AcWing 算法基础、提高课数学知识

廖涛

2022 年 4 月 15 日

目录

Chapter 1

质数

$\forall n \in [(1, +\infty) \cap \mathbb{Z}^+]$, 若 n 只有 1 和 n 两个约数, 那么 n 为质数（素数）, 否则为合数.

0、1 以及负整数既不是质数也不是合数.

1.1 判定质数

```
bool is_prime(int n)
{
    if(n < 2) return false;

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

1. 是否是大于等于 2 的整数,2. 从 2 n-1 中是否有 n 的约数.

不要写 $i * i < n$, 可能 $i * i$ 结果溢出 int；也不要在循环条件里写 $i <= sqrt(n)$, 否则每次循环都会调用 $sqrt()$.

$d \mid n$, 则必有 $\frac{n}{d} \mid n$, 故不需要枚举到 $n-1$, 即枚举每组约数的最小值即可.

时间复杂度为 $O(\sqrt{n})$

1.2 分解质因数

```
void divide(int n)
{
    for(int i = 2; i <= n / i; i ++)
    {
        int cnt = 0;
        if(n % i == 0)
        {
            while(n % i == 0)
            {
                n /= i;
                cnt ++;
            }
            printf("%d %d\n", i, cnt);
        }

    }

    if(n > 1) printf("%d 1\n", n);
    puts("");
}
```

从 2 开始枚举到 $\sqrt{n}$, 每找到一个质因子则将 n 中的该质因子除尽, 最后判断 n 是否大于 1, 若大于 1, 则此时的 n 也是所求质因子之一.

Tips1：所有合数因子在被枚举到之前会被更小的质数因子除尽. 故枚举到的合数必定不会满足 $n \bmod i = 0$. eg：$n = 24$,8 这个合数因子必然会被之前出现的 2 除尽（除 3 次）.

Tips2：n 最多只存在 1 个大于等于 $\sqrt{n}$ 的质因子, 故只需要枚举到 $\sqrt{n}$ 再判断最后的 n 情况如何即可. 简单证明, 假设存在

$$a \mid n, b \mid n, a > \sqrt{n}, b > \sqrt{n}$$

前两个条件可得 $a \times b \mid n$, 最后一个条件可得 $a \times b > n$, 相悖, 故假设不成立, 故 n 最多只存在 1 个大于等于 $\sqrt{n}$ 的质因子.

Tips3：当 $n = 2^k$ 时, 第一次枚举即能把 n 除尽, 此时的时间复杂度为 $\log_2 n$, 最坏情况则是需要枚举到 $\sqrt{n}$, 故该方法的时间复杂度为为 $O(\log_2 n)$ 与 $O(\sqrt{n})$ 之间.

1.3 质数筛

1.3.1 埃氏筛法

```
void get_primes(int n)
{
    for(int i = 2; i <= n; i ++)
    {
        if(st[i]) continue;

        primes[cnt ++] = i;
        for(int j = 2 * i; j <= n; j += i)
        {
            st[j] = true;
        }
    }
}
```

从 2 枚举到 n, 即能找到 n 以内的所有质数. 每次枚举到的 i 若 st[i] 为 false 则说明 i 为质数, 记录到 primes[] 中, 并筛掉其倍数. 否则 i 为合数, 直接跳过本次循环.

这里说明为什么不需要用合数筛掉其他数, 当 st[i] 为 true 时证明之前已经找到了 i 的一个或多个质因子, i 的倍数也必定是这些质因子的倍数, 所以当 i 为合数时,i 的倍数已经在之前被其质因子筛掉, 不需要再筛.

在枚举到 i 时,2 $i-1$ 的所有数都已经枚举过, 若 st[i] 为 false 则说明 2 $i-1$ 均不是 i 的因数, 则 i 为质数（素数）, 否则为合数.

时间复杂度分析, 当 $i=2$ 时内层循环次数为 $\frac{n}{2}$, 当 $i=3$ 时为 $\frac{n}{3}$..., 这里我们假设合数也进行内层循环, 则内层循环的次数和为

$$\frac{n}{2}+\frac{n}{3}+\frac{n}{4}+\cdots+\frac{n}{n}=n\times(\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n})$$

调和级数

$$\lim_{n\to\infty}\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}=\ln n$$

故内层循环的次数小于等于 $n\ln n$ 小于 $nlog_2n$, 故可以认为时间复杂度为

$$O(n\log_2 n)$$

只用质数筛选的情况（埃氏筛法）的时间复杂度为

$$O(n\log_2\log_2 n)$$

可以近似认为是 $O(N)$

1.3.2 欧拉筛法 (线形筛法)

