

Алгоритмы теории чисел

Рыжичкин Кирилл

Февраль 2024

Оглавление

Введение	3
1 Простейшие алгоритмы теории чисел	4
1. Простые числа	4
2. НОД и НОК двух чисел	6
3. НОД и НОК n чисел	8
4. Расширенный алгоритм Евклида	9

Введение

Мир алгоритмов теории чисел представляет собой захватывающий путь в глубины математического мира, где числа становятся не только абстрактными сущностями, но и основой для разработки эффективных алгоритмов. Теория чисел, одна из старейших областей математики, занимается изучением свойств целых чисел и их взаимосвязей. Вместе с тем, алгоритмы теории чисел играют критическую роль в современном мире, находя применение в широком спектре областей, включая криптографию, компьютерные науки, теорию игр и многое другое.

Мы начнем с основных понятий, таких как простые числа, делители, наибольший общий делитель и модульная арифметика, и постепенно углубимся в более сложные темы, такие как теорема Эйлера, китайская теорема об остатках, алгоритмы факторизации и многое другое.

Глава 1

Простейшие алгоритмы теории чисел

1. Простые числа

Определение. *Натуральное число p называется простым (англ. prime number), если $p > 1$ и p не имеет натуральных делителей, отличных от 1 и p .*

Определение. *Натуральное число $n > 1$ называется составным (англ. composite number), если n имеет по крайней мере один натуральный делитель, отличный от 1 и n .*

Согласно определениям, множество натуральных чисел разбивается на 3 подмножества:

1. Простые числа.
2. Составные числа.
3. Число 1, которое не причисляется ни к простым, ни к составным числам.

Отсюда сразу возникает идея реализации интуитивно понятного алгоритма проверки числа n на простоту: если $n = 1$, вернуть false, иначе пройтись по всем числам от 2 до $n - 1$, и если ни на одно из них число не поделится, то вернуть true. Соответственно сложность такого алгоритма будет $O(n)$.

```

bool isPrime(int n) {
    if (n == 1) {
        return false;
    }
    for (int i = 2; i < n; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

```

Утверждение. Пусть $N = a \times b$, причем $a \leq b$. Тогда $a \leq \sqrt{N} \leq b$

Доказательство. Если $a \leq b < \sqrt{N}$, то $ab \leq b^2 < N$, но $ab = N$. А если $\sqrt{N} < a \leq b$, то $N < a^2 \leq ab$, но $ab = N$. ■

Из этого следует, что если число N не делится ни на одно из чисел $2, 3, 4, \dots, \lfloor \sqrt{N} \rfloor$, то оно не делится и ни на одно из чисел $\lceil \sqrt{N} \rceil + 1, \dots, N - 2, N - 1$, так как если есть делитель больше корня (не равный N), то есть делитель и меньше корня (не равный 1). Поэтому в цикле `for` достаточно проверять числа не до N , а до корня. Соответственно сложность такого алгоритма будет уже $O(\sqrt{n})$:

```

bool isPrime(int n) {
    if (n == 1) {
        return false;
    }
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

```

Однако и этот алгоритм вовсе не оптимален и на практике применяется разве что на маленьких числах. Более оптимальные алгоритмы проверки на простоту будут рассмотрены в следующих главах.

2. НОД и НОК двух чисел

Определение. Наибольший общий делитель (НОД) двух натуральных чисел a и b это наибольшее натуральное число d , которое делит оба числа a и b без остатка. Обозначается как $d = \text{НОД}(a, b)$.

Утверждение. Для любых двух натуральных чисел $a > b$ верно следующее равенство:

$$\text{НОД}(a, b) = \text{НОД}(a - b, b).$$

Доказательство. $n = \text{НОД}(a, b)$ — наибольший среди всех общих делителей чисел a и b , тогда $a : n$ и $b : n$. Отсюда $a - b : n$, значит, n — делитель $a - b$ и b , он не превосходит наибольшего из всех общих делителей чисел $a - b$ и b , т.е. $n \leq m = \text{НОД}(a, b)$. Аналогично, рассуждая получаем, что $a - b : m$ и $b : m = \text{НОД}(a, b)$. Отсюда $a = (a - b) + b : m$ значит, m — делитель a и b , он не превосходит наибольшего из всех общих делителей чисел a и b , т.е. $m \leq n = \text{НОД}(a, b)$. Таким образом получаем $m \leq n \leq m$, т.е. $\text{НОД}(a, b) = n = m = \text{НОД}(a - b, b)$. ■

Ход алгоритма Евклида:

1. **Инициализация:** Начнем с двух целых чисел a и b , где $a \geq b$.
2. **Шаг 1:** Разделим a на b и найдем остаток от деления. Пусть остаток обозначается как r . То есть, $a = bq_1 + r$, где q_1 — частное, а r — остаток.
3. **Шаг 2:** Если остаток r равен нулю, то $\text{НОД}(a - b, b)$ равен b , и алгоритм завершается.
4. **Шаг 3:** Если остаток r не равен нулю, заменим a на b и b на r , и вернемся к шагу 1.
5. **Шаг 4:** Повторяем процесс, пока остаток r не станет равным нулю. Когда это произойдет, $\text{НОД}(a, b)$ будет равен последнему ненулевому остатку, который был получен на предыдущем шаге.
6. **Вывод результата:** Когда остаток становится равным нулю, последнее значение b будет искомым $\text{НОД}(a, b)$.

