

Математичка, ты довольна?

Часть 2.

Сапожников Денис

Contents

1	Алгоритм Евклида	2
1.1	Определения	2
1.2	Медленный алгоритм	2
1.3	Быстрый алгоритм	3
1.4	Время работы	3
1.5	Упражнения	3
2	Диофантовы уравнения и расширенный алгоритм Евклида	4
2.1	Одно решение $ax + by = gcd(a, b)$	4
2.2	Одно решение $ax + by = c$	4
2.3	Все решения $ax + by = c$	5
2.4	Возвращаемся к обратным элементам по модулю	5
3	Решето Эратосфера	6
3.1	Задача	6
3.2	Самый простой подход за $O(n\sqrt{n})$	6
3.3	Решение за $O(n \log n)$	6
3.4	Решение за $O(n \log \log n)$	6
3.5	Решение за $O(n)^*$	7
4	Китайская теорема об остатках (КТО)*	8

1 Алгоритм Евклида

1.1 Определения

Определение (НОД). Наибольшим общим делителем (НОДом) двух чисел a и b называется

такое число g , что
$$\begin{cases} a : g \\ b : g \\ g - \text{максимальное из возможных} \end{cases}$$

Обозначения: $\text{НОД}(a, b)$, или просто (a, b) .

Лемма 1. Пусть $(a, b) = g$. Пусть d — какой-то (не обязательно наибольший) общий делитель a и b . Тогда $g : d$.

Proof. Через разложение на простые множители по ОТА. □

Определение (НОК). Наименьшим общим кратным (НОКом) двух чисел a и b называется

такое число l , что
$$\begin{cases} l : a \\ l : b \\ l - \text{минимальное из возможных} \end{cases}$$

Обозначения: $\text{НОК}(a, b)$, или просто $[a, b]$.

Лемма 2. $(a, b) \cdot [a, b] = a \cdot b$

Proof. Через разложение на простые множители по ОТА. □

Данная лемма позволяет находить НОК по НОДу, таким образом надо научиться искать лишь НОД.

1.2 Медленный алгоритм

Лемма 3. Пусть $a \geq b$, тогда $(a, b) = (a - b, b)$.

Proof. Пусть $(a, b) = g_1, (a - b, b) = g_2$.

Докажем, что $g_2 : g_1$. Если $(a, b) = g_1$, то по определению НОДа: $a : g_1, b : g_1$. Значит и $a - b : g_1$. То есть получили, что $a - b$ и $b : g_1$. По лемме 1 $g_2 : g_1$.

Теперь докажем, что $g_1 : g_2$. Если $(a - b, b) = g_2$, то b и $a - b : g_2$. Значит и $a : g_2$. Получили: a и $b : g_2$. Значит по лемме 1 $g_1 : g_2$.

Итого: $g_1 : g_2$ и $g_2 : g_1$. Значит, очевидно, $g_1 = g_2$ что и т.д. □

Лемма 3 позволяет легко находить НОД двух чисел без разложения на простые множители.

```
1 int gcd(int a, int b) {
2   if (a == 0 || b == 0)
3     return a + b;
4   if (a >= b)
5     return gcd(a - b, b);
6   else
7     return gcd(a, b - a);
8 }
```

По сути, мы уже доказали, что алгоритм корректный, но, увы, он долго работает, например, на тесте $(10^9, 1)$. На данном тесте будет выполняться 10^9 преобразований $(a, 1) = (a - 1, 1) = \dots (1, 1) = (0, 1) = 1$.

1.3 Быстрый алгоритм

Идея по ускорению: заметим, что из $(a, b) = (a - b, b)$ следует, что $(a, b) = (a \% b, b)$, так как операция взятия по модулю эквивалентна большому количеству вычитаний b .

Теперь алгоритм имеет вид

```
1 int gcd(int a, int b) {
2   if (a == 0 || b == 0)
3     return a + b;
4   return gcd(b, a % b);
5 }
```

Более того, начиная с C++17 этот алгоритм есть в стандартной библиотеке `numeric` и называется `gcd`.