```
void get_primes(int n)
{
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i]) primes[cnt ++] = i;

        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[primes[j] * i] = true;
            if(i % primes[j] == 0) break;
        }
    }
}
```

欧拉筛法的一个特点之一就是每个合数都是用其最小的质因子筛掉.

这里重点讲内层循环. 首先无论 i 是否为质数, 都会进内层循环.$primes[]$ 中小的数一定在前面.

当 $i \bmod primes[j] != 0$ 时,$primes[j]$ 的不是 i 的因子, 则 $primes[j]*i$ 的最小质因子就是 $primes[j]$.

当 $i \bmod primes[j] == 0$ 时, $primes[j]$ 是 i 的最小质因子, 也是 $primes[j]*i$ 的最小质因子. 但当 j ++ 后, $primes[j+1]*i$ 有更小的质因子 $primes[j]$（$primes[j]\mid i$）这里就不满足用最小质因子筛去合数的要求, 所以要 break.

每个合数 i 在被枚举到之前一定会被筛掉。假设合数 i 的最小质因子为 pj，另一个因子 i/pj 必然会在 i 之前被枚举到，当 i/pj 被枚举到时，内层循环则会把合数 i 筛掉。

Chapter 2

约数

2.1 试除法求约数

```
vector<int> get_divisors(int x)
{
vector<int> res;
for(int i = 1; i <= x / i; i ++)
{
    if(x % i == 0)
    {
        res.push_back(i);
        if(i * i != x) res.push_back(x / i);
    }
}
sort(res.begin(), res.end());
return res;
}
```

2.2 约数个数

对于整数 n, 由算数基本定理

$$n = p_1^{a_1} * p_2^{a_2} * p_3^{a_3} * \cdots * p_n^{a_n}$$

那么 n 的约数个数为

$$(a_1+1)*(a_2+1)*(a_3+1)*\cdots*(a_n+1)$$

简单证明，对于 n 的每个约数 d 都是这样的形式

$$d=p_1^{b_1}*p_2^{b_2}*\cdots*p_n^{b_n},0\le b_i\le a_i$$

每一个 b_i 的不同取值即取得了一个不同的 d, 故 b_i 不同的组合数量就是所求的约数个数.

```
#include <iostream>
#include <algorithm>
#include <unordered_map>

using namespace std;

const int N = 110, mod = 1e9 + 7;
typedef long long LL;

int main()
{
    int n;
    cin >> n;
    unordered_map<int, int> primes;

    while(n --)
    {
        int x;
        cin >> x;
        for(int i = 2; i <= x / i; i ++)
        {
            while(x % i == 0)
            {
                x /= i;
                primes[i] ++;
            }
        }
        if(x > 1) primes[x] ++;
    }
```

```

    LL res = 1;
    for(auto t : primes)
    {
        LL p = t.first, s = t.second;
        res = res * (s + 1) % mod;
    }

    cout << res << endl;

    return 0;
}
```

2.3 约数之和

对于整数 n, 由算数基本定理

$$n = p_1^{a_1} * p_2^{a_2} * p_3^{a_3} * \cdots * p_n^{a_n}$$

那么 n 的约数之和为

$$(p_1^0+p_1^1+\cdots+p_1^{a_1})*(p_2^0+p_2^1+\cdots+p_2^{a_2})*(p_3^0+p_3^1+\cdots+p_3^{a_3})*\cdots*(p_n^0+p_n^1+\cdots+p_n^{a_n})$$

简单理解为什么这样是所有约数之和，用乘法分配律对上式展开，相当于每个括号内选一项出来相乘，乘积相加。容易理解每个括号内选出来的一项相乘得到的结果就是一个约数，再相加就是所有约数之和。

