langle 2 \rangle$, $a = 5$.

x_1	y_1	b_1	x_2	y_2	b_2
1	0	0	1	0	0
5	0	1	25	0	2
25	0	2	18	1	3
9	0	3	6	2	4
18	1	3	24	4	4
7	2	3	8	5	5
6	2	4	12	11	10
12	3	4	19	13	10
24	4	4	6	26	22
19	5	4	24	0	22
8	5	5	8	1	23

Получили сравнение: $2^5 \cdot 5^5 = 2^1 \cdot 5^{23} \pmod{29}$.

Вычисляем:

$r = (5 - 23) \bmod 28 = 10$;

$10x = 1 - 5 = 24 \pmod{28}$;

$d = (10, 28) = 2$;

$5x = 12 \pmod{14}$;

$5^{-1} \bmod 14 = 3$;

$x = 3 \cdot 12 = 8 \pmod{14}$.

Таким образом, решениями сравнения являются $z_0 = 8$, $z_1 = 22$. Проверяем: $2^8 \bmod 29 = 24 \neq 5$.

Ответ: $\log_2 5 = 22$.

4.3. Алгоритм Адлемана

Этот алгоритм называют также «методом вычисления индексов» («index-calculus algorithm»). Основная идея алгоритма состоит в том, что в группе G выбирается некоторое подмножество элементов, называемое факторной базой, и сначала определяются логарифмы элементов этого подмножества. Затем с их помощью вычисляется искомый логарифм.

Алгоритм

Вход: $G = \langle g \rangle$ – циклическая группа, $|G| = n$, $a \in G$.

Выход: $x = \log_g a$.

1. Выбираем факторную базу $S = \{p_1, p_2, \ldots, p_t\} \subseteq G$.
2. Ищем значения g^k, разложимые в базе S; для этого:
   2.1. Выбираем случайное k, $0 < k < n$, и вычисляем $b = g^k$.
   2.2. Ищем разложение $b = \prod_{i=1}^{t} p_i^{e_i}$; в случае успеха составляем линейное сравнение $\sum_{i=1}^{t} e_i \cdot \log_g p_i = k (\mathrm{mod}\, n)$.
   2.3. Шаги 2.1 и 2.2 повторяем до тех пор, пока не получим систему сравнений, имеющую единственное решение.
3. Решая полученную на шаге 2 систему сравнений, находим значения $\log_g p_i$ для всех $i = 1, \ldots, t$.
4. Ищем $x = \log_g a$; для этого:
   4.1. Выбираем случайное k, $0 \le k < n$, и вычисляем $b = a \cdot g^k$.
   4.2. Ищем разложение $b = \prod_{i=1}^{t} p_i^{e_i}$; в случае успеха находим $x = \sum_{i=1}^{t} e_i \cdot \log_g p_i - k (\mathrm{mod}\, n)$, иначе возвращаемся к п. 4.1.
5. Ответ: x.

Замечания по алгоритму:

1. Алгоритм лучше всего применим в группах простого порядка, так как в этом случае в шаге 3 приходится решать систему линейных уравнений над полем, а не над кольцом, как в случае составного n, что много проще. Об особенностях работы алгоритма Адлемана в этом случае см. раздел 4.5.

2. При логарифмировании в группе Z_p^* в качестве факторной базы S рекомендуется выбирать t первых простых чисел.

3. При определении мощности факторной базы приходится искать компромисс между легкостью разложимости элементов группы в этой базе и размером получаемой системы: при слишком малом t трудно найти элементы, эффективно разложимые в S, но с ростом t растёт и количество сравнений, которые необходимо построить. Экспериментально установлено, что для $n \approx 2^{32}$ оптимальным значением является $t \approx 30$. Подробно результаты исследования алгоритма при различных способах составления и решения системы сравнений представлены в [2].

4. Шаги 1–3 алгоритма не зависят от a и могут быть выполнены один раз при поиске логарифмов нескольких элементов.

Пример. Пусть $G = Z_{37}^{*} = <2>$, $a = 13$.

1. Положим $S = \{2, 3, 5\}$.
2. $k = 1$, $2^1 = 2$;
   $k = 13$, $2^{13} = 15 = 3 \cdot 5$;
   $k = 25$, $2^{25} = 20 = 2^2 \cdot 5$.
   Получили систему сравнений:
$$\begin{cases} \log 2 = 1 (\mathrm{mod}\, 36), \\ \log 3 + \log 5 = 13 (\mathrm{mod}\, 36), \\ 2\log 2 + \log 5 = 25 (\mathrm{mod}\, 36). \end{cases}$$
3. Решением системы является:
   $\log 2 = 1$,
   $\log 3 = 26$,
   $\log 5 = 23$.
4. $k = 2$, $13 \cdot 2^2 = 15 = 3 \cdot 5$, $x = \log 3 + \log 5 - 2 \pmod{36} = 11$.
5. Ответ: 11.

4.4. Алгоритм Полига–Хеллмана

Алгоритм позволяет быстро вычислять логарифмы в случае, если известно разложение порядка группы на простые множители и все эти множители невелики. Перейдём сразу к изложению алгоритма; необходимые пояснения будем давать по ходу этого изложения.

Алгоритм

Вход: $G = <g>$ – циклическая группа, $|G| = n = \prod_{i=1}^{k} p_i^{e_i}$, $a \in G$.

Выход: $x = \log_g a$.

1. Предварительный этап (построение таблиц)
   Для $i = 1, \ldots, k$ выполнять:

   1.1. Вычислить $\alpha_i = g^{\frac{n}{p_i}}$.

   1.2. Для $j = 0, 1, \ldots, p_i - 1$ вычислить и запомнить $r_{ij} = \alpha_i^j$.

   (Заметим, что r_{ij} суть все различные корни степени p_i из 1 в G).
2. Для $i = 1, \ldots, k$ вычислить значения

   $x \bmod p_i^{e_i} = x_0 + x_1 \cdot p_i + x_2 \cdot p_i^2 + \ldots + x_{e_i - 1} \cdot p_i^{e_i - 1}$. Для этого:

2.1. Вычислить $b = a^{\frac{n}{p_i}}$. Поскольку $a = g^x$ и $x = x_0 + t \cdot p_i$ для некоторого t, то можем записать:

$$b = (g^x)^{\frac{n}{p_i}} = (g^{x_0})^{\frac{n}{p_i}} \cdot g^{t \cdot n} = \left(g^{\frac{n}{p_i}} \right)^{x_0} = \alpha_i^{x_0} = r_{ix_0}.$$

Таким образом, значение x_0 равно номеру элемента, равного b, в i-ой таблице.

2.2. Для $j = 1, \ldots, e_i - 1$ определим значение x_j аналогично. А именно: пусть $x_0, \ldots, x_{j-1}$ уже найдены. Обозначим $y = x_0 + x_1 \cdot p_i + \ldots + x_{j-1} \cdot p_i^{j-1}$ и представим x в виде $x = y + x_j \cdot p_i^j + t \cdot p_i^{j+1}$. Вычислим

$$b = (a \cdot g^{-y})^{\frac{n}{p_i^{j+1}}} = (g^{x_j p_i^j} \cdot g^{t p_i^{j+1}})^{\frac{n}{p_i^{j+1}}} = (g^{x_j})^{\frac{n}{p_i}} \cdot g^{t \cdot n} = \left(g^{\frac{n}{p_i}} \right)^{x_j} = \alpha_i^{x_j} = r_{ix_j}.$$ Заметим, что вновь значение x_j равно номеру элемента b в i-ой таблице.

3. Имея значения $x \bmod p_i^{e_i}$ для всех $i = 1, \ldots, k$, по *Китайской теореме об остатках* найти ответ: значение $x \bmod n$.

Замечания по алгоритму:

1. В случае простого n работа алгоритма сводится к полному перебору всех возможных значений логарифма.

2. Как и в предыдущем алгоритме, при поиске логарифмов нескольких элементов можно шаг 1 выполнить один раз, так как он не зависит от значения логарифмируемого элемента.

Пример. Пусть $G = Z_{101}^{*} = \langle 2 \rangle$, $a = 3$.

Порядок группы $n = 100 = 2^2 \cdot 5^2$.

