4.1 Элементы теории чисел

Делимость, остатки и сравнения по модулю

Целые числа — самые знакомые объекты в математике, и кажется, что про них трудно сказать что-то новое. Однако именно из вопросов о *делимости* целых чисел вырос целый раздел математики — **теория чисел**, — и именно её результаты сейчас работают внутри каждого мессенджера и банковской карты.

Определение 4.1

Говорят, что целое число a **делится** на целое число $b \neq 0$ (или что b **делит** a), если существует такое целое число q, что $a = b \cdot q$. Пишут $b \mid a$.

Если a не делится на b, то всё равно можно записать

$$a = b \cdot q + r, \qquad 0 \leq r < |\, b \,|\,. \tag{1}$$

Это знакомое со школы **деление с остатком**. Число q называется *неполным частным*, а r — **остатком от деления** a на b. Остаток обозначают также $a \bmod b$ (читается «a по модулю b»).

Пример 4.2

$17 = 5 \cdot 3 + 2$, поэтому $17 \bmod 5 = 2$.
$-17 = 5 \cdot (-4) + 3$, поэтому $(-17) \bmod 5 = 3$ (остаток — неотрицательное число!).

Простые числа — «атомы» делимости.

Определение 4.3

Натуральное число $p > 1$ называется **простым**, если оно делится только на единицу и на само себя. Натуральное $n > 1$, не являющееся простым, называется **составным**.

Первые простые числа: $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots$. Простых чисел бесконечно много — это доказал ещё Евклид (доказательство мы повторим в п. 4.1).

Основная теорема арифметики говорит, что любое натуральное число $n > 1$ *единственным* (с точностью до порядка) способом раскладывается в произведение простых:

$$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}. \tag{2}$$

Сравнения по модулю. В каждой задаче, где важна только «остаточная» информация (например, день недели в зависимости от номера дня в году), удобно вместо чисел рассматривать их остатки от деления на некоторое фиксированное число.

Определение 4.4

Пусть $m \geq 2$ — натуральное число (**модуль**). Целые числа a и b называются **сравнимыми по модулю m**, если их разность $a - b$ делится на m. Пишут

$$a \equiv b \pmod{m}. \tag{3}$$

Равносильное определение: $a \equiv b \pmod{m}$ тогда и только тогда, когда a и b дают одинаковые остатки при делении на m.

Пример 4.5

$23 \equiv 9 \pmod{7}$, потому что $23 - 9 = 14 = 2 \cdot 7$.
В обоих случаях остаток от деления на 7 равен 2.

Сравнения — очень удобный язык, потому что они *ведут себя как обычные равенства*: их можно складывать, вычитать, умножать и возводить в степень.

Теорема 4.6. свойства сравнений

Если $a \equiv b \pmod{m}$ и $c \equiv d \pmod{m}$, то:

1. $a + c \equiv b + d \pmod{m}$;
2. $a - c \equiv b - d \pmod{m}$;
3. $a \cdot c \equiv b \cdot d \pmod{m}$;
4. $a^n \equiv b^n \pmod{m}$ для любого натурального n.

Доказательство. По определению $a - b = m \cdot k_1$ и $c - d = m \cdot k_2$. Тогда $(a + c) - (b + d) = m(k_1 + k_2)$ делится на

m. Для умножения: $ac - bd = ac - bc + bc - bd = c(a-b) + b(c-d)$, что тоже делится на m. Возведение в степень — следствие умножения. ▫

С *делением* ситуация сложнее. Нельзя «просто так» поделить обе части сравнения — например, $6 \equiv 0 \pmod 6$, но «сократив на 2», получим $3 \equiv 0 \pmod 6$, что неверно. Когда деление законно — мы разберём чуть позже, после знакомства с алгоритмом Евклида.