1.4 Время работы

Сейчас будет трюк. Следите за руками.

Пусть $a \geq b$. Тогда $0 \leq a \% b < b$. Что эквивалентно тому, что остатки от деления a на b лежат в интервале $\left[-\frac{b}{2}; \frac{b}{2}\right]$. Тогда после каждой итерации НОДа модулю одного из чисел уменьшается хотя бы в 2 раза.

Таким образом время работы алгоритма будет $O(\log n)$.

1.5 Упражнения

Задача 4. $(a_1, a_2, \dots, a_n) = (a_1, (a_2, a_3, \dots, a_n))$

Задача 5. Докажите, что простых чисел бесконечно много.

Решение. Возможно, стоит посмотреть на числа вида $n! - 1$.

Задача 6. Пусть $x = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$. Найдите количество делителей числа.

Решение. $\alpha(x) = (\alpha_1 + 1)(\alpha_2 + 1) \dots (\alpha_k + 1)$

Задача 7. Докажите, что число $\underbrace{111 \dots 1}_n$ для какого-то n делится на 123456789.

Решение. Посмотрим на остатки от деления первых $123456789+1$ чисел вида $\underbrace{111 \dots 1}_n$. Различных остатков не больше 123456789. Значит по принципу Дирихле найдутся два одинаковых остатка, пусть это будут $\underbrace{111 \dots 1}_k$ и $\underbrace{111 \dots 1}_m$, где $k > m$.

Посмотрим на число $\underbrace{111 \dots 1}_k - \underbrace{111 \dots 1}_m$. Оно делится на 123456789 и имеет вид $\underbrace{111 \dots 1}_{k-m} \underbrace{000 \dots 0}_m$. Поделим его на 10^m , результат будет иметь вид $\underbrace{111 \dots 1}_{k-m}$ и всё ещё делиться на 123456789, так как 10 и 123456789 были взаимнопросты.

Задача 8. Научитесь вычислять $\left\lceil \frac{a}{b} \right\rceil$ без `if`.

Решение.

```
1 (a + b - 1) / b;
```

2 Диофантовы уравнения и расширенный алгоритм Евклида

2.1 Одно решение $ax + by = \gcd(a, b)$

Задача. Найти одно решение уравнения $ax + by = g$, где g — это $\text{НОД}(a, b)$.

Пусть (x_0, y_0) — это решение уравнения $(b \% a)x + ay = g$. Научимся находить решение исходного уравнения.

По определению остатка от деления: $b \bmod a = b - \lfloor \frac{b}{a} \rfloor a$

Значит: $g = (b \bmod a)x_1 + ay_1 = (b - \lfloor \frac{b}{a} \rfloor a)x_1 + ay_1 = bx_1 + a(y_1 - \lfloor \frac{b}{a} \rfloor x_1)$

В итоге формулы для пересчёта следующие:

$$\begin{cases} x_0 = y_1 - \lfloor \frac{b}{a} \rfloor x_1 \\ y_0 = x_1 \end{cases}$$

```
1 int gcd(int a, int b, int &x, int &y) {
2   if (a == 0) {
3     x = 0, y = 1;
4     return a + b;
5   }
6   int x1, y1;
7   int g = gcd(b % a, a, x1, y1);
8   x = y1 - (b / a) * x1;
9   y = x1;
10  return g;
11 }
```

2.2 Одно решение $ax + by = c$

А теперь давайте решим уравнение $ax + by = c$ для произвольного c .