1. $\alpha_1 = 2^{50} = \left(\frac{2}{101} \right) = -1 \bmod 101 = 100,$

$r_{10} = 100^0 = 1, r_{11} = 100^1 = 100.$

$\alpha_2 = 2^{20} = 95, r_{20} = 95^0 = 1, r_{21} = 95^1 = 95, r_{22} = 95^2 = 36,$

$r_{23} = 95^3 = 87, r_{24} = 95^4 = 84.$

2. а) Ищем $x \bmod 4 = x_0 + 2x_1$.

2.1. $b = 3^{50} = \left(\frac{3}{101} \right) = -1 \bmod 101 = 100 \Rightarrow x_0 = 1.$

2.2. $b = (3 \cdot 2^{-1})^{25} = (3 \cdot 51)^{25} = 1 \Rightarrow x_1 = 0.$

Итак, $x = 1 \pmod 4$.

б) Ищем $x \bmod 25 = x_0 + 5x_1$.

2.1. $b = 3^{20} = 84 \Rightarrow x_0 = 4$.

2.2. $b = (3 \cdot 2^{-4})^4 = 87 \Rightarrow x_1 = 3$.

Итак, $x = 19 \pmod{25}$.

1. Решаем систему сравнений

$$\begin{cases} x = 1 \pmod 4, \\ x = 19 \pmod{25}. \end{cases}$$

Ответ: $x = 69$.

4.5. Логарифмирование в подгруппах простого порядка

Пусть p – простое число, q – простой делитель $p - 1$, G – циклическая подгруппа порядка q группы Z_p^*, и a – порождающий элемент G. Тогда задача дискретного логарифмирования в G заключается в следующем: для данных p, q, a и $b \in G$ требуется найти единственное целое x, $0 \le x \le q - 1$, такое, что $a^x = b \pmod p$.

На сложности этой задачи основана стойкость таких криптосистем, как протоколы идентификации Окамото, Шнорра, цифровой подписи Шаума-ван-Антверпена, американский стандарт цифровой подписи DSS и другие. Обсудим возможности применения рассмотренных выше алгоритмов для решения этой задачи.

В алгоритме Гельфонда и ρ-методе Полларда особенности группы никак не учитываются; их работоспособность зависит только от того, насколько велик её порядок. Таким образом, эти алгоритмы могут применяться для логарифмирования в группе G без каких-либо изменений и с тем же успехом, что и в других группах порядков, приблизительно равных q. Алгоритм Полига – Хеллмана, как уже отмечалось (см. замечание 1 в предыдущем разделе), совершенно неприменим для логарифмирования в группах простого порядка. Алгоритм Адлемана, казалось бы, наоборот, поведёт себя наилучшим образом именно в этом случае (см. замечание 1 раздела 4.3), однако тут возникает сложность с выбором факторной базы S. Должно выполняться условие $S \subseteq G$, т.е. в качестве S нельзя просто взять первые простые числа, как это рекомендуется при логарифмировании в Z_p^*, ведь не обязательно все они принадлежат G. «Handbook of Applied Cryptography» [20] предлагает поступать следующим образом.

Пусть g – порождающий элемент группы Z_p^* и $p = q \cdot s + 1$. С помощью алгоритма Адлемана найдём $y = \log_g a$ и $z = \log_g b$ в Z_p^*. Заметим, что в

силу равенства $a^q = g^{y \cdot q} = 1 \pmod p$ выполняется условие $p - 1 = q \cdot s \mid y \cdot q$, или $s \mid y$. Аналогично, ввиду $b \in G$, верно, что $s \mid z$. Пусть $y = s \cdot y_1$, $z = s \cdot z_1$. Запишем:

$$a^{z_1} = g^{yz_1} = g^{y_1 sz_1} = g^{y_1 z} = b^{y_1} \pmod p,$$

откуда получаем $x = \log_a b = z_1 \cdot y_1^{-1} \bmod q$; заметим, что $y_1^{-1} \bmod q$ всегда существует в силу условия $0 < y_1 < q$.

Пример. Рассмотрим подгруппу G порядка 7 группы $Z_{29}^* = <2>$; порождающим элементом G является $a = 7$. Найдём $\log_7 25$.

$y = \log_g a = \log_2 7 = 12$,
$z = \log_g b = \log_2 25 = 16$,
$y_1 = 12/4 = 3$,
$z_1 = 16/4 = 4$,
$y_1^{-1} \bmod q = 3^{-1} \bmod 7 = 5$.
Ответ: $x = \log_7 25 = 4 \cdot 5 \bmod 7 = 6$.
Проверка: $7^6 = 25 \pmod{29}$.

Другой способ решения задачи предложен А. Беловым [2] для случая $p = q \cdot s + 1$, где $(q, s) = 1$. Заметим, что подгруппа G порядка q группы Z_p^* есть множество всех вычетов степени s по модулю p, т.е. множество тех и только тех элементов, логарифмы которых по основанию g кратны s.

Выберем в качестве факторной базы множество $S = \{p_1, p_2, \ldots, p_t\}$ – не обязательно подмножество группы G, и рассмотрим разложение $a^k \bmod p = \prod_{i=1}^{t} p_i^{e_i}$. Пусть $s' = s^{-1} \bmod q$. Тогда $a^{ss'} = a \pmod p$ и $a^k = a^{kss'} = \left(\prod_{i=1}^{t} p_i^{e_i} \right)^{ss'} = \prod_{i=1}^{t} \left(p_i^{ss'} \right)^{e_i} \pmod p$. Заметим, что $p_i^s \in G$, значит, и $p_i^{ss'} \in G$. Следовательно, существует $x_i = \log_a p_i^{ss'}$. Составим сравнение $\sum_{i=1}^{t} e_i \log_a p_i^{ss'} = k \pmod q$.

Таким образом, в шаге 3 алгоритма Адлемана при решении системы сравнений будут найдены значения $x_i = \log_a p_i^{ss'}$, а не $\log_a p_i$ (которых может и не существовать).

Шаг 4 переформулируем следующим образом:
4.1. Выбираем случайное k, $0 \le k < n$, и вычисляем $c = b \cdot a^k \bmod p$.

4.2. Ищем разложение $c = \prod_{i=1}^{t} p_i^{e_i}$; в случае успеха вычисляем

$\log_a b = \sum_{i=1}^{t} e_i \cdot \log_a p_i^{ss'} - k \pmod n$, иначе возвращаемся к п. 4.1.

Для обоснования правильности получаемого результата проведём те же рассуждения: $c \in G$, откуда $c^{ss'} = c \pmod p$ и

$$b \cdot a^k = \left(\prod_{i=1}^{t} p_i^{e_i} \right)^{ss'} = \prod_{i=1}^{t} \left(p_i^{ss'} \right)^{e_i} \pmod p.$$

Пример тот же: $G = \langle 7 \rangle$ – подгруппа порядка 7 группы Z_{29}^*; найдём $\log_7 25$. Вычислим $s = (29 - 1)/7 = 4$, $s' = 4^{-1} \bmod 7 = 2$.

1. $S = \{2, 3, 5\}$.
2. $k = 2$, $7^2 = 20 = 2^2 \cdot 5$;
   $k = 3$, $7^3 = 24 = 2^3 \cdot 3$;
   $k = 5$, $7^5 = 16 = 2^4$.
   Получили систему сравнений:

$$\begin{cases} 2x_1 + x_3 = 2 \pmod 7, \\ 3x_1 + x_2 = 3 \pmod 7, \\ 4x_1 = 5 \pmod 7. \end{cases}$$

Здесь $x_1 = \log_7 (2^8) = \log_7 24$, $x_2 = \log_7 (3^8) = \log_7 7$, $x_3 = \log_7 (5^8) = \log_7 24$.

3. Решением системы является: $x_1 = 3$, $x_2 = 1$, $x_3 = 3$.
4. $k = 0$, $25 \cdot 2^0 = 25 = 5^2$, $x = 2x_3 = 6$.
5. Ответ: 6.

Обратим внимание на одну особенность этого примера. Переменные x_1 и x_3 соответствуют разным элементам факторной базы (2 и 5 соответственно), но оказалось, что они имеют одинаковый содержательный смысл – обе равны $\log_7 24$, и, следовательно, одинаковые значения. Попробуем объяснить этот факт.