```
#include <iostream>
#include <algorithm>
#include <unordered_map>

using namespace std;

typedef long long LL;
const int mod = 1e9 + 7;
unordered_map<int, int> primes;
int n;

//分解质因数
```

```cpp
void divide(int x)
{
    for(int i = 2; i <= x / i; i ++)
    {
        if(x % i == 0)
        {
            while(x % i == 0)
            {
                primes[i] ++;//质因数 i 的次数 +1
                x /= i;
            }
        }
    }
    if(x > 1) primes[x] ++;
}

int main()
{
    cin >> n;
    while(n --)
    {
        int x;
        scanf("%d", &x);
        divide(x);
    }

    LL res = 1;
    for(auto prime : primes)
    {
        int s = prime.second, nums = prime.first;
        LL tmp = 1;
        while(s --) tmp = (tmp * nums + 1) % mod;
        res = res * tmp % mod;
    }
    cout << res << endl;

    return 0;
}
```

这里是直接模拟了公式中的运算从头到尾直接循环相加，相乘，但是当分解出的质数个数较多时，这样的运算速度不够快。观察公式可以看到每个括号里的序列都是一个等比数列，可以直接利用等比数列的求和公式来计算。等比数列求和公式中的除法运算转换为乘以其逆元即可，因为这里的模数是质数，所以可以直接用快速幂求逆元。

```
#include <iostream>
#include <unordered_map>
#include <algorithm>
using namespace std;
typedef long long LL;
const int N = 2e9 + 10, mod = 1e9 + 7;

unordered_map<int, int> primes;

void get_primes(int n)
{
    for(int i = 2; i <= n / i; i ++)
    {
        if(n % i == 0)
        {
            while(n % i == 0)
            {
                primes[i] ++;
                n /= i;
            }
        }
    }
    if(n > 1) primes[n] ++;
}

LL qmi(LL a, LL b, LL p)
{
    LL res = 1;
    while(b)
    {
        if(b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
```

```
    }
    return res;
}

int main()
{
    int T;
    cin >> T;
    while(T --)
    {
        int a;
        cin >> a;
        get_primes(a);
    }

    LL res = 1;
    for(auto e : primes)
    {
        int p = e.first, k = e.second;
        if((p - 1) % mod == 0)
        {
            res = res * (k + 1) % mod;
        }
        else
        {
            res = res * (qmi(p, k + 1, mod) - 1) % mod * qmi(p - 1, mod - 2, mod) %
        }
    }
    cout << (res % mod + mod) % mod << endl;
    return 0;
}
```

注意是从 p^0 一直加到 p^k，即首项是 $p^0 = 1$，共有 $k+1$ 项。求逆元的时候要判断逆元是否存在，若逆元不存即为 $(p-1) \bmod mod = 0$，即 $p \bmod mod = 1$，这种情况直接让 res 乘以 $(k+1)*1$ 即可。
补充等比数列的求和公式，若

$$S = a_1 + a_2 + a_3 + \cdots + a_n$$

并且

$$\forall i \in [1, n], p = \frac{a_{i+1}}{a_i} = \frac{a_{i+2}}{a_{i+1}}, p \neq 1$$

则有

$$S = \frac{a_1 * (p^n - 1)}{p - 1}$$

2.4 最大公约数

$(a/\gcd(a,b)) * (b/\gcd(a,b)) * \gcd(a,b)$ 即 $a/\gcd(a,b) * b$ 为 a 和 b 的最小公倍数.

