Математика

Плотников Даниил Михайлович

Санкт-Петербургский государственный университет

Оглавление

Делители

Разложение на делители

Тема часто встречается как подзадача: количество/сумма делителей, проверка на квадрат, перебор делителей.

Пример: Дано число $n \leq 10^9$. Найти количество делителей ≥ 239

Наивное решение: пройтись по всем d = 1..n и если $d \mid n$ и при этом d ≥ 239, то увеливаем ответ. Сложность $O(n)$

Как придумать лучшую сложность?

Возьмем 2 делителя n, что мы можем о них сказать? Любой делитель d образует пару с n/d и произведение пары = n. Из двух чисел пары хотя бы одно $\leq \sqrt{n}$ Значит, если перебрали все $i \leq \sqrt{n}$, то нашли все пары $\rightarrow$ все делители. .

Разложение на делители (Код)

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

  sort(div.begin(), div.end());
  return div;
}
```

Факторизация (Код)

Определение: Факторизация (англ. factorization) - представление объекта в виде произведения других объектов .

```
vector<pair<long long,int>> factorize(long long n){
    vector<pair<long long,int>> f;
    for(long long p=2; p*p<=n; p += (p==2?1:2)){
        if(n%p==0){
            int cnt=0;
            while(n%p==0){ n/=p; ++cnt; }
            f.push_back({p,cnt});
        }
    }
    if(n>1) f.push_back({n,1});
    return f;
}
```

Простые числа

На основе предыдущего кода реализовать алгоритм, который проверяет, является ли число простым, не трудно:

```
bool is_prime(int n) {

    if (n <= 1) return false;
    if (n % 2 == 0) return n == 2;
    for (int i = 3; i * i <= n; i += 2) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
```

Решето Эратосфена

Решето Эратосфена (англ. sieve of Eratosthenes) - алгоритм нахождения всех простых чисел от 1 до n .

Основная идея соответствует названию алгоритма: запишем ряд чисел 1,2,...,n, а затем будем вычеркивать:

- сначала числа, делящиеся на 2, кроме самого числа 2,
- потом числа, делящиеся на 3, кроме самого числа 3,
- с числами, делящимися на 4, ничего делать не будем — мы их уже вычёркивали,
- потом продолжим вычеркивать числа, делящиеся на 5, кроме самого числа 5,

...и так далее.

Решето Эратосфена (Код)

Самая простая релизация будет выглядеть так:

```
vector<bool> sieve(int n) {
    vector<bool> is_prime(n + 1, true);
    for (int i = 2; i <= n; i++)
        if (is_prime[i])
            for (int j = 2 * i; j <= n; j += i) is_prime[j] = false;
    return is_prime;
}
```

Если память позволяет, то для оптимизации скорости лучше использовать не вектор bool, а вектор char - но он займёт в 8 раз больше места. Компьютер не умеет напрямую оперировать с битами, и поэтому при индексации к vector он сначала достаёт нужный байт, а затем битовыми операциями получает нужное значение, что занимает приличное количество времени.

Линейное решето

Идея: Пусть у нас есть:

- массив d[], где d[i] - минимальный простой делитель числа i;
- список всех найденных простых чисел p.

Когда мы идём по числам k = 2..n, мы делаем: Если k ещё не помечено (d[k] == 0), то k простое:

- записываем d[k] = k;
- добавляем k в список простых p.

Для каждого простого x из p в порядке возрастания:

- помечаем число m =$k * x$ как составное, записав d[m] = x;
- если x > d[k], останавливаемся - дальше числа будут иметь меньший простой делитель, и мы нарушим условие уникальности.

Линейное решето (Код)

```
const int N = 1e6;
int d[N + 1];           // d[i] = минимальный простой делитель
vector<int> primes;     // список найденных простых
void linear_sieve()
    for (int k = 2; k <= N; ++k)
        if (d[k] == 0)              // k простое
            d[k] = k;
            primes.push_back(k);
        for (int i = 0; i < (int)primes.size(); ++i)
            int x = primes[i];
            long long m = 1LL * k * x;
            if (x > d[k] || m > N) break;
            d[m] = x;               // минимальный простой делитель m -
это x
```

НОД и НОК

Определение НОД: $d = \max\{c \in N \mid c \mid a, c \mid b\}$

Определение НОК: $m = \min\{n \in N \mid a \mid n, b \mid n\}$

Задачи, чтобы научиться чувствовать:

- Найдите НОД(504, 540).
- Пусть НОД(a,b)= (a,b). Докажите, что (a) (a,b)= (a− b,b); (b) (ac,bc)= c·(a,b).
- Докажите, что НОД(a,b)·НОК(a,b)= ab.

- Еще задачки: https://3.shkolkovo.online/catalog/2887?Page=2&SubTheme=2888

Алгоритм Евклида

```
long long gcdll(long long a, long long b) {
    while (b) {
        long long t = a % b;
        a = b;
        b = t;
    }
    return a;                    // когда b == 0, a - это НОД
}
long long lcmll(long long a,long long b){
    return a / gcdll(a,b) * b;
}
```

Расширенный Евклид и обратный по модулю

- Находит x,y: ax+by=g, где g=gcd(a,b).
- Если gcd(a,m)=1 → обратный элемент $a^{-1} \equiv x \pmod{m}$.