Ещё раз присмотримся к элементу $p_i^{ss'}$. Составим и решим сравнение

$$x^s = p_i^s \pmod p.$$

Получаем:

$$s \cdot \log x = s \cdot \log p_i \pmod{p-1},$$
$$\log x = \log p_i \pmod q,$$
$$\log x \in \{\log p_i, \log p_i + q, \ldots, \log p_i + (s-1) \cdot q\},$$
$$x \in X = \{p_i, p_i g^q, \ldots, p_i g^{(s-1)q}\} = \{ g^{\log p_i + qt} : t = 0, \ldots, s-1 \}.$$

Множество X – это все корни s-й степени из p_i^s. Здесь t пробегает полную систему вычетов по модулю s и $(q, s) = 1$, значит, и $\log p_i + qt$ также пробегает полную систему вычетов по модулю s. Поскольку единственный из показателей $\log p_i + qt$ кратен s, то один и только один элемент из X, а именно $p_i^{ss'}$, принадлежит G. Таким образом, если в факторной базе S окажется несколько корней s-й степени из одного и того же элемента, то для каждого из них будет найдено одно и то же значение, а именно значение логарифма того корня s-й степени из этого элемента, который принадлежит G.

Возвращаясь к нашему примеру, заметим: $2^4 = 16$, корни 4-й степени из 16 – это ± 2, ± 5. Из этих четырёх корней один, а именно $-5 = 24 = 7^3 \bmod 29$, принадлежит подгруппе $G = \langle 7 \rangle$; таким образом, переменные x_1 и x_3, сопоставленные элементам 2 и 5, не случайно принимают одно и то же значение $\log_7 24$.

В случае если $s = 2^k$ (в частности, если p – надёжное простое число; тогда $s = 2$) можно расширить факторную базу S без увеличения количества переменных. А именно: для каждого $p_i \in S$ вычисляем все корни s-й степени из p_i^s по модулю p (это можно сделать, так как вычисление квадратных корней по простому модулю – лёгкая задача) и включаем их в факторную базу, сопоставляя всем им одну и ту же переменную.

Сравнивая два способа решения задачи дискретного логарифмирования в подгруппах простого порядка, отметим, что в первом случае [20] нужно дважды решать задачу логарифмирования в Z_p^*, что предполагает решение двух систем линейных уравнений над кольцом, второй же способ [2] сводится к решению одной системы линейных уравнений над полем. Что касается ограничения применимости второго способа (условия $(q, s) = 1$), то это условие автоматически выполнено, если простое p построено методом Маурера (так как в этом случае $s < q$), и выполнено с вероятностью $\frac{q-1}{q}$, если p построено по процедуре А алгоритма ГОСТ (поскольку s – случайное число в некотором диапазоне, в котором каждое q-е число кратно q).

5. АЛГОРИТМЫ НАД ПОЛИНОМАМИ

Введение

Будем рассматривать полиномы над некоторым полем F, т. е. выражения вида $f(x) = f_n \cdot x^n + f_{n-1} \cdot x^{n-1} + \ldots + f_1 \cdot x + f_0$, где коэффициенты f_i, $i = 0, \ldots, n$, принадлежат полю F, переменная x принимает значения из того же поля, $+$ и $\cdot$ – операции поля. Множество всех полиномов над полем F обозначается $F[x]$ и образует *кольцо* с операциями сложения и умножения полиномов. Наибольшее n такое, что $f_n \neq 0$, называется *степенью полинома* $f(x)$ и обозначается $\deg f$, а сам коэффициент f_n называется *старшим коэффициентом*. Полином, старший коэффициент которого равен 1, называется *нормированным*.

Задачи:

1. Подсчитайте количество различных полиномов в $Z_p[x]$:
   а) степени n;
   б) нормированных степени n;
   в) степени которых не превосходят n.
2. Что из следующего верно:
   а) если $f(x)$ и $g(x)$ нормированы, то и $f(x) + g(x)$ нормирован;
   б) если $f(x)$ и $g(x)$ нормированы, то и $f(x) \cdot g(x)$ нормирован;
   в) $\deg (f + g) = \max (\deg f, \deg g)$;
   г) $\deg (f \cdot g) = \deg f + \deg g$.

В этой главе мы рассмотрим следующие алгоритмы над полиномами, часто применяемые в криптографических приложениях: проверка полиномов на неприводимость и на примитивность, факторизация полиномов. Во многих из них приходится находить наибольший общий делитель (НОД) двух полиномов, для чего применяется хорошо уже нам знакомый алгоритм Евклида. Однако небольшая особенность возникает в определении самого понятия наибольшего общего делителя полиномов; с её рассмотрения и начнём знакомство с алгоритмами над полиномами.

5.1. Наибольший общий делитель полиномов

Говорят, что *полином* $g(x)$ *делит полином* $f(x)$, и пишут $g(x) | f(x)$, если существует полином $q(x)$ такой, что $f(x) = g(x) \cdot q(x)$. Если $g(x) | f(x)$ и $g(x) | h(x)$, то $g(x)$ – *общий делитель* $f(x)$ и $h(x)$. По аналогии с числами вводится понятие

наибольшего общего делителя полиномов, при этом сравнение полиномов производится по их степени: полином наибольшей степени среди всех общих делителей $f(x)$ и $g(x)$ логично назвать наибольшим общим делителем $f(x)$ и $g(x)$. Однако тут возникает следующая особенность. Пусть $f(x) = g(x) \cdot q(x)$ и $a \in F$ – любой ненулевой (а значит, обратимый) элемент поля. Тогда можем записать: $f(x) = a \cdot g(x) \cdot a^{-1} \cdot q(x)$, откуда следует, что полином $a \cdot g(x)$ также делит $f(x)$. Для однозначности договоримся в качестве наибольшего общего делителя рассматривать нормированный делитель.

Определение. Наибольшим общим делителем полиномов $f(x)$ и $g(x)$ называется их нормированный общий делитель наибольшей степени.

Будем обозначать наибольший общий делитель $f(x)$ и $g(x)$ через НОД($f(x)$, $g(x)$). Для нахождения НОД($f(x)$, $g(x)$) применим следующий алгоритм.

Алгоритм Евклида

Вход: $f(x), g(x) \in F[x]$.

Выход: НОД($f(x)$, $g(x)$).

1. Пока $g(x) \nmid f(x)$, выполнять:

   1.1. Поделить $f(x)$ на $g(x)$ с остатком:

   $f(x) = g(x) \cdot q(x) + r(x)$, где $\deg r < \deg g$.

   1.2. Положить $f(x) = g(x)$ и $g(x) = r(x)$.

2. Ответ: $g_n^{-1} \cdot g(x)$, где $n = \deg g$.

Пример. Найдём НОД($x^7 + x^3 + x + 1$, $x^6 + x^5 + x^4 + x^3 + x^2 + 1$) над Z_2.

$x^7 + x^3 + x + 1 = (x^6 + x^5 + x^4 + x^3 + x^2 + 1) \cdot (x + 1) + (x^3 + x^2)$;

$x^6 + x^5 + x^4 + x^3 + x^2 + 1 = (x^3 + x^2) \cdot (x^3 + x) + (x^2 + 1)$;

$x^3 + x^2 = (x^2 + 1) \cdot (x + 1) + (x + 1)$;

$x^2 + 1 = (x + 1) \cdot (x + 1)$.

Ответ: $x + 1$.

5.2. Тесты на неприводимость полиномов над Z_p

Понятие неприводимости в кольце полиномов аналогично понятию простоты в кольце целых чисел.

Определение. Полином $f(x)$ называется *неприводимым над* Z_p, если он не имеет делителя в $Z_p[x]$, степень которого больше 0 и меньше, чем $\deg f$.

Имеет место теорема разложения для полиномов, которая формулируется и доказывается аналогично соответствующей теореме для чисел.

Теорема разложения

Всякий полином $f(x) \in Z_p[x]$ единственным образом (с точностью до порядка сомножителей) представим в виде $f(x) = f_n \cdot g_1(x) \cdot \ldots \cdot g_k(x)$, где

$g_1(x)$, …, $g_M(x)$ – нормированные неприводимые над Z_p полиномы, $n = \deg f$.

Утверждение 1. Полином $f(x)$ неприводим над Z_p, если и только если он не делится на неприводимые полиномы степеней 1, 2, …, $\left[\frac{\deg f}{2}\right]$.

Утверждение верно ввиду равенства $\deg(f \cdot g) = \deg f + \deg g$.