Это интересно

Сравнения встречаются повсюду в жизни. Часы — арифметика по модулю 12 (или 24): если сейчас 10 часов, то через 5 часов будет $10 + 5 = 15 \equiv 3 \pmod{12}$. День недели — арифметика по модулю 7. Контрольная цифра ИНН, номер банковской карты, штрихкод книги — все они вычисляются как сумма цифр (с разными весами) по некоторому модулю.

Алгоритм Евклида

Одна из главных операций в теории чисел — нахождение **наибольшего общего делителя** (НОД) двух чисел.

i Определение 4.7

Наибольший общий делитель чисел a и b (не равных одновременно нулю) — это наибольшее натуральное число, на которое делятся одновременно и a, и b. Обозначается $\gcd(a, b)$.

Если $\gcd(a, b) = 1$, то числа a и b называются **взаимно простыми**.

Как найти $\gcd(a, b)$? Можно, например, разложить оба числа на простые множители и взять общие. Но при больших числах разложение на множители — очень тяжёлая задача (об этом мы поговорим в конце параграфа). К счастью, ещё в III веке до н. э. Евклид предложил способ, который работает несравнимо быстрее.

Геометрическая идея. Представим себе прямоугольник со сторонами a и b, причём $a > b$. Будем «выкладывать» в него квадраты со стороной b, пока это возможно. Когда останется полоска со сторонами b и $r = a \bmod b$, повторим процедуру для неё, и так далее. Когда выкладывание закончится точным квадратом — сторона этого последнего квадрата и есть $\gcd(a, b)$ (рис. 4.1).

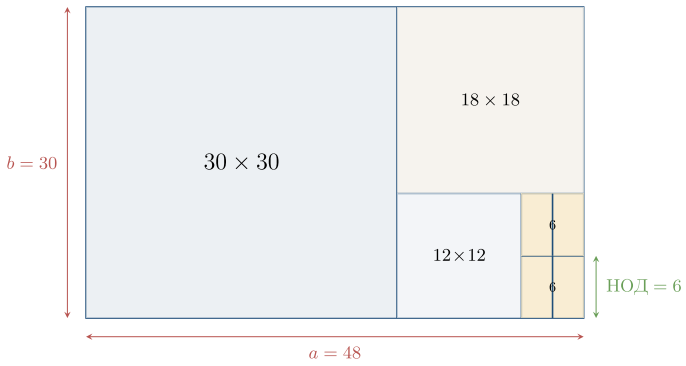

Рис. 4.1. Геометрическая иллюстрация алгоритма Евклида для пары $(48, 30)$. Прямоугольник заполняется квадратами: 30×30, затем 18×18, затем 12×12 и, наконец, два квадрата 6×6. Сторона последнего квадрата равна $\gcd(48, 30) = 6$.

Численно это соответствует последовательным делениям с остатком:

$$\begin{aligned} 48 &= 1 \cdot 30 + 18, \\ 30 &= 1 \cdot 18 + 12, \\ 18 &= 1 \cdot 12 + 6, \\ 12 &= 2 \cdot 6 + 0 \Rightarrow \gcd(48, 30) = 6. \end{aligned} \tag{4}$$

Почему это работает? Ключевое наблюдение: если $a = bq + r$, то $\gcd(a, b) = \gcd(b, r)$. Действительно, любое число, делящее a и b, делит и $r = a - bq$; обратно, любое число, делящее b и r, делит и $a = bq + r$. Значит, у пар (a, b) и (b, r) один и тот же набор общих делителей — и, следовательно, один и тот же gcd. На каждом шаге второй аргумент уменьшается; раньше или позже он станет нулём, и тогда первое число — искомый НОД.

> **!** Алгоритм: нахождение НОД (Евклид)
>
> **Вход:** натуральные числа a, b, $a \geq b > 0$.
>
> **Выход:** $\gcd(a, b)$.
>
> 1. Если $b = 0$, вернуть a.
> 2. Иначе вычислить $r = a \bmod b$ и повторить с парой (b, r).