Если $c : \gcd(a, b)$, то всё просто: знаем решение уравнения

$$ax_0 + by_0 = g$$

Домножим обе части на $\frac{c}{g}$

$$a \left(\frac{c}{g} x_0 \right) + b \left(\frac{c}{g} y_0 \right) = c$$

То есть

$$\begin{cases} x = \frac{c}{g} x_0 \\ y = \frac{c}{g} y_0 \end{cases}$$

Если же $c \not\vdots \gcd(a, b)$, то решений, очевидно, нет, так как

2.3 Все решения $ax + by = c$

Лемма. Пусть (x_0, y_0) — это решение уравнения $ax + by = c$. Тогда все решения имеют вид:

$$\begin{cases} x = x_0 + \frac{b}{g}t \\ y = y_0 - \frac{a}{g}t \end{cases}, t \in \mathbb{Z}$$

Proof. Рассмотрим ещё одно решение (x_1, y_1) уравнения $ax + by = c$ и приравняем правые части:

$$\begin{aligned} ax_0 + by_0 &= ax_1 + by_1 \\ a(x_0 - x_1) &= b(y_1 - y_0) \\ \left(\frac{a}{g}\right)(x_0 - x_1) &= \left(\frac{b}{g}\right)(y_1 - y_0) \\ a'(x_0 - x_1) &= b'(y_1 - y_0) \end{aligned}$$

Левая часть делится на a' , значит и правая часть тоже должна делиться на a' . Так как мы сократили обе части на (a, b) , то $(a', b') = 1$, значит $(y_1 - y_0) : a'$. То есть $y_1 = y_0 + a' \cdot t = y_0 - \frac{a}{g} \cdot t$ для $t \in \mathbb{Z}$. В пару к такому y_1 подходит единственное решение $x_1 = x_0 + \frac{b}{g} \cdot t$. \square

2.4 Возвращаемся к обратным элементам по модулю

$$\begin{aligned} ab &\equiv 1 \pmod{n} \\ ab + kn &= 1 \end{aligned}$$

А это просто диофантово уравнение, решив которое мы найдем искомое a .

Итоговая сложность алгоритма $O(\log n)$

При этом мы решили для всех случаев, когда $\gcd(a, n) = 1$, а для других доказали, что решений нет.

3 Решето Эратосфера

3.1 Задача

Хотим найти все простые числа от 2 до n .

3.2 Самый простой подход за $O(n\sqrt{n})$

Заметим, что все имеет смысл перебирать делители лишь до \sqrt{x} , так как если есть делитель больше корня - d , то будет и делитель меньше корня: $\frac{x}{d}$. Поэтому код будет такой:

```
1 vector<bool> is_prime(n, true);
2 for (int i = 2; i <= n; i++)
3     for (int j = 2; j * j <= i; j++)
4         if (i % j == 0)
5             is_prime[i] = false;
```

3.3 Решение за $O(n \log n)$

Пусть если число простое, то переберём все числа, которые делятся на него, то есть $2p, 3p, 4p, \dots$ и пометим их как составные. Ну и действительно так мы пометим все числа составными, когда в качестве p мы возьмем любой простой делитель составного числа.

```
1 vector<bool> is_prime(n, true);
2 for (int i = 2; i <= n; i++)
3     if (is_prime[i])
4         for (int j = 2 * i; j <= n; j += i)
5             is_prime[j] = false;
```

Время работы. Для каждого простого числа p внутренний фор сделает $\frac{n}{p}$ итераций. Значит всего алгоритм сделает не больше чем $\sum_{i=1}^n \frac{n}{i} = O(n \log n)$ ¹, что и т.д.

3.4 Решение за $O(n \log \log n)$

Заметим факт, что нам необходимо пометать все простые делители, начиная не с $2i$, а с i^2 по идее из пункта 1. Таким образом нужно лишь заменить строку 4 и поменять условия для строки 2, чтобы не возникало переполнений:

```
1 vector<bool> is_prime(n, true);
2 for (int i = 2; i * i <= n; i++)
3     if (is_prime[i])
4         for (int j = i * i; j <= n; j += i)
5             is_prime[j] = false;
```

Почему это работает за $O(n \log \log n)$? Для доказательства используется факт, что простых чисел от 1 до n порядка $\frac{n}{\ln n}$ и они распределены примерно равномерно, а далее считается интеграл, более подробно можно прочитать на e-шахх.