Утверждение 2. В $Z_p[x]$ имеет место тождество

$$x^p - x = x \cdot (x - 1) \cdot \ldots \cdot (x - (p - 1)).$$

Верно ввиду *теоремы Ферма*.

Пример. В $Z_5[x]$ выполняется:

$$x^5 - x = x \cdot (x^4 - 1) = x \cdot (x^2 - 1) \cdot (x^2 + 1) = x \cdot (x^2 - 1) \cdot (x^2 - 4) =$$
$$= x \cdot (x - 1) \cdot (x + 1) \cdot (x - 2) \cdot (x + 2) = x \cdot (x - 1) \cdot (x - 4) \cdot (x - 2) \cdot (x - 3).$$

Утверждение 3. В $Z_p[x]$ для любого k полином $x^{p^k} - x$ есть произведение всех различных нормированных неприводимых над Z_p полиномов, степени которых делят k.

Доказательство этого утверждения известно из курса общей алгебры.

Пример. Пусть $p = 3$, $k = 2$.

$$x^9 - x = x \cdot (x^8 - 1) = x \cdot (x^4 - 1) \cdot (x^4 + 1) = x \cdot (x^2 - 1) \cdot (x^2 + 1) \cdot (x^4 + 1) =$$
$$= x \cdot (x - 1) \cdot (x + 1) \cdot (x^2 + 1) \cdot (x^4 + 1),$$

и в соответствии с утверждением 3 множитель $x^2 + 1$ неприводим над Z_3, а полином $x^4 + 1$ раскладывается в произведение двух неприводимых множителей степени 2:

$$x^4 + 1 = (x^2 - x - 1) \cdot (x^2 + x - 1).$$

Тем самым найдены все неприводимые над Z_3 полиномы степеней 1 и 2.

Утверждения 1 и 3 доказывают следующую теорему.

Критерий 1 неприводимости полиномов

Полином $f(x)$ неприводим над Z_p, где p простое, если и только если НОД$(f(x), x^{p^k} - x) = 1$ для всех $k = 1, 2, \ldots, \left[\frac{\deg f}{2}\right]$.

На проверке этого критерия основан первый тест неприводимости.

Тест 1 на неприводимость полиномов

Вход: $f(x) \in Z_p[x]$, $\deg f = n$.

Выход: ответ на вопрос «f приводим?».

1. Положить $u(x) = x$.
2. Выполнять $\left[\frac{n}{2}\right]$ раз:

   2.1. Положить $u(x) = u(x)^p \bmod f(x)$.

   2.2. Вычислить $d(x) =$ НОД$(f(x), u(x) - x)$.

2.3. Если $d(x) \neq 1$, выход, ответ: «приводим».

3. Ответ: «неприводим».

Пример 1. Проверим на неприводимость полином $x^4 + x^3 + 1 \in Z_2[x]$. Для этого надо дважды вычислить НОД:

1) НОД$(x^4 + x^3 + 1, x^2 - x) = 1$;

2) $x^4 = x^3 + 1 \pmod{x^4 + x^3 + 1}$, НОД$(x^4 + x^3 + 1, x^3 - x + 1) = 1$.

Ответ: полином $x^4 + x^3 + 1$ неприводим над Z_2.

Докажем другой критерий неприводимости и сформулируем ещё один тест.

Критерий 2 неприводимости полиномов

Пусть p простое, $f(x) \in Z_p[x]$, $\deg f = n$. Тогда $f(x)$ неприводим над Z_p, если и только если:

1) $f(x) \mid x^{p^n} - x$ и

2) НОД$(x^{p^{n/q}} - x, f(x)) = 1$ для любого простого $q \mid n$.

Доказательство.

Необходимость. Из утверждения 3 следует, что полином $x^{p^n} - x$ делится на все неприводимые полиномы степени n, в частности, на $f(x)$. С другой стороны, полином $x^{p^{n/q}} - x$ не может иметь неприводимого делителя степени n, значит, он не делится на $f(x)$, и в силу неприводимости последнего остаётся единственная возможность НОД$(x^{p^{n/q}} - x, f(x)) = 1$.

Достаточность. Предположим, что $g(x) \mid f(x)$, $g(x)$ неприводим и $\deg g < n$. Из первого условия теоремы следует, что $g(x) \mid x^{p^n} - x$, откуда в силу всё того же утверждения 3 получаем, что $\deg g \mid n$. Из второго условия: $g(x) \nmid x^{p^{n/q}} - x$, и значит, $\deg g \nmid \frac{n}{q}$. Из того, что последнее выполняется для всех простых $q \mid n$, следует $\deg g = n$. Полученное противоречие доказывает теорему.

Тест 2 на неприводимость полиномов

Вход: $f(x) \in Z_p[x]$, $\deg f = n = \prod_{i=1}^{k} p_i^{e_i}$

Выход: ответ на вопрос «f приводим?».

1. Положить $u(x) = x^{p^n} \bmod f(x)$. Если $u(x) \neq x$, выход, ответ: «приводим».
2. Выполнять для $i = 1, \ldots, k$:

   2.1. Положить $u(x) = x^{p^{n/p_i}} \bmod f(x)$.

2.2. Вычислить $d(x) = НОД(u(x) - x, f(x))$.

2.3. Если $d(x) \neq 1$, выход, ответ: «приводим».

3. Ответ: «неприводим».

Заметим, что в отличие от теста 1 данный тест не всегда находит делитель полинома, даже если заканчивает работу с ответом «приводим».

Пример 2. Проверим на неприводимость полином $x^4 + x + 1 \in Z_2[x]$.

1. Вычисляем $x^{16} \bmod (x^4 + x + 1)$:

$x^4 \bmod (x^4 + x + 1) = x + 1$;

$x^8 \bmod (x^4 + x + 1) = x^2 + 1$;

$x^{16} \bmod (x^4 + x + 1) = (x^4 + 1) \bmod (x^4 + x + 1) = x$.

2. $q = 2$:

2.1. $u(x) = x^4 \bmod (x^4 + x + 1) = x + 1$.

2.2. $d(x) = НОД(1, x^4 + x + 1) = 1$.

3. Ответ: $x^4 + x + 1$ неприводим над Z_2.

Пример 3. Проверим на неприводимость полином $x^3 + x + 1 \in Z_3[x]$.

1. Вычисляем $x^{27} \bmod (x^3 + x + 1)$:

$x^3 \bmod (x^3 + x + 1) = -x - 1$;

$x^9 \bmod (x^3 + x + 1) = (-x^3 - 1) \bmod (x^3 + x + 1) = x$;

$x^{27} \bmod (x^3 + x + 1) = x^3 \bmod (x^4 + x + 1) = -x - 1 \neq x$.

Ответ: $x^3 + x + 1$ приводим над Z_3.

Вычисление степеней полиномов в примерах 2, 3 производится с помощью формулы $(u(x))^p = u(x^p)$, которая верна для любого $u(x) \in Z_p[x]$, что будет показано в разделе 5.4.1. Кроме того, заметим, что в примере 3 мы не получили никакой информации о делителях полинома $x^3 + x + 1$.

5.3. Тесты на примитивность полиномов над Z_p

Вспомним сначала известное из курса общей алгебры понятие показателя полинома.

Определение. Показателем полинома $g(x)$ называется наименьшее целое положительное число m такое, что $g(x) | x^m - 1$.

Или, что то же самое: показатель $g(x)$ – наименьшее целое положительное число m такое, что $x^m = 1 \pmod g$.

Ещё одна формулировка: показатель полинома $g(x) \in Z_p[x]$ равен порядку элемента x в группе $Z_p^*[g]$ обратимых элементов кольца $Z_p[g] = Z_p[x]/(g)$ классов вычетов полиномов по модулю полинома $g(x)$.

Пусть $g(x) \in Z_p[x]$ и $\deg g = n$. Будем обозначать показатель полинома $g(x)$ через $O_p(g)$. Перечислим некоторые свойства показателя.

1. Показатель существует для всякого $g(x)$ такого, что $g(0) \neq 0$.
2. $O_p(g) \leq p^n - 1$.
3. Если $g(x)$ неприводим над Z_p, то $O_p(g) \mid (p^n - 1)$.