```
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}
```

若

$$d \mid a, d \mid b$$

那么有

$$d \mid ax + by$$

即 d 也能整除 a 和 b 的线性组合

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor * b$$

令 $c = \left\lfloor \frac{a}{b} \right\rfloor$ 则有

$$a \bmod b = a - c * b$$

即 $a \bmod b$ 也是 a 和 b 的线形组合，故当 $d \mid a, d \mid b$ 时，$d \mid a \bmod b$ 也成立。而当 $d \mid b, d \mid a \bmod b$ 时候，$a \bmod b = a - c * b$ 和 b 的一个线性组合 $a - c * b + c * b = a$ 也能被 d 整除，即 $d \mid a$。

这样就说明了 a 和 b 的约数也是 b 和 $a \bmod b$ 的约数，反过来 b 和 $a \bmod b$ 的约数也是 a 的约数。

当 $b = 0$ 时，a 和 b 最大公约数就为 a, $a \mid a$ 且 $a \mid 0$。

Chapter 3

欧拉函数

n 的欧拉函数为 [1, $n-1$] 内与 n 互质的数的个数.
若

$$n = p_1^{a_1} + p_2^{a_2} + p_3^{a_3} + \cdots + p_n^{a_n}$$

那么

$$\phi(n) = n*(1-\frac{1}{p_1})*(1-\frac{1}{p_2})*\cdots*(1-\frac{1}{p_n}) = n*(\frac{p_1-1}{p_1})*(\frac{p_2-1}{p_2})*\cdots*(\frac{p_n-1}{p_n})$$

当 n 为质数时

$$\phi(n) = n-1$$

3.1 欧拉函数筛

```
const int N = 1e6 + 10;
int n, cnt, prime[N], euler[N];
bool st[N];

void solve()
{
    euler[1] = 1;//1的欧拉函数为1
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])
        {
```

```
            prime[cnt ++] = i;
            euler[i] = i - 1;
        }

        for(int j = 0; prime[j] <= n / i; j ++)
        {
            int t = prime[j] * i;
            st[t] = true;
            if(i % prime[j] == 0)
            {
                euler[t] = euler[i] * prime[j];
                break;
            }
            euler[t] = euler[i] * (prime[j] - 1);
        }
    }

}
```

当 $i\%prime[j] == 0$ 时，$prime[j]$ 是 i 的一个质因子，此时 $euler[i]$ 内已经乘了 $1 - \frac{1}{prime[j]}$ 所以 $euler[t]$ 只需要额外再乘一个 $prime[j]$ 即可。

当 $i\%prime[j]! = 0$ 时，$prime[j]$ 不是 i 的质因子，$euler[i]$ 内没有乘 $1 - \frac{1}{prime[j]}$，所以 $euler[t]$ 还需要再乘 $prime[j] * (1 - \frac{1}{prime[j]}) = prime[j] - 1$ 即可。

3.2 欧拉定理

若 $a, p \in N^+, \gcd(a, p) = 1$，则有

$$a^{\phi(p)} \equiv 1 \pmod p$$

以上即欧拉定理

3.3 费马小定理

当 p 为质数时，欧拉定理即为一个特例—费马小定理。

若 p 为质数并且 a 不是 p 的倍数, 那么

$$a^{\phi(p)} \equiv 1 \pmod p$$

即

$$a^{p-1} \equiv 1 \pmod p$$

Chapter 4

快速幂

4.1 快速幂原理

```
LL qmi(LL a, LL b, LL p)
{
LL res = 1;
while(b)
{
    if(b & 1) res = res * a % p;
    a = a * a % p;
    b >>= 1;
}
return res;
}
```

从低到高枚举指数的每一二进制位, 若该位为 1 则结果乘以一次 a, 同时每次枚举每一位时 a 都要 $a = a * a$. 每次直接将指数 b 的最后一位右移去掉。eg: 当判断第一位是否为 1 时,$a = a^1$; 第二位是否为 1 时, $a = a^2$, ..., 第 n 位是否为 1 时, $a = a^n$(类似 a 进制)。时间复杂度为 $O(\log_2 b)$

4.2 快速幂求逆元

用快速幂求逆元必须保证模数 p 为质数，因为要用费马小定理构造逆元形式。若模数不是质数则只能用扩展欧几里得求逆元。

当模数 p 为质数时，保证 $a \bmod p \neq 0$ 则 a 必定与 p 互质，则 a 的逆元必定存在。由费马小定理知 $a^{p-1} \equiv 1 \pmod p$，显然有 $a * a^{p-2} \equiv 1 \pmod p$，故 a 的逆元为 a^{p-2}, 直接用快速幂计算即可.

