Математика

Плотников Даниил Михайлович

Санкт-Петербургский государственный университет

Оглавление

Делители

Разложение на делители

Простые числа

НОД и НОК

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 $a \equiv b \pmod m$

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 степерь 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 $\forall p \in \text{Primes}, \forall a \in \mathbb{Z} \Rightarrow a^p \equiv a \pmod p$.

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

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:

$$a \times x + m \times y = 1 \Leftrightarrow a \times x \equiv 1 \mid : a \Leftrightarrow x \equiv a^{-1} \pmod{m}.$$

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 коэфиценты

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:

- $C_n^k = C_n^{n-k}$
- $\mathrm{C}_n^0 + C_n^1 + ... + C_n^n = 2^n$

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:

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

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;
}
```