Последнее свойство следует из того, что для неприводимого полинома $g(x)$ кольцо $Z_p[g]$ является полем и по *теореме Лагранжа* порядок элемента x является делителем порядка группы $|Z_p^*[g]| = p^n - 1$. Большой интерес вызывают полиномы, имеющие максимально возможный показатель; такие полиномы называются примитивными.

Определение. Полином $g(x) \in Z_p[x]$ степени n называется *примитивным*, если его показатель равен $p^n - 1$.

Из определения следует, что примитивный полином неприводим, так как в противном случае $|Z_p^*[g]| < p^n - 1$. Таким образом, приходим к другой формулировке этого понятия: неприводимый полином $g(x) \in Z_p[x]$ называется *примитивным*, если x является порождающим элементом группы $Z_p^*[g]$. Последнее, в свою очередь, равносильно условию: $x^{\frac{p^n-1}{q}} \neq 1 (\mathrm{mod}\, g)$ для любого простого $q \mid p^n - 1$. Следовательно, необходимо знать разложение числа $p^n - 1$ на простые множители. Для случая, когда это разложение неизвестно, не существует эффективных способов проверки примитивности полиномов.

Тест на примитивность

Вход: неприводимый полином $g(x) \in Z_p[x]$ степени n, разложение на простые множители числа $p^n - 1$: $p^n - 1 = \prod_{i=1}^{k} p_i^{e_i}$.

Выход: ответ на вопрос «g примитивный?»

1. Выполнять для $i = 1, \ldots, k$:

   1.1. Вычислить $r(x) = x^{(p^n-1)/p_i} \bmod g(x)$.

   1.2. Если $r(x) = 1$, выход, ответ: «не примитивный».
2. Ответ: «примитивный».

Пример. Ранее мы установили (пример 2 в предыдущем разделе) неприводимость полинома $x^4 + x + 1$ над Z_2. Проверим этот же полином на примитивность.

$2^4 - 1 = 15 = 5 \cdot 3$.

Вычисляем $r(x) = x^5 \bmod (x^4 + x + 1) = x^2 + x \neq 1$;

$r(x) = x^3 \bmod(x^4 + x + 1) = x^3 \neq 1$.
Ответ: полином $x^4 + x + 1$ примитивный.

Возвратные полиномы

Определение. Возвратным полиномом к полиному $f(x)$ степени n называется полином $f^*(x) = x^n \cdot f(1/x)$.

Другими словами, вектор коэффициентов возвратного полинома получается реверсированием (переписыванием в обратном порядке) вектора коэффициентов исходного полинома. Интерес к этому понятию возник в данном разделе в связи со следующим утверждением.

Утверждение. Если полином $f(x)$ примитивный, то и $f^*(x)$ также примитивный.

Это верно в силу следующих легко проверяемых свойств.

1. Если $f(0) \neq 0$, то $\deg f = \deg f^*$ и $(f^*)^* = f$.
2. $(fg)^*(x) = f^*(x) \cdot g^*(x)$.
3. Если $f(0) \neq 0$ и $f(x)$ неприводим, то и $f^*(x)$ неприводим.
4. Если $x^r = 1 (\bmod f)$, то $x^r = 1 (\bmod f^*)$.

В заключение раздела отметим одно из важнейших применений примитивных полиномов в криптографии. А именно, доказан следующий факт: регистр сдвига с линейной обратной связью порождает последовательность максимального периода из любого ненулевого начального состояния, если и только если его характеристический полином примитивен. Так что все рассмотренные в данной главе алгоритмы имеют большое практическое значение для задачи синтеза поточных шифров.

5.4. Факторизация полиномов над Z_p

Задача факторизации полиномов находит своё применение в методах криптоанализа (в частности, при вычислении дискретных логарифмов в поле $GF(2^k)$) и в некоторых криптосистемах, например, в криптосистеме с открытым ключом Шора–Ривеста. Рассмотрим алгоритм факторизации полиномов над полем Z_p, который имеет как самостоятельное значение, так и может применяться для факторизации полиномов над кольцом целых чисел Z: результат разложения полинома над Z можно найти с помощью разложений над Z_p для нескольких p, а задача разложения над Z_p решается легче, чем над Z. Вообще о сложности задачи факторизации полиномов можно сказать следующее: на настоящий момент, так же как для задач факторизации чисел и дискретного логарифмирования, не найден полиномиальный алгоритм её решения, но и не доказано, что такого

алгоритма не существует. Вместе с тем считается, что разложить на множители полином степени n над Z_2 легче, чем факторизовать n-битное число. Задача факторизации полинома решается в два этапа: освобождение полинома от квадратов и затем разложение на множители свободных от квадратов полиномов.

5.4.1. Освобождение от квадратов

Говорят, что полином $f(x) \in Z_p[x]$ *свободен от квадратов*, если его можно представить в виде $f(x) = f_n \cdot g_1(x) \cdot \ldots \cdot g_k(x)$, где $g_1(x)$, ..., $g_k(x)$ – различные нормированные неприводимые над Z_p полиномы, $n = \deg f$. Задача освобождения от квадратов ставится так: представить полином $f(x)$ в виде $f(x) = f_n \cdot h_1(x) \cdot \ldots \cdot h_k(x)$, где $h_1(x)$, ..., $h_k(x)$ – нормированные свободные от квадратов, не обязательно неприводимые, не обязательно различные полиномы. Докажем несколько необходимых для формулировки алгоритма утверждений.

Утверждение 1. Для любого $v(x) \in Z_p[x]$ имеет место тождество $(v(x))^p = v(x^p)$.

Утверждение верно в силу теоремы Ферма и бинома Ньютона:

$$(v(x))^p = (v_n x^n + v_{n-1} x^{n-1} + \ldots + v_1 x + v_0)^p = v_n^p x^{np} + v_{n-1}^p x^{(n-1)p} + \ldots + v_1^p x + v_0^p =$$
$$= v_n x^{np} + v_{n-1} x^{(n-1)p} + \ldots + v_1 x^p + v_0 = v(x^p).$$

Вспомним понятие производной полинома над конечным полем. Пусть $f(x) = f_n \cdot x^n + \ldots + f_1 \cdot x + f_0$. Тогда

$$f(x + y) = f_n(x + y)^n + \ldots + f_1(x + y) + f_0.$$

Раскроем в последнем выражении скобки и перегруппируем слагаемые, собрав вместе мономы, в которые y входит в одинаковой степени. Получим:

$$f(x + y) = g_0(x) + g_1(x) \cdot y + g_2(x) \cdot y^2 + \ldots + g_n(x) \cdot y^n.$$

Нетрудно заметить, что $g_0(x) = f(x)$. Полином $g_1(x)$ называется *производной* $f(x)$ и обозначается $f'(x)$. Производная вычисляется по обычным правилам; в частности:

$(f^{-1}(x))' = e \cdot f^{-1}(x) \cdot f'(x)$;

$(f_1(x) \cdot f_2(x))' = f_1'(x) \cdot f_2(x) + f_1(x) \cdot f_2'(x)$.

Утверждение 2. Если $f'(x) = 0$, то $f(x) = (v(x))^p$ для некоторого $v(x)$.

Доказательство.

Если $f(x) = f_n \cdot x^n + f_{n-1} \cdot x^{n-1} + \ldots + f_1 \cdot x + f_0$, то $f'(x) = n \cdot f_n \cdot x^{n-1} + (n-1) \cdot f_{n-1} \cdot x^{n-2} + \ldots + f_1$, и это последнее выражение равно тождественно 0, если и только если $f_i = 0$ для всех $p \nmid i$, т. е. $f(x) = f_{pk} \cdot x^{pk} + f_{p(k-1)} \cdot x^{p(k-1)} + \ldots + f_p \cdot x^p + f_0 = v(x^p) = (v(x))^p$.

Утверждение 3. Полином $f(x)$ свободен от квадратов, если и только если $d(x) = \text{НОД}(f(x), f'(x)) = 1$.

Доказательство.

Необходимость. Пусть $f(x) = g_1(x) \cdot \ldots \cdot g_k(x)$, где $g_1(x), \ldots, g_k(x)$ – различные неприводимые полиномы. Предположим, что $d(x) \neq 1$. Тогда существует $i \in \{1, \ldots, k\}$ такое, что $g_i(x) | d(x)$. Представим $f(x)$ в виде $f(x) = g_i(x) \cdot v(x)$; заметим, что $\text{НОД}(g_i(x), v(x)) = 1$. Отсюда, а также из условия $g_i(x) | f'(x) = g_i'(x) \cdot v(x) + g_i(x) \cdot v'(x)$, следует, что $g_i(x) | g_i'(x)$. Поскольку $\deg g_i' < \deg g_i$, то последнее возможно только в случае $g_i'(x) = 0$, что, в свою очередь, невозможно в силу неприводимости g_i. Необходимость доказана.