¹В целом, это известный факт, но доказывать мы его, конечно же, не будем

3.5 Решение за $O(n)^*$

Вместо того, чтобы считать только пометку, простое ли было число, но и минимальное простое, на которое оно делится - p_i . Ещё будем поддерживать массив простых чисел от 2 до текущего i .

Пусть у нас есть текущее i . Обновим массив p с помощью i : пройдём по всем числам вида $x_j = p_j \cdot i$ (пока $p_j \leq p_i$) и отметим для них минимальное простое как p_j .

Итого код получается очень простым и лаконичным.

```
1  vector<int> pr;  
2  vector<int> p(n, 0);  
3  for (int i = 2; i < n; i++) {  
4      if (p[i] == 0) {  
5          p[i] = i;  
6          pr.push_back(i);  
7      }  
8      for (int j = 0; j < pr.size() && pr[j] <= pr[i] && i * pr[j] < n; ++j)  
9          pr[i * pr[j]] = pr[j];  
10 }
```

Рассмотрим произвольное число x . Из того, что оно единственным образом представляется в виде $x = p(x) \cdot y$, где $p(x)$ - минимальное простое число, на которое делится x следует, что алгоритм посетит каждое число ровно один раз. Значит время работы $O(n)$

4 Китайская теорема об остатках (КТО)*

Пусть у нас есть система сравнений:

$$\begin{cases} a \equiv x_1 \pmod{p_1} \\ a \equiv x_2 \pmod{p_2} \\ \dots \\ a \equiv x_n \pmod{p_n} \end{cases}$$

И ещё все модули попарно взаимнопросты, то есть $\forall i, j : \gcd(p_i, p_j) = 1$

Утверждения про такую систему:

- Каждому числу из отрезка $[0; p_1 p_2 \dots p_n - 1]$ соответствует единственный набор переменных (x_1, x_2, \dots, x_n) , удовлетворяющий системе уравнений.

Будем обозначать это как $a \Leftrightarrow (x_1, x_2, \dots, x_n)$

- Любой из операций $+$, $-$, $*$ соответствует ровно такая же над набором чисел $\{x_i\}$, то есть, если $a \Leftrightarrow (x_1, x_2, \dots, x_n)$ и $b \Leftrightarrow (y_1, y_2, \dots, y_n)$, тогда

$$a + b \Leftrightarrow (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$$

$$ab \Leftrightarrow (x_1 y_1, x_2 y_2, \dots, x_n y_n)$$

Понятно, как, зная число, получить набор $\{x_i\}$. Но как это сделать в обратную сторону?

Обозначим за $r_{i,j}$ обратный элемент к p_i по модулю p_j

Будем находить наше число a в таком виде:

$$a = y_1 + y_2 p_1 + y_3 p_1 p_2 + \dots + y_n p_1 \dots p_n$$

Тогда из 1-го уравнения знаем, что

$$y_1 = x_1$$

Из второго:

$$x_2 = y_1 + y_2 p_1$$

$$x_2 - y_1 = y_2 p_1$$

$$(x_2 - x_1) r_{1,2} = x_2$$

Ровно так же можно посчитать любое x_k алгоритм, очень похожим на схему Горнера за $O(k)$, значит восстановление всех x_i и искомого a - $O(n^2)$.

```
1  int a = 0, curp = 1;
2  for (int i = 0; i < n; ++i) {
3      y[i] = x[i];
4      for (int j = 0; j < i; ++j) {
5          y[i] = r[j][i] * (y[i] - y[j]);
6
7          y[i] = y[i] % p[i];
8          if (y[i] < 0)
9              y[i] += p[i];
10     }
11
12     a += curp * y[i];
13     curp *= p[i];
14 }
```

Но проблема в том, что встроенные типы очень быстро переполняются, поэтому обычно придётся написать длинную арифметику.