

AcWing 算法基础、提高课数学知识

廖涛

2022 年 4 月 15 日

目录

1	质数	5
1.1	判定质数	5
1.2	分解质因数	6
1.3	质数筛	7
1.3.1	埃氏筛法	7
1.3.2	欧拉筛法 (线形筛法)	8
2	约数	9
2.1	试除法求约数	9
2.2	约数个数	9
2.3	约数之和	11
2.4	最大公约数	15
3	欧拉函数	17
3.1	欧拉函数筛	17
3.2	欧拉定理	18
3.3	费马小定理	18
4	快速幂	21
4.1	快速幂原理	21
4.2	快速幂求逆元	21
5	扩展欧几里得算法 (粉碎机)	23
5.1	裴蜀定理	23
5.2	扩展欧几里得算法原理	23

5.3 扩展欧几里得算法求逆元	24
6 中国剩余定理 (孙子定理)	25
7 组合数 C_a^b	27
7.1 a、b 较小的组合数	27
7.2 a、b 较大的组合数	28
7.3 a、b 特别大的组合数	29
7.4 不取模的高精度组合数	31
8 容斥原理	35
9 博弈论	37

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

```
1 bool is_prime(int n)
2 {
3     if(n < 2) return false;
4
5     for(int i = 2; i <= n / i; i++)
6     {
7         if(n % i == 0) return false;
8     }
9     return true;
10 }
```

1. 是否是大于等于 2 的整数, 2. 从 2 到 \sqrt{n} 中是否有 n 的约数.

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

$d | n$, 则必有 $\frac{n}{d} | n$, 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

```

1 void divide(int n)
2 {
3     for(int i = 2; i <= n / i; i++)
4     {
5         int cnt = 0;
6         if(n % i == 0)
7         {
8             while(n % i == 0)
9             {
10                 n /= i;
11                 cnt++;
12             }
13             printf("%d %d\n", i, cnt);
14         }
15     }
16 }
17
18 if(n > 1) printf("%d 1\n", n);
19 puts("");
20 }
```

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 $O(\log_2 n)$ 与 $O(\sqrt{n})$ 之间.

1.3 质数筛

1.3.1 埃氏筛法

```

1 void get_primes(int n)
2 {
3     for(int i = 2; i <= n; i++)
4     {
5         if(st[i]) continue;
6
7         primes[cnt++] = i;
8         for(int j = 2 * i; j <= n; j += i)
9             {
10                 st[j] = true;
11             }
12     }
13 }
```

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, $2i-1$ 的所有数都已经枚举过, 若 $st[i]$ 为 `false` 则说明 $2i-1$ 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

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

调和级数

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

故内层循环的次数小于等于 $n \ln n$ 小于 $n \log_2 n$, 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)

```

1 void get_primes(int n)
2 {
3     for(int i = 2; i <= n; i++)
4     {
5         if(!st[i]) primes[cnt++] = i;
6
7         for(int j = 0; primes[j] <= n / i; j++)
8         {
9             st[primes[j] * i] = true;
10            if(i % primes[j] == 0) break;
11        }
12    }
13 }
```

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 $i \bmod primes[j] \neq 0$ 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 p_j , 另一个因子 i/p_j 必然会在 i 之前被枚举到, 当 i/p_j 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

```
1 vector<int> get_divisors(int x)
2 {
3     vector<int> res;
4     for(int i = 1; i <= x / i; i ++ )
5     {
6         if(x % i == 0)
7         {
8             res.push_back(i);
9             if(i * i != x) res.push_back(x / i);
10        }
11    }
12    sort(res.begin(), res.end());
13    return res;
14 }
```

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

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

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.

```

1  #include <iostream>
2  #include <algorithm>
3  #include <unordered_map>
4
5  using namespace std;
6
7  const int N = 110, mod = 1e9 + 7;
8  typedef long long LL;
9
10 int main()
11 {
12     int n;
13     cin >> n;
14     unordered_map<int, int> primes;
15
16     while(n --)
17     {
18         int x;
19         cin >> x;
20         for(int i = 2; i <= x / i; i ++ )
21         {
22             while(x % i == 0)
23             {
24                 x /= i;
25                 primes[i] ++;
26             }
27         }
28         if(x > 1) primes[x] ++;
29     }

```

```

30
31     LL res = 1;
32     for(auto t : primes)
33     {
34         LL p = t.first, s = t.second;
35         res = res * (s + 1) % mod;
36     }
37
38     cout << res << endl;
39
40     return 0;
41 }