Достаточность. Предположим, что существует полином $g(x)$ такой, что $g^2(x) | f(x)$. Тогда

$$f(x) = g^2(x) \cdot v(x), \; f'(x) = 2 \cdot g(x) \cdot g'(x) \cdot v(x) + g^2(x) \cdot v'(x),$$

откуда $g(x) | f'(x)$, что противоречит условию $\text{НОД}(f(x), f'(x)) = 1$. Утверждение доказано.

Утверждение 4. Полином $\dfrac{f(x)}{\text{НОД}(f(x), f'(x))}$ свободен от квадратов.

Следует из того, что если $g^k(x) | f(x)$, то $g^{k-1}(x) | f'(x)$. Следовательно, любой полином $g(x)$ входит в разложение $\dfrac{f(x)}{\text{НОД}(f(x), f'(x))}$ не более чем в первой степени.

Утверждения 3, 4 обосновывают следующий рекурсивный алгоритм освобождения полиномов от квадратов.

Алгоритм освобождения $f(x)$ от квадратов – Square-Free(f)

Вход: $f(x) \in Z_p[x]$.

Выход: представление $f(x)$ в виде $f(x) = h_1^{e_1}(x) \cdot \ldots \cdot h_k^{e_k}(x)$, где $h_1(x), \ldots, h_k(x)$ –свободные от квадратов полиномы.

1. Если $\deg f < 2$, то выход, ответ: $f(x)$.
2. Вычислить $g(x) = f'(x)$.
3. Если $g(x) = 0$, то $f(x) = (v(x))^p$. Ответ: (Square-Free(v))^p.
4. Вычислить $d(x) = \text{НОД}(f(x), g(x))$. Если $d(x) = 1$, то выход, ответ: $f(x)$.
5. Иначе получить разложение $f(x) = d(x) \cdot h(x)$.
6. Ответ: Square-Free(d)$\cdot h(x)$.

Пример 1. Рассмотрим полином

$f(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1 \in Z_2[x]$.

2. $g(x) = f'(x) = x^6 + x^4$.

4. $d(x) = \text{НОД}(x^7 + x^6 + x^5 + x^4 + x^2 + 1, x^6 + x^4) = x^2 + 1$.
5. $f(x) = (x^2 + 1)\cdot(x^5 + x^4 + x^2 + 1)$.
6. Square-Free($x^2 + 1$):
   2. $g(x) = f'(x) = 0$.
   3. $x^2 + 1 = (x + 1)^2$.

Ответ: $f(x) = (x + 1)^2\cdot(x^5 + x^4 + x^2 + 1)$.

Пример 2. Рассмотрим полином

$f(x) = x^5 + 5x^3 + 4x^2 + 2x + 4 \in Z_7[x]$.

2. $g(x) = f'(x) = 5x^4 + x^2 + x + 2$.
4. $d(x) = \text{НОД}(x^5 + 5x^3 + 4x^2 + 2x + 4, 5x^4 + x^2 + x + 2) = x^3 + 4x^2 + 5x + 2$.
5. $f(x) = (x^3 + 4x^2 + 5x + 2)\cdot(x^2 + 3x + 2)$.
6. Square-Free($x^3 + 4x^2 + 5x + 2$):
   2. $g(x) = f'(x) = 3x^2 + x + 5$.
   4. $d(x) = \text{НОД}(x^3 + 4x^2 + 5x + 2, 3x^2 + x + 5) = x + 1$.
   5. $f(x) = (x + 1)\cdot(x^2 + 3x + 2)$.

Ответ: $f(x) = (x + 1)\cdot(x^2 + 3x + 2)^2$.

5.4.2. Алгоритм Берлекэмпа

Алгоритм разложения свободного от квадратов полинома на неприводимые над Z_p множители открыт Элвином Р. Берлекэмпом в 1967 г. и эффективен в случае малого p. Обоснованием алгоритма служат следующие теоремы.

Китайская теорема об остатках для полиномов

Пусть $f_1(x)$, …, $f_k(x)$ – попарно простые полиномы над Z_p, $g_1(x)$, …, $g_k(x)$ – произвольные полиномы над Z_p. Тогда существует единственный полином $h(x)$ такой, что:

1) $h(x) = g_i(x) \pmod{f_i(x)}$ для всех $i = 1, \ldots, k$;

2) $\deg h < \sum_{i=1}^{k} \deg f_i$.

Теорема доказывается так же, как аналогичная теорема для чисел.

Теорема 1. Пусть $f(x) \in Z_p[x]$ – нормированный полином, $h(x) \in Z_p[x]$ – полином, удовлетворяющий сравнению

$$(h(x))^p = h(x) \pmod{f(x)}.$$

Тогда $f(x) = \prod_{a=0}^{p-1} \text{НОД}(f(x), h(x) - a)$.

Доказательство. Обозначим

$$d_a(x) = \text{НОД}(f(x), h(x) - a), \quad g(x) = \prod_{a=0}^{p-1} d_a(x).$$

Заметим, что полиномы $h(x) - a$, $a = 0, \ldots, p-1$, попарно просты, что влечёт попарную простоту полиномов $d_a(x)$. Кроме того, $d_a(x) | f(x)$ для любого a. Следовательно, $g(x) | f(x)$.

С другой стороны, в Z_p справедливо тождество $x^p - x = x \cdot (x-1) \cdot \ldots \cdot (x-(p-1))$, откуда следует, что

$$(h(x))^p - h(x) = \prod_{a=0}^{p-1} (h(x) - a).$$

Перепишем условие $(h(x))^p = h(x) \pmod{f(x)}$ в виде $f(x) | (h(x))^p - h(x)$, т. е. $f(x) | \prod_{a=0}^{p-1} (h(x) - a)$. Значит,

$$f(x) | \text{НОД}(f(x), \prod_{a=0}^{p-1} (h(x) - a)) = \prod_{a=0}^{p-1} \text{НОД}(f(x), h(x) - a) = g(x).$$

Здесь предпоследнее равенство имеет место в силу свойства мультипликативности НОД: если $(b, c) = 1$, то $(a, bc) = (a, b) \cdot (a, c)$. Из условий $g(x) | f(x)$ и $f(x) | g(x)$ и того, что оба полинома $g(x)$ и $f(x)$ нормированы, следует равенство $f(x) = g(x)$. Теорема доказана.

Теорема 2. Пусть $f(x) \in Z_p[x]$ и $f(x) = f_1(x) \cdot \ldots \cdot f_k(x)$, где $f_1(x), \ldots, f_k(x)$ – различные нормированные неприводимые над Z_p полиномы. Тогда полином $h(x)$ удовлетворяет сравнению

$$(h(x))^p = h(x) \pmod{f(x)},$$

если и только если для любого $i = 1, \ldots, k$ имеет место: $h(x) = a_i \pmod{f_i(x)}$ для некоторого $a_i \in Z_p$. Каждому набору $(a_1, \ldots, a_k) \in Z_p^k$ соответствует ровно один полином, степень которого меньше, чем $\deg f$.

Доказательство. Последнее утверждение следует из Китайской теоремы об остатках для полиномов.

Необходимость. Условие $(h(x))^p = h(x) \pmod{f(x)}$ равносильно делимости $f(x) | \prod_{a=0}^{p-1} (h(x) - a)$, откуда следует, что $f_i(x) | \prod_{a=0}^{p-1} (h(x) - a)$ для любого $i = 1, \ldots, k$. В силу неприводимости $f_i(x)$ это означает, что существует такое $a_i \in Z_p$, что $f_i(x) | (h(x) - a_i)$. Следовательно, $h(x) = a_i \pmod{f_i(x)}$.

Достаточность. Если $h(x) = a_i \pmod{f_i(x)}$, то

$$(h(x))^p = a_i^p = a_i = h(x) \pmod{f_i(x)},$$

а так как это верно для любого i и полиномы $f_i(x)$ попарно просты, то $(h(x))^p = h(x) \pmod{f(x)}$. Теорема доказана.