```
qmi(a, p - 2, p);
```

Chapter 5

扩展欧几里得算法（粉碎机）

5.1 裴蜀定理

对于任意两个不全为 0 的正整数 a, b，必定存在两个整数 x, y，满足

$$a * x + b * y = \gcd(a, b)$$

这样的 $\gcd(a, b)$ 是上述条件下用 a, b 能凑出来的最小的正整数。

这样的 a, b 的所有线性组合必定是 $\gcd(a, b)$ 的倍数。有一个推论：一个整数是 a 和 b 的线性组合，当且仅当它是 $\gcd(a, b)$ 的倍数。

5.2 扩展欧几里得算法原理

```
int exgcd(int a, int b, int &x, int &y)
{
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }

    int d = exgcd(b, a % b, y, x);
    y = y - a / b * x;
    return d;
}
```

$b*y+(a \bmod b)*x = b*y+(a-\left\lfloor\frac{a}{b}\right\rfloor*b)*x = a*x+(y-\left\lfloor\frac{a}{b}\right\rfloor*x)*b$

故交换后 $y=y-\left\lfloor\frac{a}{b}\right\rfloor*x$

注意 x 和 y 的取值不是唯一的。对于所有满足 $ax_0+by_0=d, \forall k \in \mathbb{N}$

$$x = x_0 - \tfrac{b}{d}k$$
$$y = y_0 + \tfrac{a}{d}k$$

$ax+by=d$ 也成立。

5.3 扩展欧几里得算法求逆元

```
int exgcd(int a, int p, int &x, int &y)
{
    if(!p)
    {
        x = 1, y = 0;
        return a;
    }

    int d = exgcd(p, a % p, y, x);
    y = y - a / p * x;
    return d;
}
```

运算结束后首先判断返回值 d 是否为 1，因为 a 的逆元存在的条件是 $\gcd(a,p)=1$，即两数互质, 若 a 的逆元存在则

$$(x+p)\%p$$

为 a 的逆元

Chapter 6

中国剩余定理（孙子定理）

6.1 中国剩余定理的形式

中国剩余定理给出了以下一元线性同余方程组

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \quad\quad\vdots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

有解的判定条件以及有解时解的形式。

假设整数 $m_1, m_2, \ldots, m_n$ 两两互质，则对任意整数 $a_1, a_2, \ldots, a_n$，以上方程组有解。以下给出构造解的方法：

- 设 $M = m_1 * m_2 * \cdots * m_n = \prod_{i=1}^{n} m_i$，并设 $M_i = M/m_i, \forall i \in \{1, 2, \cdots, n\}$，即 M_i 是除了 m_i 以外的所有 m 的乘积。

- 设 $t_i = M_i^{-1}$ 为 M_i 模 m_i 的逆元，即 $t_i M_i \equiv 1 \pmod{m_i}, \forall i \in \{1, 2, \cdots, n\}$。

- 在模 M 的意义下，方程组只有一个解 $\sum_{i=1}^{n} a_i t_i M_i$。

6.2 扩展：模数不互质的情况

当给出的 $m_1, m_2, \ldots, m_n$ 不满足两两互质时也可以尝试构造解。给定下列方程组

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \end{cases}$$

将方程组转换为不定方程 $x = pm_1 + a_1 = qm_2 + a_2$，其中 p, q 为整数，则有 $pm_1 - qm_2 = a_2 - a_1$。由裴蜀定理可知当且仅当 $\gcd(m1, m2) \mid (a_2 - a_1)$ 时有解，否则无解。若有解则可以通过扩展欧几里得找到一组满足条件的 (p, q)，则原方程组的解为 $x \equiv b \pmod{M}$，其中 $b = pm_1 + a_1$，$M = \frac{m1}{\gcd(m1,m2)} * m2 = lcm(m1, m2)$。

若需要给出最小非负解：一组满足条件的解 p, q 可以改写成表示所有解的形式

$$\begin{cases} p = p + k\frac{m2}{d} \\ q = q - k\frac{m1}{d} \end{cases}$$

其中 $\forall k \in \mathbb{Z}$，运算`(p % (m2/d) + (m2/d)) % (m2/d)`即可得到最小非负解。

若方程组中有多个方程，则可以通过合并方程直至只剩下一个方程的方式来求解。将 $p = p + k\frac{m2}{d}$ 代入 $x = pm_1 + a_1$ 可得

$$x = (p + k\frac{m_2}{d})m_1 + a_1 = k\frac{m_1 m_2}{d} + pm_1 + a_1$$

则有新方程 $x = km + a$，其中 $m = \frac{m_1 m_2}{d} = lcm(m_1, m_2) = m_1/m_2 * \gcd(m_1, m_2)$，$a = pm_1 + a_1$，重复以上合并步骤直到只剩一个方程即可。当只剩一个方程时，我们让这个方程 $k = 0$ 即可得到解 $x = pm_1 + a_1$，即将方程合并后的 a，注意要对 $lcm(m_1, m_2) = m$ 取模。

下面给出 *AcWing 204. 表达整数的奇怪方式*的代码

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;