На псевдокоде:

```
function gcd(a, b):
while b ≠ 0:
a, b := b, a mod b
return a
```

Скорость работы. Сколько шагов делает алгоритм? Можно доказать (теорема Ламе, 1844 г.): число делений в

алгоритме Евклида для $\gcd(a, b)$ не превосходит примерно $5 \log_{10} \min(a, b)$. Это значит, что для чисел в *сотни цифр* (а такие числа реально используются в криптографии) НОД находится за несколько сотен делений — доли секунды.

Расширенный алгоритм Евклида

Поразительный факт: Евклид нашёл не только gcd, но и способ *представить* его в виде «целочисленной линейной комбинации» исходных чисел.

⚠ Теорема 4.8. соотношение Безу

Для любых натуральных a, b существуют целые числа x, y (*не обязательно положительные*), такие что

$$a \cdot x + b \cdot y = \gcd(a, b). \tag{5}$$

Эти x, y называются **коэффициентами Безу**. Найти их позволяет **расширенный алгоритм Евклида**: при делениях с остатком мы заодно «отслеживаем», как выражается каждый текущий остаток через исходные a и b.

Пример: те же 48 и 30. Прокрутим цепочку делений «снизу вверх», выражая 6 (наш НОД) через всё бо'льшие числа.

$$\begin{aligned} 6 &= 18 - 1 \cdot 12 && \text{(из третьего деления)} \\ &= 18 - 1 \cdot (30 - 1 \cdot 18) = 2 \cdot 18 - 1 \cdot 30 && \text{(подставили } 12 = 30 - 18\text{)} \\ &= 2 \cdot (48 - 1 \cdot 30) - 1 \cdot 30 = 2 \cdot 48 - 3 \cdot 30 && \text{(подставили } 18 = 48 - 30\text{)} \end{aligned} \tag{6}$$

Получили: $2 \cdot 48 + (-3) \cdot 30 = 6$. Коэффициенты Безу: $x = 2$, $y = -3$.

💡 Пример 4.9

Проверим: $2 \cdot 48 - 3 \cdot 30 = 96 - 90 = 6 = \gcd(48, 30)$. Совпадает.

Удобно вести расчёт в таблице:

шаг	r	q	x	y
0	48	—	1	0
1	30	—	0	1
2	18	1	1	-1
3	12	1	-1	2

шаг	r	q	x	y
4	6	1	2	-3
5	0	2	—	—

В каждой строке выполняются равенства $r = 48x + 30y$. На предпоследней строке (где $r = \gcd = 6$) и считываются коэффициенты.

Решение линейных сравнений

Применим то, что получили, к решению уравнений в модулярной арифметике.

Обратный элемент по модулю. В обычной арифметике число, обратное к a, — это $1/a$: при умножении даёт единицу. По модулю m **обратным к a** называется такое целое a^{-1}, что

$$a \cdot a^{-1} \equiv 1 \pmod{m}. \tag{7}$$

Например, по модулю 7: $3 \cdot 5 = 15 = 2 \cdot 7 + 1 \equiv 1 \pmod 7$, значит, $3^{-1} \equiv 5 \pmod 7$.

⚠ Теорема 4.10

Обратный элемент к a по модулю m *существует* тогда и только тогда, когда $\gcd(a, m) = 1$. При этом он находится с помощью расширенного алгоритма Евклида.

Доказательство. Если $\gcd(a, m) = 1$, то по теореме 4.9 существуют целые x, y, такие что $ax + my = 1$. Это сравнение по модулю m принимает вид $ax \equiv 1 \pmod m$, то есть $x \equiv a^{-1} \pmod m$.

