* [Теоретико-числовые алгоритмы на языке C++](https://foxford.ru/lessons/32268/conspects/1)
* [Операции с целыми числами в кольце вычетов на языке C++](https://foxford.ru/lessons/32268/conspects/2)

**Теоретико-числовые алгоритмы на языке C++**

**Алгоритм Евклида**

Алгоритм Евклида находит наибольший общий делитель двух данных целых неотрицательных чисел. Сложность алгоритма - O(log⁡n).

Рекурсивная реализация алгоритма Евклида:

**int gcd(int a, int b)**

**{**

**if (b == 0)**

**return a;**

**else**

**return gcd(b, a % b);**

**}**

Нерекурсивная реализация алгоритма Евклида:

**int gcd(int a, int b)**

**{**

**while (b > 0)**

**{**

**int t = a % b;**

**a = b;**

**b = t;**

**}**

**return a;**

**}**

**Проверка числа на простоту**

Функция ищет минимальный делитель числа n, не превосходящий n. Функция возвращает true, если число простое, и false — если составное. Сложность алгоритма — O(n).

**int isprime(int n)**

**{**

**int d = 2;**

**while (n % d != 0 && d \* d <= n)**

**++d;**

**return d \* d > n;**

**}**

**Разложение числа на множители**

Следующая функция возвращает список простых делителей числа с учетом их кратности (т.е. каждое простое число входит в результат столько раз, чему равна степень этого числа в разложении на простые). Сложность алгоритма — O(n).

**vector<int> factor(int n)**

**{**

**vector<int> res;**

**int d = 2;**

**while (d \* d <= n)**

**{**

**while (n % d == 0)**

**{**

**res.push\_back(d);**

**n /= d;**

**}**

**++d;**

**}**

**if (n > 1)**

**res.push\_back(n);**

**return res;**

**}**

**Решето Эратосфена**

Результатом работы этого алгоритма является массив, в котором i-й элемент равен true, если число i — простое. Сложность алгоритма — O(nlog⁡(log⁡n)).

Количество простых чисел, не превосходящих n примерно равно nln⁡n.

**vector<bool> is\_prime(n + 1, true);**

**for (int i = 2; i \* i <= n; ++i)**

**if (is\_prime[i])**

**for (int j = i \* i; j <= n; j += i)**

**is\_prime[j] = false;**

**Нахождение всех делителей числа**

Все делители числа n разбиваются на пары: если число d — делитель, то и число n/d тоже будет делителем. Таким образом, количество делителей числа n четно, если число не является точным квадратом.

Будем перебирать все делители числа и добавлять их сразу же по два. В конце добавим число n, если n — точный квадрат, отсортируем полученный список.

**vector<int> div;**

**int i = 1, j, k;**

**while (i \* i < n)**

**{**

**if (n % i == 0)**

**{**

**div.push\_back(i);**

**div.push\_back(n / i);**

**}**

**++i;**

**}**

**if (i \* i == n)**

**div.push\_back(i);**

**sort(div.begin(), div.end());**

#### Операции с целыми числами в кольце вычетов на языке C++

Во многих задачах необходимо в качестве ответа вывести "остаток от деления результата на некоторое число M".  Это делается для того, чтобы проверять большие ответы, без необходимости реализовывать длинную арифметику. В качестве числа M как правило берется большое простое число, наиболее распространенное значение для M — число 109+7. Это значение — простое, а также удовлетворяет условию, что при умножении его на 2 результат результат вмещается в 32-битной целочисленной знаковой переменной.

Поэтому в таких задачах можно выполнять все вычисления в кольце вычетов по модулю M, то есть после каждой операции брать остаток от деления на M.

Будем дальше, в основном, рассматривать значение M=109+7.

Примеры вычислений. Пусть переменные a и b хранят два значения из кольца вычетов по модулю M, то есть 0⩽a<M, 0⩽b<M. Реализуем арифметические операции в кольце вычетов.

Значение M лучше всего объявить глобальной константой в начале программы:

**const int M = 1000000007;**

#### Сложение

Сложение выполняется просто:

**int res = (a + b) % N;**

Если складывать три и более чисел, то может произойти переполнение, т.к. 3M>231. Поэтому складывать числа нужно по одному:

**int res = (a + b) % M;**

**res = (res + c) % M;**

#### Вычитание

При вычитании результат может стать отрицательным, что приведет к неправильному вычислению остатка (в С++ и Pascal как минимум). Поэтому к результату нужно добавить число M:

**int res = (a - b + M) % M;**

#### Умножение

При умножении нужно приводить сомножители к типу long long, т.к. при умножении двух чисел может произойти переполнение 32-битной целочисленной переменной:

**int res = (long long) a \* b % M;**

#### 

#### Быстрое возведение в степень

Если возникает необходимость вычислить значение anmodM (то есть остаток от деления am на M, то это можно сделать при помощи быстрого возведения в степень, которое использует двоичное представление числа n и следующие рекуррентные соотношения:

an=(a2)n/2 при четном n,  
an=a⋅an−1 при нечетном n.

Сложность алгоритма быстрого возведения в степень - O(log⁡n).

Можно использовать, например, следующий код:

**int pow(int a, int n)**

**{**

**if (n == 0)**

**return 1;**

**if (n % 2 == 1)**

**return (long long)a \* pow(a, n - 1) % M;**

**else**

**return pow((long long)a \* a % M, n / 2);**

**}**

#### Нахождение обратного элемента

Может возникнуть необходимость осуществления деления в кольце вычетов по модулю M. То есть для некоторого числа A известен его остаток от деления на M, равный a, и для числа B также известен остаток от деления на M и он равен b. Пусть C=A/B, нужно узнать, чему будет равен остаток от деления на M значения C.

Если M — простое, и b≠0, то есть число B не делится на M, то эта задача имеет решение. Для этого нужно найти для числа b обратный элемент: такое число b−1, что

b⋅b−1≡1(modM).

Это можно сделать при помощи малой теоремы Ферма, которая гласит, что для простого M и числа b, которое не делится на M верно:

bM−1≡1(modM).

Это означает, что обратный элемент b−1≡bM−2(modM), то есть его можно вычислить при помощи быстрого возведения в степень.

Таким образом, при простом M и b≠0 деление  a/b в кольце вычетов по модулю M равносильно операции abM−2(modM).

Для числа 2 значение обратного элемента можно найти еще проще: оно равно (M+1)/2(modM).