```

```cpp
LL exgcd(LL a, LL b, LL &x, LL &y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }

    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}


int main()
{
    int n;
    cin >> n;

    LL x = 0, m1, a1;
    cin >> m1 >> a1;
    for (int i = 0; i < n - 1; i ++ )
    {
        LL m2, a2;
        cin >> m2 >> a2;
        LL k1, k2;
        LL d = exgcd(m1, m2, k1, k2);
        if ((a2 - a1) % d)
        {
            x = -1;
            break;
        }

        k1 = k1 * (a2 - a1) / d;
        k1 = (k1 % (m2 / d) + m2 / d) % (m2 / d);//取最小非负解

        a1 = k1 * m1 + a1;//计算合并后方程的 a
        m1 = abs(m1 / d * m2);//计算合并后方程的 m = lcm(m1, m2)
    }
```

```

    if (x != -1) x = (a1 % m1 + m1) % m1;// 答案要对 lcm(m1, m2) 即现在的 m1 取模

    cout << x << endl;

    return 0;
}
```

Chapter 7

组合数 C_a^b

7.1 a、b 较小的组合数

```
#include <cstdio>
using namespace std;
const int N = 2010, mod = 1e9 + 7;

int f[N][N];

int main()
{
    int n;
    scanf("%d", &n);

    for(int i = 0; i < N; i ++)
        for(int j = 0; j <= i; j ++)
        {
            if(!j) f[i][j] = 1;
            else f[i][j] = ((long long)f[i - 1][j] + f[i - 1][j - 1]) % mod;
        }

    while(n --)
    {
        int a, b;
        scanf("%d%d", &a, &b);
```

```
        printf("%d\n", f[a][b]);
    }
    return 0;
}
```

$$\mathrm{C}_a^b = \mathrm{C}_{a-1}^{b-1} + \mathrm{C}_{a-1}^b$$

类似动态规划，从 a 中选 b 个的方法数量等于不选第 b 个物品的方法数量加上选第 b 个物品的方法数量。即从 $a-1$ 中选 b 的方法数量加上从 $a-1$ 中选 $b-1$ 的方法数量。

7.2 a、b 较大的组合数

```
#include <cstdio>
using namespace std;
typedef long long LL;
const int N = 1e5 + 10, mod = 1e9 + 7;

int infact[N], fact[N];
int T;

int qmi(int a, int b, int p)
{
    int res = 1;
    while(b)
    {
        if(b & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        b >>= 1;
    }
    return res;
}

int main()
{
    scanf("%d", &T);

    fact[0] = 1, infact[0] = 1;
```

```
    for(int i = 1; i < N; i ++)
    {
        fact[i] = (LL)fact[i - 1] * i % mod;
        infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
    }

    while(T --)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        printf("%d\n", (LL)fact[a] * infact[b] % mod * infact[a - b] % mod);
    }

    return 0;
}
```