Осмыслим полученные результаты. Теорема 1 сводит задачу факторизации полиномов к решению сравнения

$$(h(x))^p = h(x) \pmod{f(x)}. \qquad (*)$$

Пусть $h_1(x)$, …, $h_s(x)$ – все линейно независимые решения этого сравнения. Рассмотрим любую их линейную комбинацию:

$$a(x) = a_1 \cdot h_1(x) + \ldots + a_s \cdot h_s(x), \text{ где } a_1, \ldots, a_s \in Z_p.$$

Тогда
$$(a(x))^p = a_1^p \cdot (h_1(x))^p + \ldots + a_s^p \cdot (h_s(x))^p =$$
$$= a_1 \cdot h_1(x) + \ldots + a_s \cdot h_s(x) = a(x) \pmod{f(x)},$$

т.е. полином $a(x)$ – также решение сравнения (*). Таким образом, это сравнение имеет p^s различных решений.

С другой стороны, теорема 2 характеризует все решения сравнения (*), сопоставляя им наборы $(a_1, \ldots, a_k) \in Z_p^k$. Количество решений совпадает с количеством наборов и равно p^k, где k – количество неприводимых множителей в разложении полинома $f(x)$. Получаем отсюда, что $k = s$ – количеству линейно независимых решений сравнения (*).

Кроме того, заметим следующее. Если $h(x) = a_i \pmod{f_i(x)}$ и $h(x) = a_j \pmod{f_j(x)}$ и $a_i \neq a_j$, то $f_i(x) \mid h(x) - a_i$, но $f_j(x) \nmid h(x) - a_i$. Значит, $f_i(x) \mid$НОД$(f(x), h(x) - a_i)$ и $f_j(x) \nmid$ НОД$(f(x), h(x) - a_i)$. Другими словами, полином НОД$(f(x), h(x) - a_i)$ «разделяет» неприводимые делители $f_i(x)$ и $f_j(x)$, и, перебрав всевозможные $h(x)$ – решения сравнения (*), и всевозможные $a_i \in Z_p$, найдём все неприводимые делители полинома $f(x)$.

Осталось рассмотреть задачу решения сравнения (*). Пусть $\deg f = n$; заметим, что $\deg h < n$. Представим $h(x)$ в виде

$$h(x) = t_0 + t_1 \cdot x + \ldots + t_{n-1} \cdot x^{n-1}.$$

Тогда $(h(x))^p = h(x^p) = t_0 + t_1 \cdot x^p + \ldots + t_{n-1} \cdot x^{p \cdot (n-1)} \pmod{f(x)}$.
Вычислим $x^{p \cdot j} \bmod f(x)$ для $j = 0, \ldots, n-1$; пусть

$$x^{p \cdot j} \bmod f(x) = \sum_{i=0}^{n-1} q_{ji} x^i .$$

Тогда можно записать:

$$(h(x))^p = \sum_{j=0}^{n-1} t_j \cdot \sum_{i=0}^{n-1} q_{ji} x^i = \sum_{i=0}^{n-1} x^i \cdot \sum_{j=0}^{n-1} t_j q_{ji} = h(x) = \sum_{i=0}^{n-1} t_i x^i .$$

Отсюда имеем систему линейных уравнений:

$$\sum_{j=0}^{n-1} t_j q_{ji} = t_i, i = 1, \ldots, n-1 ,$$

или в матричном виде $T \cdot (Q - E) = 0$,
где $T = [t_0 \ldots t_{n-1}]$ – искомый вектор коэффициентов полинома $h(x)$;

$Q = \|q_{ji}\|$ – матрица, строки которой суть коэффициенты полиномов $x^{pj} \bmod f(x)$ для $j = 0, \ldots, n-1$;

E – единичная матрица порядка n.

Пусть r – ранг матрицы $Q - E$. Тогда система имеет $k = n - r$ линейно независимых решений $h^{(1)}(x), \ldots, h^{(k)}(x)$, и по доказанному выше k есть количество неприводимых делителей полинома $f(x)$. Сами делители найдём, вычисляя НОД$(f(x), h^{(i)}(x) - a)$ для $i = 1, \ldots, k$, $a = 0, \ldots, p-1$.

Алгоритм Берлекэмпа

Вход: $f(x) \in Z_p[x]$ – свободный от квадратов нормированный полином степени n.

Выход: $F = \{f_1(x), \ldots, f_k(x)\}$ – множество всех нормированных неприводимых над Z_p делителей $f(x)$.

1. Положить $F = \{f(x)\}$.
2. Построить матрицу Q, j-я строка которой представляет собой вектор коэффициентов полинома $x^{pj} \bmod f(x)$, $j = 0, \ldots, n-1$.
3. Вычислить $r = \text{rang}\,(Q - E)$, $k = n - r$. Если $k = 1$, выход.
4. Найти $h^{(1)}(x), \ldots, h^{(k)}(x)$ – линейно независимые решения системы линейных однородных уравнений $T \cdot (Q - E) = 0$, положив $h^{(1)}(x) = 1$ (тривиальное решение).
5. Для $i = 2, \ldots, k$ выполнять:
   5.1. Удалить из множества F любой полином $v(x)$, степень которого больше 1.
   5.2. Вычислить полиномы НОД$(v(x), h^{(i)}(x) - a)$ для $a = 0, \ldots, p-1$ и добавить в множество F те из них, степени которых больше 0.
   5.3. Если $|F| = k$, выход.

Сложность алгоритма равна $O(n^3 + k \cdot p \cdot n^2)$; таким образом, алгоритм эффективно работает в полях небольшого порядка.

Пример 1. Рассмотрим полином $f(x) = x^5 + x^3 + x + 1 \in Z_2[x]$.

1. $F = \{f(x)\}$.
2. $x^0 = 1 \pmod{f(x)}$; $x^2 = x^2 \pmod{f(x)}$; $x^4 = x^4 \pmod{f(x)}$;
   $x^6 = x^4 + x^2 + x \pmod{f(x)}$; $x^8 = x^3 + x^2 + x \pmod{f(x)}$.

Таким образом, $Q = \begin{Vmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{Vmatrix}$, $Q - E = \begin{Vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{Vmatrix}$.

3. $r = \text{rang}\,(Q - E) = 3$, $k = n - r = 2$.
4. $h^{(1)}(x) = 1$, $h^{(2)}(x) = x^4 + x^3$.

5. $\nu(x) = x^5 + x^3 + x + 1$;
   НОД$(x^5 + x^3 + x + 1, x^4 + x^3) = x + 1$;
   НОД$(x^5 + x^3 + x + 1, x^4 + x^3 + 1) = x^4 + x^3 + 1$.
   Ответ: $F = \{x + 1, x^4 + x^3 + 1\}$.

Пример 2. Рассмотрим полином $f(x) = x^4 + 3x^2 + 2 \in Z_5[x]$.

1. $F = \{f(x)\}$.
2. $x^0 = 1 \pmod{f(x)}$; $x^5 = 2x^3 + 3x \pmod{f(x)}$;
   $x^{10} = x^2 \pmod{f(x)}$; $x^{15} = 2x^3 + x \pmod{f(x)}$.

   Таким образом, $Q = \begin{Vmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{Vmatrix}$, $Q - E = \begin{Vmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{Vmatrix}$.
3. $r = \text{rang}\,(Q - E) = 1$, $k = n - r = 3$.
4. $h^{(1)}(x) = 1$, $h^{(2)}(x) = 3x^3 + x$, $h^{(3)}(x) = x^2$.
5. $\nu(x) = x^4 + 3x^2 + 2$;
   НОД$(x^4 + 3x^2 + 2, 3x^3 + x) = x^2 + 2$;
   НОД$(x^4 + 3x^2 + 2, 3x^3 + x - 1) = x - 2$;
   НОД$(x^4 + 3x^2 + 2, 3x^3 + x - 2) = 1$;
   НОД$(x^4 + 3x^2 + 2, 3x^3 + x - 3) = 1$;
   НОД$(x^4 + 3x^2 + 2, 3x^3 + x - 4) = x + 2$.
   Ответ: $F = \{x^2 + 2, x - 2, x + 2\}$.

ЛИТЕРАТУРА

Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.

Белов А.Г. Исследование алгоритма дискретного логарифмирования Адлемана // Вестник Томского госуниверситета. Приложение. 2005. № 14. С. 45–49.

