Олимпиадное программирование
Занятие 6. Арифметические алгоритмы

Труфанов Павел Николаевич

Онлайн-школа Фоксфорд

Foxford.ru 2019-2020

Проверьте число на простоту

Число простое, если оно делится только на 1 и на себя.
Проверьте, что данное число простое.

1. Перебрать все делители, $O(n)$ - долго
2. Перебрать делители до $\sqrt{n}$ - умно
3. Наука умеет быстрее, но мы это проходить не будем.

Код

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

Выпишите все делители числа

Для каждого делителя i есть обратный ему - $\frac{n}{i}$.
Хотя бы один из них не больше $\sqrt{n}$.
Переберем все числа до $\sqrt{n}$.

Код

```
vector<int> ans;
for (int i = 1; i * i <= n; ++i) {
    if (n % i == 0) {
        ans.push_back(i);
        if (i * i != n) {
            ans.push_back(n / i);
        }
    }
}
```

Факторизация - разложите число на простые множители

Хотелось бы тоже перебирать до корня. Но возможны простые делители больше $\sqrt{n}$. Например, $14 = 2 \times 7, 7 > \sqrt{14}$

Код

```
vector<int> ans;
for (int i = 2; i * i <= n; ++i) {
    while (n % i == 0) {
        ans.push_back(i);
        n /= i;
    }
}
if (n > 1) {
    ans.push_back(n)
}
```

Алгоритм Евклида

НОД - наибольший общий делитель. gcd(greatest common divisor) - по-английски.

Несколько свойств:

$gcd(a, b) = gcd(b, a)$

$gcd(a, b) = gcd(a, b - a)$

$gcd(a, 0) = a$

Алгоритм Евклида

$$gcd(a, b) = gcd(a, b - a) = gcd(a, b - 2a) = \cdots = gcd(a, b\%a) = gcd(b\%a, a)$$

Код с рекурсией

```
int gcd(int a, int b) {
    if (a == 0) {
        return b;
    }
    return gcd(b % a, a);
}
```

Код без рекурсии

```
int gcd(int a, int b) {
    while (a != 0) {
        b %= a;
        swap(a, b);
    }
    return b;
}
```

НОК

НОК - наименьшее общее кратное. lcm(least common multiple) - по-английски.

Утверждение: $lcm(a, b) = \frac{a \times b}{gcd(a,b)}$.
Докажем его

НОД нескольких чисел

Берем НОД первого и второго числа. Далее берем НОД результата и третьего числа. Далее НОД результата и четвертого числа. И так далее.

Расширенный алгоритм Евклида

Кроме нахождения НОД, научимся находить решения уравнения $ax + by = gcd(a, b)$.
Возьмем рекурсивную функцию нахождения НОД.
Пусть я уже нашел решение $(x1, y1)$ для пары чисел $b\%a, a$. Теперь хочу найти решение для пары a, b.

$$(b\%a) * x1 + a * y1 = gcd(a, b)$$
$$b\%a = b - \frac{b}{a} * a$$
$$(b - \frac{b}{a} * a) * x1 + a * y1 = gcd(a, b)$$
$$a * (y1 - \frac{b}{a} * x1) + b * x1 = gcd(a, b)$$
$$x = y1 - \frac{b}{a} * x1$$
$$y = x1$$

Расширенный алгоритм Евклида

База данного алгоритма - когда $a = 0$, то решением может стать пара чисел $(0, 1)$.

$a * 0 + b * 1 = gcd(a, b)$

$a = 0, gcd(a, b) = b$

$a * 0 + b * 1 = b$

Код

```
int gcd(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int g = gcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return g;
}
```

Диофантовы уравнения

$ax + by = c$
Мы умеем решать уравнения $ax + by = gcd(a, b)$
Сразу вытекает решение:
$g = gcd(a, b)$
$ax(c/g) + by(c/g) = c$
Имеет решения только, когда c кратно
$g = gcd(a, b)$.
Тогда решение, которое мы получили из расширенного алгоритма Евклида, нужно умножить на c/g.

Все решения диофантова уравнения

Если числа a, b были отрицательны в начале, при передаче их в алгоритм Евклида, от знака надо избавиться, но потом решение умножить на -1
Пусть $ax_0 + by_0 = c, g = gcd(a, b)$
$a(x_0 + b/g) + b(y_0 - a/g) =$
$ax_0 + ab/g + by_0 - ba/g = c$
То есть к коэффициенту при a можно прибавить $k \times b/g$, а из коэффициенты у b можно вычесть $k \times a/g$, и получим тоже корректные решения.
k можно выбрать любое.

Бинарное возведение в степень

Хочу посчитать выражение a^n % mod.

Напишем два равенства:
n - четное, $a^n = a^{n/2} * a^{n/2}$
n - нечетное, $a^n = a^{n-1} * a$

Напишем рекурсивную функцию для подсчета степени таким образом.

Код

```
int pow(int a, int n, int mod) {
    if (n == 1) {
        return a;
    }
    if (n % 2 == 0) {
        int x = pow(a, n / 2, mod);
        return (x * x) % mod;
    } else {
        int x = pow(a, n - 1, mod);
        return (x * a) % mod;
    }
}
```

Решето Эратосфена

Дано число N. Найдите все простые числа, не большие N. Сделайте это быстрее, чем $N\sqrt{N}$.

Давайте вычеркнем все числа, кратные 2. Потом вычеркнем все числа, кратные 3. Потом вычеркнем все числа, кратные 5.
Заметим, что все не вычеркнутые числа - простые. То есть алгоритм прост - идем по очереди по числам, берем невычеркнутые и вычеркиваем все, кратные им.
Сложность - $O(\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \dots) = O(N \log\log N)$

Код

```
vector<bool> isprime(N + 1);
for (int i = 2; i <= N; ++i) {
    if (isprime[i]) {
        for (int j = 2 * i; j <= N;
             j += i) {
            isprime[j] = false;
        }
    }
}
```

Улучшенное решето

Улучшенный алгоритм Евклида.
Проблема в том, что я могу число вычеркнуть несколько раз. Пусть есть массив lp, где $lp[i]$ - минимальный делитель числа i. lp - least prime.
Тогда каждое число x представимо как $lp[x] * y$.
Заметим, что $lp[x] < y, lp[x] < lp[y]$.
Давайте для каждого числа y, которое мы перебираем, перебирать все простые числа z, не большие $lp[y]$. Тогда мы смотрим на числа $x = z * y$, где $z = lp[x]$, так как простого делителя меньше нет. И такие числа x мы вычеркиваем.

Код

```
vector<int> lp(N + 1);
vector<int> pr;
for (int i = 2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int j = 0; j < pr.size() &&
         pr[j] <= lp[i] &&
         i * pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
    }
}
```

До встречи!

FOXFORD.RU

Онлайн-школа Фоксфорд