```
long long egcd(long long a,long long b,long long& x,long long& y){
    if(b==0){ x=1; y=0; return a; }
    long long x1,y1; long long g=egcd(b,a%b,x1,y1);
    x=y1; y=x1 - (a/b)*y1; return g;
}
long long invmod(long long a,long long m){
    long long x,y; long long g=egcd(a,m,x,y);
    if(g!=1) return -1; // не существует
    x%=m; if(x<0) x+=m; return x;
}
```

Функция Эйлера ϕ(n)

Определение: $\phi(n) = \{1 \leq k \leq n \mid \gcd(k, n) = 1\}$

Свойства:

- Если n простое $\longrightarrow \phi(n)$= n − 1.
- Если n = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$, то

$$\varphi(n) = n * \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$$

- Упражнение: посчитать функцию Эйлера от этих аргументов 6, 12, 25

Функция Эйлера ϕ(n) (Код)

```
int phi(int n) {
    int res = n;
    for (int i = 2; i*i <= n; ++i)
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            res -= res / i;
        }
    if (n > 1) res -= res / n;
    return res;
}
```

Задача на пройденные темы (CF - 1700 rate):

Целое число x считается полупростым, если его можно записать в виде p·q, где p и q - (не обязательно разные) простые числа. Например, 9 является полупростым, так как его можно записать как 3·3, а 3 является простым числом. Скибидусу был дан массив a, содержащий n целых чисел. Он должен сообщить количество пар (i,j) таких, что $i \le j$ и lcm(ai,aj) является полупростым. Входные данные достаточно большие, чтобы решение за квадрат и больше не зашло

Модульная арифметика

Прежде чем переходить к комбинаторным задачам, нужно разобраться с модульной арифметикой. Возможно на codeforces вы встречались с формулировками по типу:

Выведите одно целое число — значение по модулю $10^9 + 7$.

В некоторых задачах ответ может быть огромен. В таких случаях просят использовать модульную арифметику. В противном случае приходится использовать длинную арифметику, что даст некоторое преимущество пишушим на java или python, а так же повысит вычислительную сложность алгоритма без видимой на то причины.

Операции по модулю обычно записываются как $(a + b) \bmod m$. Элементы ведущие сябея одинаково по модулю записываются как $a \equiv b (\bmod m)$

Базовые операции

Часть базовых операций можно выполнять смело не опасаясь за переполнение или потерю данных:

- Сложение: $(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$
- Вычитание: $(a - b) \bmod m = ((a \bmod m) - (b \bmod m)) \bmod m$
- Умножение: $(a \times b) \bmod m = ((a \bmod m) \times (b \bmod m)) \bmod m$

Однако для операций возведения в степень и деления всё обстоит чуть сложнее. Если в случае с возведением в степеры можно обойтись подобным образом: $a^3 \bmod m = (((a \bmod m) \times (a \bmod m)) \bmod m) \times (a \bmod m)) \bmod m$, что легко пишется простым циклом, то с делением всё куда сложнее, поскольку значение вовсе становится неверным:

1. $(10/2) \bmod 7 = 5$
2. $((10 \bmod 7)/(2 \bmod 7)) \bmod 7 = (3/2) \bmod 7 = 1$

В коде все операции выглядят как просто взятие остатка от деления:

```
int a = 10, b = 2, m = 7;
int sum = (a%m + b%m) % m;                  // 5
int diff = (a%m - b%m) % m;                 // 1
int mult = (a%m * b%m) % m;                 // 6
int power = (((a%m)*a%m)%m*a%m)%m;          // 6
int factorial = 1 % m;                      // 6! mod 7 = 720 mod 7 = 6
for(int i = 1; i <= 6; ++i){
    factorial = (factorial * i % m) % m;
}
```

Но что же делать с делением? Попробуем предсавить деление чуть иначе. $a/b = a \times b^{-1}$. Тогда нам достаточно найти число обратное данному, а в конетексте модульной арифметики достаточно найти число эквивалентое по модулю обратному.

Бинарное возведение в степень

Малая теорема ферма гласит $\forall p \in \text{Primes}, \forall a \in \mathbb{Z} \Rightarrow a^p \equiv a (\text{mod}\, p)$.

Два раза “поделим” этот известный результат на a: $a^p \equiv a \Rightarrow a^{p-2} \equiv a^{-1}$.

Мы нашли число которое эквивалентно обратному по простому модулю, что чаще всего и просят в задачах. Да, числа $10^9 + 7$ и 998244353 являются простыми числами, так что если просят по их модулю, то можно спокойно применять этот метод. Однако возводить в степень $10^9 + 5$ за $O(n)$ крайне неэффективно, однако бинарное возведение в степень позволяет сделать это за $O(\log n)$.

$$a^n = \begin{cases} a^{\frac{n}{2}} \times a^{\frac{n}{2}} \text{ if } n \text{ is even} \\ a^{n-1} \times a \text{ if } n \text{ is odd} \end{cases}$$

Этот подход простой и быстрый, однако следует помнить, что он работает только для простых модулей

Этот снипет можно просто вставить себе в шаблон и пользоваться.