Белов А.Г., Панкратова И.А. Сравнительный анализ двух генераторов простых чисел // Вестник Томского госуниверситета. Приложение. 2007. № 23. С. 77–80.

Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. М.: МЦНМО, 2002.

Введение в криптографию / Под. ред. В.В. Ященко. М.: МЦНМО – ЧеРо, 1998.

Земор Ж. Курс криптографии. Москва; Ижевск: НИЦ «Регулярная и хаотическая динамика», 2006.

Кнут Д. Искусство программирования для ЭВМ. Т. 2: Получисленные алгоритмы. М.: Мир, 1977.

Коблиц Н. Курс теории чисел и криптографии. М.: Научное изд-во ТВП, 2001.

Маховенко Е.Б. Математические основы криптографии: Конспект лекций. СПб.: Изд-во СПбГТУ, 1999.

Маховенко Е.Б. Теоретико-числовые методы в криптографии. М.: Гелиос АРВ, 2006.

Миллер Г.Л. Гипотеза Римана и способы проверки простоты чисел // Кибернетический сборник. 1986. Вып. 23. С. 31–50.

Ноден П., Китте К. Алгебраическая алгоритмика. М.: Мир, 1999.

Саломаа А. Криптография с открытым ключом. М.: Мир, 1996.

Уильямс Х. Проверка чисел на простоту с помощью вычислительных машин // Кибернетический сборник. 1986. Вып. 23. С. 51–99.

Фергюсон Н., Шнайер Б. Практическая криптография. Москва; Санкт-Петербург; Киев: Диалектика, 2005.

Харин Ю.С., Берник В.И., Матвеев Г.В., Агиевич С.В. Математические и компьютерные основы криптологии. Минск: Новое знание, 2003.

Черёмушкин А.В. Вычисления в алгебре и теории чисел: Курс лекций. М., 2002.

Черёмушкин А.В. Лекции по арифметическим алгоритмам в криптографии. М.: МЦНМО, 2002.

Coutinho S.C. The Mathematics of Ciphers: Number Theory and RSA Cryptography. A K Peters: Natick, Massachusetts, 1999.

Menezes A.J., Van Oorshot P.C., Vanstone S.A. Handbook of Applied Cryptography. N.Y.: CRC Press Series on Discrete Mathematics and Its Applications, 1997.

Приложение

Некоторые определения и теоремы алгебры и теории чисел

Элемент b называется *обратным к a по модулю m*, если $ab = 1 \pmod m$.

Элемент a называется *обратимым по модулю m*, если для него существует обратный элемент по этому модулю.

Теорема обратимости. Элемент a обратим по модулю m, если и только если $(a, m) = 1$.

Китайская теорема об остатках (Сунь Цзе, I в. до н. э.)

Пусть $m_1, \ldots, m_k$ – попарно простые числа. Тогда система сравнений

$$\begin{cases} x = a_1 (\bmod m_1), \\ \ldots \\ x = a_k (\bmod m_k) \end{cases}$$

имеет одно и только одно решение по модулю $M = m_1 \cdot \ldots \cdot m_k$, а именно $x = \sum_{i=1}^{k} M_i \cdot M_i' \cdot a_i$, где $M_i = M / m_i$, $M_i' = M_i^{-1} \bmod m_i$.

Формула Гарнера задаёт решение системы сравнений по попарно простым модулям. Рассмотрим частный случай – систему из двух сравнений. Пусть задана система

$$\begin{cases} x = a (\bmod p) \\ x = b (\bmod q) \end{cases},$$

где $(p, q) = 1$.

Тогда её решение определяется формулой

$$x = (((a - b) \cdot (q^{-1} \bmod p)) \bmod p) \cdot q + b.$$

Функция Эйлера $\varphi(n)$ определена для любого положительного n и равна количеству чисел в ряду $1, \ldots, n$, взаимно простых с n.

Показателем числа x по модулю m называется наименьшее положительное число n, удовлетворяющее условию $x^n = 1 \pmod m$.

Число, показатель которого равен функции Эйлера $\varphi(m)$, называется *первообразным корнем по модулю m*.

Приведенной системой вычетов по модулю n называется множество всех взаимно простых с n и попарно несравнимых по модулю n чисел. Приведённая система вычетов, являющаяся подмножеством $Z_n = \{0, 1, \ldots, n - 1\}$, обозначается $U(n)$ и образует *группу* с операцией умножения по модулю n.

Число a, взаимно простое с m, называется *квадратичным вычетом* по модулю m, если квадратное сравнение $x^2 = a \pmod m$ имеет решения; в противном случае a называется *квадратичным невычетом* по модулю m.

Пусть p – простое нечётное, a – любое целое. *Символ Лежандра* $\left(\frac{a}{p}\right)$ определяется следующим образом:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, \text{ если } p \mid a, \\ 1, \text{ если } a \text{ – квадратичный вычет по модулю } p, \\ -1, \text{ если } a \text{ – квадратичный невычет по модулю } p \end{cases}$$

Пусть n – нечётное, $n = p_1 \cdot \ldots \cdot p_k$, a – любое целое. *Символ Якоби* $\left(\frac{a}{n}\right)$ определяется как произведение символов Лежандра:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \cdot \ldots \cdot \left(\frac{a}{p_k}\right).$$

Теорема Ферма. Пусть p – простое. Тогда $x^p = x \pmod p$ для любого x.

Множество G с определённой на нём операцией $\bullet$ называется *группой*, если выполнены следующие аксиомы группы:

1) замкнутость: $a, b \in G \Rightarrow a \bullet b \in G$;
2) ассоциативность: $a \bullet (b \bullet c) = (a \bullet b) \bullet c$;
3) существование нейтрального элемента:
$\exists e \in G \; \forall a \in G \; (e \bullet a = a \bullet e = a)$;
4) существование обратного элемента:
$\forall a \in G \; \exists a^{-1} \in G \; (a^{-1} \bullet a = a \bullet a^{-1} = e)$.

Пусть n – целое положительное число, $(G, \bullet)$ – группа, и $a \in G$. *n-ой степенью элемента a* называется элемент $a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ раз}}$.

Группа G называется *циклической*, если все её элементы суть степени некоторого одного элемента g, называемого *порождающим элементом группы G*. Обозначается такая группа $\langle g \rangle$: $\langle g \rangle = \{g, g^2, g^3, \ldots\}$.

Порядком конечной группы называется количество элементов в ней. *Порядком элемента w* группы G называется наименьшее положительное число n со свойством $w^n = e$.

Теорема Лагранжа. В конечной группе порядок любого элемента является делителем порядка группы.

Группа G называется *абелевой*, если операция в ней коммутативна, т. е. $\forall a, b \in G \; (a \bullet b = b \bullet a)$.

Множество R с заданными на нём операциями $+$ и $\cdot$ называется *кольцом*, если:

1. $(R, +)$ – абелева группа.
2. $(R, \cdot)$ – полугруппа, т.е. имеют место замкнутость множества R относительно умножения и ассоциативность умножения.
3. Операция умножения дистрибутивна относительно сложения:

$a \cdot (b + c) = a \cdot b + a \cdot c$; $(a + b) \cdot c = a \cdot c + b \cdot c$.

Кольцо R называется *коммутативным*, если коммутативна операция умножения. Кольцо R называется *кольцом с* 1, если в нём существует 1 – нейтральный элемент по умножению.

Множество F с заданными на нём операциями $+$ и $\cdot$ называется *полем*, если:

1. $(F, +)$ – абелева группа.
2. $(F - \{0\}, \cdot)$ – абелева группа.
3. Операция умножения дистрибутивна относительно сложения.

Учебное издание

Ирина Анатольевна Панкратова

ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ

Учебное пособие

Издание подготовлено в авторской редакции
Оригинал-макет – А.И. Лелюр
Дизайн обложки – А.В. Бабенко

Подписано к печати 19.01.2009 г. Формат 60x84/16.
Ризография. Бумага офсетная. Гарнитура Times.
Усл. печ. л. 6,9. Тираж 50 экз. Заказ № 18.

Отпечатано на оборудовании
редакционно-издательского отдела
Томского государственного университета
634050, г. Томск, пр. Ленина, 36. Корп. 4. Оф. 011
Тел. 8+(382-2)–52-98-49