

Лекция 4 от 20.09.2016. Простейшие теоретико-числовые алгоритмы

Числовые алгоритмы играют огромную роль в криптографии, фактически вся криптография держится на том, что не придуман до сих пор алгоритм, который умеет факторизовать числа за полиномиальное время от размера числа.

Алгоритм Евклида

Начнём, пожалуй, с одного из самых известных алгоритмов нахождения наибольшего общего делителя, а именно — алгоритм Евклида и его расширенную версию.

Algorithm 1 Алгоритм Евклида.

```

1: function gcd(int  $a$ , int  $b$ )
2:   if  $b = 0$  then
3:     return  $a$ ;
4:   else
5:     return gcd( $b, a \bmod b$ );
```

Практически очевидно, что данный алгоритм возвращает нужное нам число. Вспомните курс дискретной математики или выпишите на бумаге то, что делает данный алгоритм.

Асимптотика такого алгоритма $O(\log n)$ (где n — максимальная длина числа) — легко проверить, что каждое число уменьшается хотя бы в 2 раза за 2 шага алгоритма.

Расширенный алгоритм Евклида

Пусть даны числа a, b, c , мы хотим найти хотя бы одну пару решений x, y таких, что $ax + by = c$. Понятно, что $\gcd(a, b) \mid c$, поэтому если это условие не выполняется, то найти решение мы не сможем. Пусть $c = k \gcd(a, b)$. Сейчас мы предъявим хотя бы одну пару чисел x, y , что $ax + by = \gcd(a, b)$ — после этого мы просто домножим x, y на k и получим, что сможем представить c в таком виде.

Algorithm 2 Расширенный алгоритм Евклида.

```

1: function EXTENDED_gcd(int  $a$ , int  $b$ )      ▷ — возвращаем тройку чисел  $(x, y, \gcd(a, b))$ .
2:   if  $b = 0$  then
3:     return  $(1, 0, a)$ ;
4:    $(x', y', d) \leftarrow \text{EXTENDED\_gcd}(b, a \bmod b)$ 
5:   return  $(y', x' - \lfloor \frac{a}{b} \rfloor y', d)$ 
```

Лемма 1. Для произвольных неотрицательных чисел a и b ($a \geq b$) расширенный алгоритм Евклида возвращает целые числа x, y, d , для которых $\gcd(a, b) = d = ax + by$.

Доказательство. Если не рассматривать x, y в алгоритме, то такой алгоритм полностью повторяет обычный алгоритм Евклида. Поэтому алгоритм 3-им параметром действительно вычислит $\gcd(a, b)$.

Про корректность x, y будет вести индукцию по b . Если $b = 0$, тогда мы действительно вернём верное значение. Шаг индукции: заметим, что алгоритм находит $\gcd(a, b)$, произведя рекурсивный вызов для $(b, a \bmod b)$. Поскольку $(a \bmod b) < b$, мы можем воспользоваться предположением индукции и заключить, что для возвращаемых рекурсивным вызовом чисел x', y' выполняется равенство:

$$\gcd(b, a \bmod b) = bx' + (a \bmod b)y'$$

Понятно, что $a \bmod b = a - \lfloor \frac{a}{b} \rfloor b$, поэтому

$$\begin{aligned} d = \gcd(a, b) &= \gcd(b, a \bmod b) = bx' + (a \bmod b)y' = \\ &= bx' + \left(a - \left\lfloor \frac{a}{b} \right\rfloor b\right)y' = ay' + b\left(x' - \left\lfloor \frac{a}{b} \right\rfloor y'\right) \end{aligned}$$

□

Пример 1. Мы умеем с помощью расширенного алгоритма Евклида вычислять обратные остатки по простому модулю (в поле \mathbb{F}_p). Действительно, если $(a, p) = 1$ то существуют x, y , что $ax + py = 1$, а значит в поле \mathbb{F}_p — $ax = 1$, откуда $x = a^{-1}$.

Алгоритм быстрого возведения в степень по модулю

Хотим вычислить $a^b \bmod p$. Основная идея в том, чтобы разложить b в двоичную систему и вычислять только $a^{2^i} \bmod p$. Здесь будем предполагать, что операции с числами выполняются достаточно быстро. Приведём ниже псевдокод такого алгоритма:

Algorithm 3 Алгоритм быстрого возведения в степень.

```

1: function FAST_POW(int  $a$ , int  $b$ , int  $p$ )                                ▷ — возвращаем  $a^b \bmod p$ .
2:   if  $b = 0$  then
3:     return 1;
4:   if  $b \bmod 2 = 1$  then
5:     return FAST_POW( $a, b - 1, p$ ) ·  $a \bmod p$ 
6:   else
7:      $c \leftarrow$  FAST_POW( $a, b/2, p$ )
8:     return  $c^2 \bmod p$ 

```

Корректность этого алгоритма следует из того, что $a^b = a^{b-1} \cdot a$ для нечетных b и

$a^b = a^{b/2} \cdot a^{b/2}$ для четных b . Также мы здесь неявно пользуемся индукцией по b , в которой корректно возвращается база при $b = 0$.