Алгоритм Евклида гарантирует, что за конечное число шагов мы получим $\text{НОД}(a, b)$, так как на каждом шаге остаток уменьшается и стремится к нулю.

```

int64_t NumberTheory::Gcd(int64_t a, int64_t b) {
    while (b != 0) {
        int64_t temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

```

Определение. Наименьшее общее кратное (НОК) двух натуральных чисел a и b это наименьшее натуральное число l , которое делится на оба числа a и b без остатка. Обозначается как $l = \text{НОК}(a, b)$.

Утверждение. $\forall a, b \in \mathbb{N}: \text{НОД}(a, b) \cdot \text{НОК}(a, b) = ab$.

Доказательство. Во время доказательства нам пригодится следующее простое свойство:

$$\min(n; m) + \max(n; m) = n + m.$$

Оно верно, т.к. $\min(n, m)$ совпадает с одним из чисел n или m , а $\max(n, m)$ с другим. Запишем каноническое разложение на простые множители чисел a и b :

$$a = p_1^{n_1} p_2^{n_2} \dots p_k^{n_k} \text{ и } b = p_1^{m_1} p_2^{m_2} \dots p_k^{m_k}$$

(возможно, что степени некоторых простых нулевые, т.е. простое число входит в разложение одного из чисел a или b на простые множители, но не входит в разложение второго). Запишем теперь разложение НОД (a, b) и НОК (a, b) :

$$\begin{aligned} \text{НОД}(a, b) &= p_1^{\min\{n_1, m_1\}} p_2^{\min\{n_2, m_2\}} \dots p_k^{\min\{n_k, m_k\}} \\ \text{НОК}(a, b) &= p_1^{\max\{n_1, m_1\}} p_2^{\max\{n_2, m_2\}} \dots p_k^{\max\{n_k, m_k\}} \end{aligned}$$

То есть

$$\text{НОД}(a, b) \cdot \text{НОК}(a, b) = \prod_{i=1}^k p_i^{\min\{n_i, m_i\} + \max\{n_i, m_i\}} =$$

в силу свойства, указанного в самом начале доказательства, получаем

$$= \prod_{i=1}^k p_i^{n_i + m_i} = \prod_{i=1}^k p_i^{n_i} \prod_{i=1}^k p_i^{m_i} = ab.$$

■

```
int64_t NumberTheory::Lcm(int64_t a, int64_t b) {  
    return a * b / Gcd(a, b);  
}
```

3. НОД и НОК n чисел

Алгоритм Евклида для нескольких чисел использует основное свойство наибольшего общего делителя: если d делит каждое из чисел a_1, a_2, \dots, a_n , то d также делит и их НОД.

Ход алгоритма Евклида для n чисел:

1. **Инициализация:** Пусть у нас есть набор чисел a_1, a_2, \dots, a_n .
2. **Нахождение НОДа для первых двух чисел:**

$$d_1 = \text{НОД}(a_1, a_2)$$

3. **Обновление НОДа для следующих чисел:**

Для каждого $i = 3, 4, \dots, n$:

$$d_i = \text{НОД}(d_{i-1}, a_i)$$

4. **Конечный результат:**

Наибольший общий делитель всех чисел: $\text{НОД}(a_1, a_2, \dots, a_n) = d_n$.

Поскольку d_1 является НОДом a_1 и a_2 , он является делителем обоих этих чисел. Следовательно, d_1 делит оба числа нацело.

Когда мы переходим к следующему числу a_i , мы находим НОД между текущим НОДом d_{i-1} и a_i . Это значит, что d_{i-1} делит a_i и d_{i-1} нацело. Поскольку d_{i-1} также является НОДом всех предыдущих чисел, он делит все предыдущие числа нацело. Таким образом, d_{i-1} делит a_i и все предыдущие числа нацело.

Стало быть, после обработки всех чисел мы получаем d_n , который является НОДом всех чисел a_1, a_2, \dots, a_n .


```
int64_t NumberTheory::Gcd(std::vector<int64_t> numbers) {  
    int64_t res = numbers[0];  
    for (int i = 1; i < numbers.size(); ++i) {  
        res = Gcd(res, numbers[i]);  
    }  
    return res;  
}
```

Абсолютно аналогично получается алгоритм для нахождения НОКа нескольких чисел:

```
int64_t NumberTheory::Lcm(std::vector<int64_t> numbers) {  
    int64_t res = numbers[0];  
    for (int i = 1; i < numbers.size(); ++i) {  
        res = Lcm(res, numbers[i]);  
    }  
    return res;  
}
```

4. Расширенный алгоритм Евклида