Обратно, если $a \cdot a^{-1} \equiv 1 \pmod m$, то $a \cdot a^{-1} - 1 = m \cdot k$ для некоторого k, откуда любой общий делитель a и m должен делить 1, — значит, $\gcd(a, m) = 1$.▫

💡 Пример 4.11

Найдём $11^{-1} \pmod{26}$. Применяем расширенный алгоритм Евклида к $(26, 11)$:

$$\begin{aligned} 26 &= 2 \cdot 11 + 4, \\ 11 &= 2 \cdot 4 + 3, \\ 4 &= 1 \cdot 3 + 1, \\ 3 &= 3 \cdot 1 + 0. \end{aligned} \tag{8}$$

$\gcd = 1$. Раскручиваем:

$$\begin{aligned} 1 &= 4 - 1 \cdot 3 = 4 - (11 - 2 \cdot 4) = 3 \cdot 4 - 11 \\ &= 3 \cdot (26 - 2 \cdot 11) - 11 = 3 \cdot 26 - 7 \cdot 11. \end{aligned} \tag{9}$$

Значит, $-7 \cdot 11 \equiv 1 \pmod{26}$, то есть $11^{-1} \equiv -7 \equiv 19 \pmod{26}$. Проверка: $11 \cdot 19 = 209 = 8 \cdot 26 + 1$.

Линейные сравнения. Уравнение вида

$$a \cdot x \equiv b \pmod{m} \tag{10}$$

называется **линейным сравнением**. Если $\gcd(a, m) = 1$, его решение находится «домножением на обратный»: $x \equiv a^{-1} \cdot b \pmod{m}$.

Быстрое возведение в степень

В криптографии нужно уметь вычислять выражения вида $a^n \bmod m$ для огромных n (порядка 2^{1000}). Если в лоб умножать a на себя n раз — это никогда не закончится: даже за миллиард умножений в секунду на это ушло бы больше времени, чем существует Вселенная.

Спасает идея **«разделяй и властвуй»**, известная также как **бинарное возведение в степень**. Идея проста: разделим показатель пополам:

$$a^n = \begin{cases} \left(a^{n/2}\right)^2, & n \text{ чётно}, \\ a \cdot \left(a^{(n-1)/2}\right)^2, & n \text{ нечётно}. \end{cases} \tag{11}$$

Каждое такое деление пополам уменьшает показатель примерно вдвое, поэтому всего нужно около $\log_2 n$ шагов вместо n (рис. 4.2).

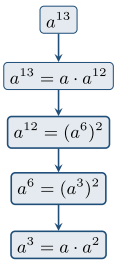

Рис. 4.2. Дерево бинарного возведения a^{13}: показатель уменьшается вдвое (с поправкой на чётность) на каждом шаге. Всего около $\log_2 13 \approx 4$ шагов.

Каждый «спуск» делит показатель примерно на 2, поэтому всего получается около $\log_2 n$ шагов. Для $n = 2^{1000}$ это *тысяча* умножений вместо 2^{1000}.

На псевдокоде:

```
function powmod(a, n, m):
result := 1
a := a mod m
while n > 0:
if n is odd:
result := (result * a) mod m
a := (a * a) mod m
n := n div 2
return result
```

Пример 4.12

Вычислим $7^{13} \bmod 23$. Запишем $13 = 1101_2$, то есть $13 = 8 + 4 + 1$. Значит, $7^{13} = 7^8 \cdot 7^4 \cdot 7^1$. Считаем последовательно по модулю 23:

$$7^1 = 7, \quad 7^2 = 49 \equiv 3, \quad 7^4 \equiv 3^2 = 9, \quad 7^8 \equiv 9^2 = 81 \equiv 81 - 3 \cdot 23 = 12.$$

Тогда $7^{13} \equiv 12 \cdot 9 \cdot 7 = 756 \equiv 756 - 32 \cdot 23 = 756 - 736 = 20 \pmod{23}$. Всего — около пяти умножений вместо тринадцати.

Та же идея в другом месте: числа Фибоначчи. Знакомая со школы последовательность Фибоначчи задаётся рекуррентно:

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+1} = F_n + F_{n-1}. \tag{13}$$

На первый взгляд, чтобы вычислить F_n, нужно сделать n сложений — линейное время. Однако и здесь работает «разделяй и властвуй»! Запишем рекуррентное соотношение в матричной форме:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}. \tag{14}$$

Обозначим матрицу как A. Применяя её n раз к вектору $(F_1, F_0)^\top = (1, 0)^\top$, получим:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{15}$$

Прямое доказательство — индукцией по n.