```

2.3 约数之和

对于整数 n , 由算数基本定理

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

那么 n 的约数之和为

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

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。

```

1  #include <iostream>
2  #include <algorithm>
3  #include <unordered_map>
4
5  using namespace std;
6
7  typedef long long LL;
8  const int mod = 1e9 + 7;
9  unordered_map<int, int> primes;
10 int n;
11
12 // 分解质因数

```

```
13 void divide(int x)
14 {
15     for(int i = 2; i <= x / i; i++)
16     {
17         if(x % i == 0)
18         {
19             while(x % i == 0)
20             {
21                 primes[i] ++; // 质因数 i 的次数 +1
22                 x /= i;
23             }
24         }
25     }
26     if(x > 1) primes[x] ++;
27 }
28
29 int main()
30 {
31     cin >> n;
32     while(n --)
33     {
34         int x;
35         scanf("%d", &x);
36         divide(x);
37     }
38
39     LL res = 1;
40     for(auto prime : primes)
41     {
42         int s = prime.second, nums = prime.first;
43         LL tmp = 1;
44         while(s --) tmp = (tmp * nums + 1) % mod;
45         res = res * tmp % mod;
46     }
47     cout << res << endl;
48
49     return 0;
50 }
```

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。

```
1  #include <iostream>
2  #include <unordered_map>
3  #include <algorithm>
4  using namespace std;
5  typedef long long LL;
6  const int N = 2e9 + 10, mod = 1e9 + 7;
7
8  unordered_map<int, int> primes;
9
10 void get_primes(int n)
11 {
12     for(int i = 2; i <= n / i; i ++){
13         {
14             if(n % i == 0)
15             {
16                 while(n % i == 0)
17                 {
18                     primes[i] ++;
19                     n /= i;
20                 }
21             }
22         }
23         if(n > 1) primes[n] ++;
24     }
25
26 LL qmi(LL a, LL b, LL p)
27 {
28     LL res = 1;
29     while(b)
30     {
31         if(b & 1) res = res * a % p;
32         a = a * a % p;
33         b >>= 1;
```

```

34     }
35     return res;
36 }
37
38 int main()
39 {
40     int T;
41     cin >> T;
42     while(T --)
43     {
44         int a;
45         cin >> a;
46         get_primes(a);
47     }
48
49     LL res = 1;
50     for(auto e : primes)
51     {
52         int p = e.first, k = e.second;
53         if((p - 1) % mod == 0)
54         {
55             res = res * (k + 1) % mod;
56         }
57         else
58         {
59             res = res * (qmi(p, k + 1, mod) - 1) % mod * qmi(p - 1, mod - 2, mod) %
60         }
61     }
62     cout << (res % mod + mod) % mod << endl;
63     return 0;
64 }

```

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.

```

1 int gcd(int a, int b)
2 {
3     return b ? gcd(b, a % b) : a;
4 }

```

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

```
1  const int N = 1e6 + 10;
2  int n, cnt, prime[N], euler[N];
3  bool st[N];
4
5  void solve()
6  {
7      euler[1] = 1; // 1 的欧拉函数为 1
8      for(int i = 2; i <= n; i++)
9      {
10         if(!st[i])
11         {
```

```

12         prime[cnt++] = i;
13         euler[i] = i - 1;
14     }
15
16     for(int j = 0; prime[j] <= n / i; j++)
17     {
18         int t = prime[j] * i;
19         st[t] = true;
20         if(i % prime[j] == 0)
21         {
22             euler[t] = euler[i] * prime[j];
23             break;
24         }
25         euler[t] = euler[i] * (prime[j] - 1);
26     }
27 }
28
29 }
```

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

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

3.2 欧拉定理

若 $a, p \in \mathbb{N}^+, \gcd(a, p) = 1$, 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 a 为任意正整数, p 为质数, 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

```
1 LL qmi(LL a, LL b, LL p)
2 {
3     LL res = 1;
4     while(b)
5     {
6         if(b & 1) res = res * a % p;
7         a = a * a % p;
8         b >>= 1;
9     }
10    return res;
11 }
```

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.