```
const int mod = 1e9 + 7;
int binpow(int a, int n) {
    int res = 1;
    while (n != 0) {
        if (n & 1)
            res = (res * a) % mod;
        a = (a * a) % mod;
        n >>= 1;
    }
    return res;
}
int inv(int x) {
    return binpow(x, mod - 2);
}
```

Но что делать если модуль не простой? Тогда можно возводить число a в степень $\varphi(m) - 1$. Однако для этого надо производить факторизацию.

Другое решение это использовать расширенный алгоритм Евклида. В выражение $A \times x + B \times y = 1$ подставим в качестве A и B соответственно a и m: $a \times x + m \times y = 1$. Одним из решений уравнения и будет a^{-1}, потому что если взять уравнение по модулю m, то мы получим:

$$a \times x + m \times y = 1 \Leftrightarrow a \times x \equiv 1 \mid : a \Leftrightarrow x \equiv a^{-1} (\mathrm{mod}\, m).$$

Преимущества этого метода:

- Если обратное существует, то оно найдется даже если модуль не простой.
- Алгоритм проще выполнять руками.
- Алгоритм чуть быстрее, если его соптимизировать.

Комбинаторика

Биномиальные коэфициенты

Биномиальный коэфицент C_n^k – число способов, которыми можно выбрать k элементов из множества, содержащего n элементов.

Биномиальные коэффициенты можно вычислить, пользуясь рекуррентной формулой $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, с начальными значениями $C_n^0 = C_n^n = 1$.

Так же можно вычислить по формуле $C_n^k = \dfrac{n!}{k! \times (n-k)!}$

Справедливы следующие тождества:

- $C_n^k = C_n^{n-k}$
- $C_n^0 + C_n^1 + ... + C_n^n = 2^n$

Название исходит из связи возведением бинома $(a+b)$ в степень n:
$(a+b)^n = C(n,0) \times a_n \times b_0 + C(n,1) \times a_n - 1 \times b_1 + ... + C(n,n) \times a_0 \times b_n$

Числа каталана

Число каталана C_n определяет, сколько существует способов правильно расставить скобки в выражении, содержащем n левых и n правых скобок.

Например, C3 = 5, т. е. существует пять способов расставить три левые и три правые скобки:

- ( ) ( ) ( )
- (( )) ( )
- ( ) (( ))
- ((( )))
- (( ) ( ))

Правильная расстановка скобок определяется следующими правилами: пустая расстановка правильна если расстановка A правильна, то расстановка (A) также правильна если расстановки A и B правильны, то расстановка AB также правильна

Числа каталана можно вычислить рекурентно:

$$C_n = C_0 \times C_{n-1} + C_1 \times C_{n-2} + ... + C_{n-1} \times C_0$$

Базой рекурсии является случай $C_0 = 1$, поскольку вообще без скобок можно построить только пустую расстановку.

Также, числа каталана можно вычислить по формуле:

$$C_n = \frac{1}{n+1} \times C_{2n}^n = \mathrm{C}_{2n}^n - C_{2n}^{n-1}$$

Из этой формылы так же следует очень удобная рекурентная формула при помощи которой можно быстро предподсчитать значения:

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

Требует деления, что в модульной арифметике требует нахождения обратного элемента.

Название задачи	Условие	Ответ
Правильные скобочные последовательности	Сколько правильных скобочных последовательностей длины $2n$?	C_n
Деревья с n узлами	Сколько бинарных деревьев с n узлами?	C_n
Триангуляции многоугольника	Сколько способов разбить выпуклый n-угольник на треугольники непересекающимися диагоналями?	C_{n-2}
Непересекающиеся хорды на окружности	Сколько способов соединить $2n$ точек на окружности попарно непересекающимися хордами?	C_n
Разбиение на пары с условием	Сколько способов разбить $2n$ человек на n пар так, чтобы никакие две пары не “перекрещивались” при расположении по кругу?	C_n

Если в задаче есть вложенность, баланс, невозможность “пересечения”, два типа элементов в равном количестве, или ограничение на префиксы — скорее всего, это число Каталана.

```
vector<long long> precalcCatalan(int max_n) {
    vector<long long> C(max_n + 1);
    C[0] = 1;
    for (int i = 1; i <= max_n; ++i) {
        long long numerator = (C[i - 1] * (4 * i - 2)) % MOD;
        long long denominator = i + 1;
        C[i] = (numerator * inv(denominator, MOD)) % MOD;
    }
    return C;
}
```