От каждого числа b , если оно четно, мы запускаем наш алгоритм от $b/2$, а если оно нечетно, то от $b - 1$, откуда получаем, что количество действий, совершенным нашим алгоритмом будет не более, чем $2 \log b = O(\log b)$.

Замечание 1. На самом деле быстрое возведение в степень работает на всех ассоциативных операциях. Например, если вы хотите вычислить A^n , где A — квадратная матрица, то это можно сделать тем же самым алгоритмом за $O(T(m) \log n)$, где $T(m)$ асимптотика перемножения матриц $m \times m$.

Китайская теорема об остатках и её вычисление

Китайская теорема об остатках звучит так — пусть даны попарно взаимно простые модули и числа r_1, \dots, r_n . Тогда существует единственное с точностью по модулю $a_1 \dots a_n$ решение такой системы:

$$\begin{cases} x \equiv r_1 \pmod{a_1} \\ x \equiv r_2 \pmod{a_2} \\ \vdots \\ x \equiv r_n \pmod{a_n} \end{cases}$$

Доказательство. Докажем и предьявим сразу алгоритм вычисления за $O(n \log \max(a_1, \dots, a_n))$.

Пусть $x = \sum_{i=1}^n r_i M_i M_i^{-1}$, где $M_i = \frac{a_1 \dots a_n}{a_i}$, M_i^{-1} это обратное к M_i по модулю a_i (такое всегда найдётся из попарной взаимной простоты). Прошу заметить, что такое число мы можем вычислить за $O(n \log \max(a_1, \dots, a_n))$ (см. пример в расширенном алгоритме Евклида).

Докажем, что это число подходит по любому модулю a_i .

$$x \equiv \sum_{j=1}^n r_j M_j M_j^{-1} \equiv r_i M_i M_i^{-1} \equiv r_i \pmod{a_i}$$

Второе равенство следует из того, что $a_i \mid M_j$ при $j \neq i$ (из построения).

Докажем единственность решения по модулю. Пусть x, x' — различные решения данной системы, тогда $0 < |x - x'| < a_1 \dots a_n$ и $|x - x'|$ делится на $a_1 \dots a_n$, что невозможно, так как ни одно положительное число до $a_1 \dots a_n$ не делится на $a_1 \dots a_n$. \square

Решето Эратосфена

Решето Эратосфена — это один из первых алгоритмов в истории человечества. Он позволяет найти все простые числа на отрезке от $[1; n]$ за $O(n \log \log n)$, а разложить все числа на простые множители за $O(n \log n)$

В первом случае у нас задача состоит в том, чтобы вернуть 1, если число простое и 0, если непростое.

Предьявим псевдокод такого алгоритма:

Лемма 2. *Алгоритм `Sieve_of_Eratosthenes` корректно оставит все простые числа.*

Доказательство. Докажем по индукции по n . База $n = 2$ очевидна.

Переход $n \rightarrow n + 1$. Заметим, что наш алгоритм и корректно завершит для n чисел, потому что мы только расширяем область рассматриваемых чисел.

Если $n + 1$ составное, тогда $n + 1 = p \cdot t$ для какого-то простого $p < n + 1$. По предположению индукции мы рассмотрим простое число p правильно, то есть удалим из массива все числа, которые кратны p , а значит и $n + 1$ мы правильно уберём.

Если $n + 1$ простое, то если мы его убрали на каком-то шаге, то оно делилось на то простое, которые мы рассматривали до этого, но это противоречит определению простых чисел. \square

Algorithm 4 Решето Эратосфена.

```

1: function SIEVE_OF_ERATOSTHENES(int  $n$ )           ▷ найти — массив  $prime_i$ , означающий
   характеристическую функцию простых чисел от 1 до  $n$ .
2:   for  $i \leftarrow 1$  to  $n$  do
3:      $prime_i \leftarrow true$ 
4:    $prime_1 \leftarrow false$ 
5:   for  $i \leftarrow 2$  to  $n$  do
6:     if  $prime_i = true$  then
7:        $j \leftarrow i$ 
8:       while  $j \leq n$  do
9:          $prime_j \leftarrow false$ 
10:         $j \leftarrow j + i$ 