$$C_a^b = \frac{a!}{b!(a-b)!}$$

预处理出从 1 到 N 每个数的阶乘及阶乘的逆元即可。除法可以转换为乘以逆元。这里的模数 1e9+7 为质数, 所以求逆元时可以方便地直接用快速幂。

7.3 a、b 特别大的组合数

```
#include <cstdio>
using namespace std;
typedef long long LL;

int qmi(LL a, LL b, int p)
{
    int res = 1;
    while(b)
    {
        if(b & 1) res = (LL)res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}
```

```
int C(LL a, LL b, int p)
{
    LL res = 1;
    for(int i = 1, j = a; i <= b; i ++, j --)
    {
        res = (LL)res * j % p;
        res = (LL)res * qmi(i, p - 2, p) % p;
    }

    return res;
}

int lucas(LL a, LL b, int p)
{
    if(a < p && b <p) return C(a, b, p);
    else return (LL)lucas(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}

int main()
{
    int n;
    scanf("%d", &n);
    while(n --)
    {
        LL a, b;
        int p;
        scanf("%lld%lld%d", &a, &b, &p);
        printf("%d\n", lucas(a, b, p));
    }
    return 0;
}
```

$$\mathrm{C}_a^b = \frac{a!}{b!(a-b)!} = \frac{a*(a-1)*(a-2)*\cdots*(a-b+2)*(a-b+1)}{b!}$$

卢卡斯定理

$$\mathrm{C}_a^b = \mathrm{C}_{a \bmod p}^{b \bmod p} + \mathrm{C}_{\frac{a}{p}}^{\frac{b}{p}} \pmod p$$

本题每次询问的 p 是不固定的，所以不能像上题预处理阶乘和阶乘的逆元后直接用来计算。代码中的 C 函数即是对组合数计算的模拟。

7.4 不取模的高精度组合数

```
#include <cstdio>
#include <vector>
#include <cstring>
using namespace std;
const int N = 5010;

int primes[N], cnt, sum[N];
bool st[N];

void get_primes(int x)
{
    for(int i = 2; i <= x; i ++)
    {
        if(!st[i]) primes[cnt ++] = i;
        for(int j = 0; primes[j] <= x / i; j ++)
        {
            st[primes[j] * i] = true;
            if(i % primes[j] == 0) break;
        }
    }
}

int get_sum(int a, int p)
{
    int res = 0;
    while(a)
    {
        res += a / p;
        a /= p;
    }
    return res;
}

vector<int> mul(vector<int>a, int b)
{
    vector<int> res;
    int t = 0;
```

```
    for(int i = 0; i < a.size(); i ++)
    {
        t += a[i] * b;
        res.push_back(t % 10);
        t /= 10;
    }

    while(t)
    {
        res.push_back(t % 10);
        t /= 10;
    }
    return res;
}

int main()
{
    int a, b;
    scanf("%d%d", &a, &b);
    get_primes(a);

    //阶乘分解，记录每个质因数的次数
    for(int i = 0; i < cnt; i ++)
    {
        int p = primes[i];
        sum[i] = get_sum(a, p) - get_sum(b, p) - get_sum(a - b, p);
    }

    //高精度乘法
    vector<int> ans;
    ans.push_back(1);
    for(int i = 0; i < cnt; i ++)
        for(int j = 1; j <= sum[i]; j ++)
            ans = mul(ans, primes[i]);

    for(int i = ans.size() - 1; i >= 0; i --)
        printf("%d", ans[i]);

    puts("");
```

```
    return 0;
}
```

对组合数计算中的阶乘做阶乘分解，即将阶乘分解为算术基本定理的形式。

$$n = p_1^{a_1} + p_2^{a_2} + p_3^{a_3} + \cdots + p_n^{a_n}$$

$get\_sum(a, p)$ 就是质因子 p 在 $a!$ 中出现的次数，$get\_sum(b, p)$ 就是质因子 p 在 $b!$ 中出现的次数，因为是除以 $b!$，所以要减去，类似地 $get\_sum(a-b, p)$ 也是一样。

记录每个质因子应该乘的次数，然后用高精度乘法将每个质因子乘以相应的次数即可。

7.5 卡特兰数

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

Chapter 8

容斥原理

Chapter 9

博弈论