Важно

Тем самым задача сводится к возведению матрицы 2×2 в степень n. Но матрицы можно перемножать «делением пополам» точно так же, как числа! Сложность — $O(\log n)$ матричных умножений, то есть около $\log_2 n$ операций; вместо линейного числа шагов получается логарифмическое (рис. 4.3).

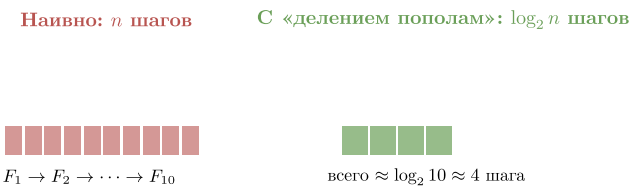

Рис. 4.3. Сравнение наивного и «разделяй и властвуй»-подхода к вычислению F_n.

Этот пример замечателен тем, что показывает: даже линейная на первый взгляд задача может «прятать» внутри себя структуру, ускоряющую решение экспоненциально.

Функция Эйлера и малая теорема Ферма

Чем дальше, тем интересней. Сейчас мы введём одну функцию, играющую ключевую роль в современной криптографии.

Определение 4.13

Функцией Эйлера $\varphi(n)$ называется количество натуральных чисел в отрезке $[1, n]$, взаимно простых с n.

Пример 4.14

$\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(6) = 2$ (а именно, числа 1 и 5), $\varphi(10) = 4$ (числа 1, 3, 7, 9).

Случай простого числа. Если p — простое число, то с p *не* взаимно прост лишь сам p среди чисел $1, 2, \ldots, p$. Значит,

$$\varphi(p) = p - 1 \quad \text{для простого} \quad p. \tag{16}$$

Случай произведения двух простых. Пусть $n = p \cdot q$, где p, q — различные простые. Какие числа из $[1, n]$ не взаимно

просты с n? Только кратные p или кратные q. Кратных p ровно q штук (это $p, 2p, \ldots, qp$); кратных q ровно p штук; единственное число, кратное обоим, — $pq = n$. По формуле включений-исключений:

$$\varphi(pq) = pq - p - q + 1 = (p-1)(q-1). \tag{17}$$

Это короткое наблюдение — сердце алгоритма RSA, который мы изучим в § 4.3.

Пример 4.15

$\varphi(15) = \varphi(3 \cdot 5) = 2 \cdot 4 = 8$. Действительно, взаимно простыми с 15 в отрезке $[1, 15]$ являются числа $\{1, 2, 4, 7, 8, 11, 13, 14\}$ — ровно 8 чисел.

В общем виде: для $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ выполняется $\varphi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$, но эта формула нам сегодня понадобится только в указанных двух частных случаях.

Малая теорема Ферма. Один из самых красивых результатов классической теории чисел.

Теорема 4.16. малая теорема Ферма, 1640 г.

Пусть p — простое число и a — целое, не делящееся на p. Тогда

$$a^{p-1} \equiv 1 \pmod{p}. \tag{18}$$

Равносильно: для любого целого a выполняется $a^p \equiv a \pmod{p}$.

Доказательство. Рассмотрим набор $\{a, 2a, 3a, \ldots, (p-1)a\}$ — $(p-1)$ ненулевых остатков, умноженных на a. Все они различны по модулю p: если $ia \equiv ja \pmod{p}$, то $a(i-j) \equiv 0 \pmod{p}$, а так как a не делится на p и $|\, i - j \,| < p$, получаем $i = j$.

