Алгоритмы и структуры данных. Теория чисел

densine

1 ноября 2025 г.

Содержание

1. Определения

Делители числа – все, что удовлетворяют

$$X \mid d <=> X = dr, r \in \mathbb{Z}$$

1. Делители числа X разбиваются на пары a и b такие что $ab = X$. Пусть $a \leq b$. Тогда $X = ab \geq a^2$, следовательно $a \leq \sqrt{x}$. Теперь пусть нам дано число x. Надо найти все его делители. Ответ – перебираем до $sqrtx$ и добавляем в ответ i и $\frac{x}{i}$, и не забываем обработать случай, когда $sqrtx$ – целое число.

2. Теперь нам, допустим, нам нужно определить простое число или нет. Используя предыдущую идею:

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

3. Теперь нам нужно разложить число на простые множители. Утверждается, что такое разложение единственно, т.е. единственно представление любого числа в виде произведения простых множителей. Например:

$$36 = 2 * 2 * 3 * 3 = 2^2 * 3^2$$

Делаем аналогичный перебор, но для каждого делим число на каждое число принадлежащее $[2..\sqrt{x}]$, и в конце добавляем в список число полученное в результате всех операций число, если оно неравно единице.

```
multiset<int> divs;
int x, n;
cin >> x;
n = x;

for (int d = 2; d*d <= n; d++) {
    while (x % d == 0) {
        divs.insert(x);
        x /= d;
    }
}

if (d != 1) divs.insert(d);
```

1.1. Что такое модульная арифметика

C++ не может хранить большие числа. int имеет лимит int $\leq 3 * 10^9$, a long long $\leq 9 * 10^{18}$. Из-за этого в некоторых задачах ответ просят выводить по определенному модулю, например $10^9 + 7$ и т.д. Суть модульной арифметики – вне зависимости от того, где мы берем модуль результат не изменяется.

$$(a * b) \bmod m = (a \bmod m * b \bmod m) \bmod m$$

Также этот закон выполняется при сложении и вычитании, но не при делении (в таком виде). Однако использовать можно и при делении. Есть такая штука... малая теорема Ферма (МТФ). Она гласит:

$$a^{m-1} \equiv 1$$

Поделим на a:

$$a^{m-2} \equiv \frac{1}{a}$$

То есть:

$$\frac{b}{a} \equiv b * a^{m-2}$$

Есть нюанс – обычно m достигает значение в 10^9. Нужен алгоритм быстрого возведения в степень, который возвращает: $a^x \bmod m$.

```
vector<char> opers;

while (x > 0) {
    if (x % 2 == 0) {
        opers.push_back('*');
        x /= 2;
    } else {
        opers.push_back('+');
        x -= 1;
    }
}

reverse(opers);
```

```
int ans = 1;

for (char o : opers) {
    if (o == '*') ans = (ans * ans) % m;
    else ans = (ans * a) % m;
}

return ans;
```

2. Алгоритм Евклида

Даны числа a, b. Найти $GCD(a, b)$. Суть в том, что $GCD(a, b) = GCD(a, b - a)$ при $a \leq b$.

```
int a, b; // Дано a < b
...

while (a > 0) { // GCD(0; a) = a
    b = b % a;
    swap(a, b); // Поменять a и b местами
}

return b;
```

3. Решето Эратосфена

Суть алгоритма – для числа x найти все простые до него. Сам алгоритм – перебор всех чисел от 2 до x.

1. Если число не зачеркнуто – оно простое. Мы хотим зачеркнуть все числа, которые стали составными из-за этого простого, т.е. все, что на него делятся – зачеркиваем.
2. иначе пропускаем число

```
vector<bool> primes(x, true);

for (int i = 2; i <= x; i++) {
    for (int j = 2*i; j <= x; j += i) {
        primes[j] = false;
    }
}
```

Хоть и кажется, что алгоритм работает за $O(n^2)$, но на самом деле сложность примерно $O(n*log(n))$. Этой оптимизации мало. Следующий вариант выполняется за $N*log(log(N)) \approx O(N)$.

```
...
for (int i = 2; i <= x; i++) {
    if (!primes[i]) continue; // Нет смысла зачеркивать числа,
```

```
                                // делящиеся на составные

    // Числа 2*i, 3*i ... i-1*i уже вычеркнуты
    for (int j = i*i; i <= x; j += i) {
        primes[j] = false;
    }
}
...
```

4. Расширенный алгоритм Евклида

Кратко суть обычного алгоритма – $GCD(a, b) = GCD(b \bmod a, a)$. Переопределим $GCD(a, b) = [x, y]$, где $[x, y]$ – корни $xa + by = g$ (g определено ранее). Скажем, что $GCD(a, b \mod a) = [x_1, y_1]$. Подставим в исходное: $x_1 a + (b \mod a)y_1 = g$. $b \mod a$ – нетривиальная операция. Представим $b \mod a = b - ar, r \in \mathbb{Z}$.

$$x_1 a + (b - ar)y_1 = g$$

$$x_1 a + y_1 b - ary_1 = g$$

$$(x_1 - ry_2)a + y_1 b = g$$

Т.е. $GCD(a, b) = [x_1 - ry_1, y_1]$. Почему это работает? При $GCD(0, a) = [1, 1]$, т.е. у нас есть переход, при котором мы уменьшаем один из параметров, и при минимальных значениях мы возвращаем что-либо, что на это укажет. Теперь решим такое уравнение:

$$ax + by = c$$

Рассмотрим два случая:

1. c делится на $GCD(a, b)$, уравнение сводится к знакомому виду.
2. c не делится на $GCD(a, b)$, тогда решений в целых числах нет.