```

Заметим, что алгоритм будет выполняться за время

$$\sum_{\substack{p \leq n, \\ p - \text{простое}}} \frac{n}{p}$$

Потому что для каждого простого числа мы рассматриваем в таблице все числа, кратные p . Можно оценить очень грубо и получим, что

$$\sum_{\substack{p \leq n, \\ p - \text{простое}}} \frac{n}{p} \leq \sum_{i=1}^n \frac{n}{i} \approx n \ln n + o(n) = O(n \log n)$$

Но используя свойства ряда $\sum_{\substack{p \leq n, \\ p - \text{простое}}} \frac{n}{p} \approx n \ln \ln n + o(n)$, следует, что алгоритм работает за

$O(n \log \log n)$, но факт про асимптотику этого ряда мы оставим без доказательства.

Если теперь первый раз, приходя в составное число в алгоритме, хранить его наименьший простой делитель, то рекурсивно мы можем разложить число на простые множители. Всего количество простых делителей у числа не может превышать $O(\log n)$ (так как самый наименьший простой делитель это двойка), поэтому разложение на множители будет выполняться за $O(n \log n)$.

Решето Эйлера

Составим двусвязный список из чисел от 2 до n , а также ещё массив длиной n с указателями на каждый элемент.

Будем идти итеративно: первый непросмотренный номер в списке берётся как простое число, и определяются все произведения с последующими элементами в списке (само на себя тоже умножим), пока не выйдем в произведении за пределы n . После этого удаляются все числа, которые мы вычислили (смотрим в массив указателей и удаляем по указателю за $O(1)$) и повторяем процедуру.

Лемма 3. После k шагов алгоритма останется первых k простых чисел в начале и в списке будут только числа взаимно простые с первыми k .

Доказательство. База при $k = 1$ очевидна. Просто убираем все четные числа.

Переход $k \rightarrow k + 1$.

Докажем, что следующим нерассмотренным элементом списка мы возьмём p_{k+1} . Действительно, простые числа мы не выкидываем, а значит следующим шагом после p_k мы возьмём число, не большее p_{k+1} , но по предположению индукции все числа от (p_k, p_{k+1}) были убраны, так как они составные и содержат в разложении только простые, меньшие p_{k+1} .

Предположим, что после ещё одного шага алгоритма у нас осталось число, кратное p_{k+1} (и большее p_{k+1}) (все числа, делящиеся на предыдущие простые до этого были убраны).

Тогда пусть это будет $m = p_{k+1} \cdot a$, $a > 1$. Если a содержит в разложении на простые хотя бы одно число, меньшее p_{k+1} , то получим противоречие, так как все числа не взаимно простые с p_1, \dots, p_k по предположению индукции были убраны.

Значит a содержит в разложении на простые числа, не меньшие p_{k+1} , а значит $a \geq p_{k+1}$ и это число ещё было в списке, значит мы это число уберем, противоречие. \square

Если мы вдруг на шаге алгоритма получили в умножении число, которое мы уже убрали, то значит у этого числа есть меньший простой делитель, чем p_k , но по доказанной лемме у нас все такие числа к k -ому шагу были убраны. Значит каждое составное число мы рассмотрим ровно 1 раз и ровно 1 раз уберем за $O(1)$.

Также по лемме получаем, что в начале списка останутся только простые числа.

Простые числа мы тоже рассматриваем по 1 разу в нашем алгоритме, значит общая сложность решета Эйлера будет $O(n)$.

Наивная факторизация числа за $O(\sqrt{n})$

На данный момент не существует алгоритма факторизации числа за полином от размера числа, а не от значения. Здесь мы рассмотрим наивный алгоритм факторизации числа. На следующей лекции рассмотрим ρ -метод Полларда, который работает за $O(\sqrt[4]{n})$.

Пусть $n' = n$. Будем перебирать от 2 до $\lceil \sqrt{n'} \rceil$ числа и пока текущее n делится на данное число, делим n на это число.

Легко показать, что делим мы только на простые числа (иначе мы поделили бы на меньшее простое несколькими шагами раньше).

В конце n будет либо 1 (тогда факторизация удалась), либо простым. Составным оно не может быть, иначе $n = ab$, $a, b > 1$ и $a, b > \lceil \sqrt{n'} \rceil$, так как на все числа, меньшие корня, мы поделили.

Осталось оценить, сколько операций раз мы обращаемся к циклу *while*. В нём мы делаем суммарно не более, чем $O(\log n + \sqrt{n})$ действий, так как сумма степеней при разложении числа не более, чем $O(\log n)$ (см. выше). Ну а также обращаемся по 1 разу каждый шаг внешнего цикла.