Значит, набор $\{a, 2a, \ldots, (p-1)a\}$ — это в точности (с точностью до порядка) набор $\{1, 2, \ldots, p-1\}$ по модулю p. Перемножим всё:

$$a \cdot 2a \cdot 3a \cdots (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p}, \tag{19}$$

то есть $a^{p-1} \cdot (p-1)! \equiv (p-1)! \pmod{p}$. Поскольку $(p-1)!$ взаимно прост с p, его можно «сократить» — получим $a^{p-1} \equiv 1 \pmod{p}$. ▫

Обобщение — **теорема Эйлера**: $a^{\varphi(n)} \equiv 1 \pmod{n}$ для любого a, взаимно простого с n. Доказывается аналогично, только вместо чисел $1, \ldots, p-1$ берутся все числа из $[1, n]$, взаимно простые с n.

Это интересно

Пьер Ферма, по профессии юрист, посвящал математике вечера. В 1640 году он написал письмо своему другу с этим утверждением, заметив: «Я бы прислал доказательство, если бы не боялся, что оно слишком длинное». Первое известное полное доказательство опубликовал Леонард Эйлер — спустя сто лет.

Сколько вокруг простых чисел?

Прежде чем переходить к криптографии, ответим на важный практический вопрос: *часто ли встречаются простые числа?* Если простых чисел мало — ими нельзя будет полноценно пользоваться.

Теорема 4.17. Чебышёв – Адамар – Валле-Пуссен, асимптотический закон распределения простых чисел

Пусть $\pi(N)$ — количество простых чисел, не превосходящих N. Тогда

$$\pi(N) \sim \frac{N}{\ln N} \qquad \text{при} \quad N \to \infty. \tag{20}$$

В пересчёте на «вероятность»: если случайно взять натуральное число вокруг N, оно окажется простым примерно с шансом $1/\ln N$. Например, среди 1000-значных чисел простыми оказывается примерно одно из $\ln 10^{1000} = 1000 \ln 10 \approx 2300$. Это совсем немало (рис. 4.4).

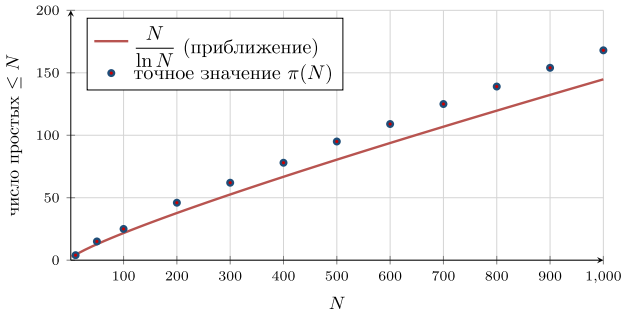

Рис. 4.4. Закон распределения простых чисел: количество простых чисел до N ведёт себя как $N/\ln N$.

Практический смысл: чтобы найти простое число длиной, скажем, 1024 бита для RSA-ключа, нужно перебрать в среднем порядка 700 случайных чисел и каждое проверить на простоту. И это нас приводит к следующему вопросу.

Проверка простоты и задача факторизации

Два «двойственных» вопроса:

Проверка простоты:
дано число n — простое оно или составное?

Факторизация:
дано составное n — разложить его на простые множители.

Может показаться, что эти задачи примерно одинаковой сложности. Но это не так: *они принципиально разной сложности*, и именно на этом контрасте основана вся современная криптография.

Тест Ферма. Если p простое, то по теореме 4.18 для любого a, не кратного p, выполняется $a^{p-1} \equiv 1 \pmod{p}$. Возьмём это как *пробный признак*: если для случайного a выполнено $a^{n-1} \not\equiv 1 \pmod{n}$, то n заведомо *не простое*. Если же $a^{n-1} \equiv 1 \pmod{n}$ — то n простое *с большой вероятностью* (но иногда бывают и составные числа, проходящие тест — так называемые **числа Кармайкла**).

Тест Миллера–Рабина. Усовершенствованный вариант теста Ферма. Он использует ту же идею, но дополнительно «отсеивает» числа Кармайкла. Принципиальные свойства:

- это **вероятностный** тест: при отрицательном ответе число точно составное; при положительном — простое с вероятностью ошибки не больше $1/4$ за один проход;
- делая k независимых проходов, мы уменьшаем вероятность ошибки до 4^{-k}. Уже при $k = 20$ это меньше, чем 10^{-12};
- работает за время $O(k \cdot \log^3 n)$ — то есть **полиномиально** от числа цифр.

Тест AKS (2002 г.). Манидра Агравал, Нираж Каял и Нитин Саксена — индийские математики — впервые предъявили *детерминированный* полиномиальный алгоритм проверки простоты. На практике он работает медленнее Миллера–Рабина, но важен как теоретическое достижение: показал, что класс задач, разрешимых за полиномиальное время, включает в себя проверку простоты.

А что с факторизацией? Тут картина совсем другая. Лучший известный сегодня алгоритм — *общий метод решета числового поля* — работает за время, растущее как

$$\exp\left(c \cdot \sqrt[3]{(\ln n)(\ln \ln n)^2}\right), \tag{21}$$

что по сути **субэкспоненциально**, но *не полиномиально*. Для 1024-битных n это означает миллиарды лет работы суперкомпьютера.

Важно

Колоссальный разрыв в сложности: *проверить, простое ли число* — легко (полиномиально), а *разложить число на простые множители* — невероятно трудно (субэкспоненциально). Именно этот разрыв и эксплуатируется в RSA и других криптосистемах. Безопасность интернета прямо опирается на то, что у человечества пока нет быстрого алгоритма факторизации.

А если появится квантовый компьютер? В 1994 году Питер Шор предложил квантовый алгоритм, факторизующий числа за полиномиальное время. Реальных квантовых компьютеров достаточного масштаба пока нет (на 2025 год удавалось факторизовать лишь крошечные числа в качестве демонстрации). Но если такие компьютеры будут построены — современный RSA рухнет, и потребуются принципиально новые криптографические схемы (это направление называется *постквантовая криптография*).

Это интересно

Открытие AKS-алгоритма стало сенсацией: один из руководителей — Нитин Саксена — был на тот момент аспирантом первого года. Результат опубликован в 2004 году в одном из самых престижных математических журналов — «Annals of Mathematics» — и был назван «алгоритмом тысячелетия» в области теории чисел.

Что мы получили

Подведём итог. На вопрос «зачем нам всё это в учебнике информатики» мы готовы ответить: только что мы построили *инструменты*, без которых не работает ни один защищённый сайт. А именно:

- **сравнения по модулю** — язык, на котором будем описывать шифрование;
- **расширенный алгоритм Евклида** — быстро ищет обратный элемент по модулю;
- **быстрое возведение в степень** — позволяет работать с гигантскими степенями;

- **малая теорема Ферма** и **функция Эйлера** — ключ к схеме RSA;
- **разрыв в сложности** между проверкой простоты и факторизацией — источник «безопасности».

В следующих параграфах мы соберём из этих кирпичиков работающие криптографические протоколы.

! Задачи для самостоятельной работы

1. Найдите $\gcd(231, 165)$ алгоритмом Евклида. Представьте НОД в виде $231x + 165y$.
2. Найдите $5^{-1} \pmod{37}$. Используйте расширенный алгоритм Евклида.
3. Вычислите $3^{100} \bmod 7$, пользуясь малой теоремой Ферма.
4. Сколько существует чисел в отрезке $[1, 100]$, взаимно простых с 100? (Подсказка: $100 = 2^2 \cdot 5^2$, но вы можете использовать формулу для $\varphi(p^2)$ или прямой подсчёт.)
5. Запишите быстрое возведение в степень в виде программы на знакомом вам языке программирования.
6. * Докажите, что для составного n всегда найдётся a из $[2, n-1]$, для которого $a^{n-1} \not\equiv 1 \pmod{n}$ — кроме чисел Кармайкла. Найдите наименьшее число Кармайкла.
7. * С помощью матричной формулы для чисел Фибоначчи вычислите F_{20}.