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

Chapter 5

扩展欧几里得算法（粉碎机）

5.1 裴蜀定理

对于任意两个正整数 a, b ，必定存在两个非 0 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。

5.2 扩展欧几里得算法原理

```
1 int exgcd(int a, int b, int &x, int &y)
2 {
3     if(!b)
4     {
5         x = 1, y = 0;
6         return a;
7     }
8
9     int d = exgcd(b, a % b, y, x);
10    y = y - a / b * x;
11    return d;
12 }
```

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

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 $ax_0 + by_0 = d, \forall k \in \mathbb{N}$

$$x = x_0 - \frac{b}{d}k$$

$$y = y_0 + \frac{a}{d}k$$

$ax + by = d$ 也成立。

5.3 扩展欧几里得算法求逆元

```

1  int exgcd(int a, int p, int &x, int &y)
2  {
3      if(!p)
4      {
5          x = 1, y = 0;
6          return a;
7      }
8
9      int d = exgcd(p, a % p, y, x);
10     y = y - a / p * x;
11     return d;
12 }
```

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）

Chapter 7

组合数 C_a^b

7.1 a、b 较小的组合数

```
1  #include <cstdio>
2  using namespace std;
3  const int N = 2010, mod = 1e9 + 7;
4
5  int f[N][N];
6
7  int main()
8  {
9      int n;
10     scanf("%d", &n);
11
12     for(int i = 0; i < N; i++)
13         for(int j = 0; j <= i; j++)
14             {
15                 if(!j) f[i][j] = 1;
16                 else f[i][j] = ((long long)f[i - 1][j] + f[i - 1][j - 1]) % mod;
17             }
18
19     while(n --)
20     {
21         int a, b;
22         scanf("%d%d", &a, &b);
```

```

23     printf("%d\n", f[a][b]);
24 }
25     return 0;
26 }

```

$$C_a^b = C_{a-1}^{b-1} + C_{a-1}^b$$

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

```

1  #include <cstdio>
2  using namespace std;
3  typedef long long LL;
4  const int N = 1e5 + 10, mod = 1e9 + 7;
5
6  int infact[N], fact[N];
7  int T;
8
9  int qmi(int a, int b, int p)
10 {
11     int res = 1;
12     while(b)
13     {
14         if(b & 1) res = (LL)res * a % p;
15         a = (LL)a * a % p;
16         b >>= 1;
17     }
18     return res;
19 }
20
21 int main()
22 {
23     scanf("%d", &T);
24
25     fact[0] = 1, infact[0] = 1;

```

```

26     for(int i = 1; i < N; i ++){
27     {
28         fact[i] = (LL)fact[i - 1] * i % mod;
29         infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
30     }
31
32     while(T --){
33     {
34         int a, b;
35         scanf("%d%d", &a, &b);
36         printf("%d\n", (LL)fact[a] * infact[b] % mod * infact[a - b] % mod);
37     }
38
39     return 0;
40 }

```

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

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

```

1  #include <cstdio>
2  using namespace std;
3  typedef long long LL;
4
5  int qmi(LL a, LL b, int p)
6  {
7      int res = 1;
8      while(b)
9      {
10         if(b & 1) res = (LL)res * a % p;
11         a = a * a % p;
12         b >>= 1;
13     }
14     return res;
15 }

```

```

16
17 int C(LL a, LL b, int p)
18 {
19     LL res = 1;
20     for(int i = 1, j = a; i <= b; i ++, j --)
21     {
22         res = (LL)res * j % p;
23         res = (LL)res * qmi(i, p - 2, p) % p;
24     }
25
26     return res;
27 }
28
29 int lucas(LL a, LL b, int p)
30 {
31     if(a < p && b < p) return C(a, b, p);
32     else return (LL)lucas(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
33 }
34
35 int main()
36 {
37     int n;
38     scanf("%d", &n);
39     while(n --)
40     {
41         LL a, b;
42         int p;
43         scanf("%lld%lld%d", &a, &b, &p);
44         printf("%d\n", lucas(a, b, p));
45     }
46     return 0;
47 }

```

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

卢卡斯定理

$$C_a^b = C_{a \bmod p}^{b \bmod p} + C_{\frac{a}{p}}^{\frac{b}{p}} \pmod{p}$$

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

```
1  #include <cstdio>
2  #include <vector>
3  #include <cstring>
4  using namespace std;
5  const int N = 5010;
6
7  int primes[N], cnt, sum[N];
8  bool st[N];
9
10 void get_primes(int x)
11 {
12     for(int i = 2; i <= x; i++)
13     {
14         if(!st[i]) primes[cnt++] = i;
15         for(int j = 0; primes[j] <= x / i; j++)
16         {
17             st[primes[j] * i] = true;
18             if(i % primes[j] == 0) break;
19         }
20     }
21 }
22
23 int get_sum(int a, int p)
24 {
25     int res = 0;
26     while(a)
27     {
28         res += a / p;
29         a /= p;
30     }
31     return res;
32 }
33
34 vector<int> mul(vector<int>a, int b)
35 {
36     vector<int> res;
37     int t = 0;
```

```
38     for(int i = 0; i < a.size(); i ++)  
39     {  
40         t += a[i] * b;  
41         res.push_back(t % 10);  
42         t /= 10;  
43     }  
44  
45     while(t)  
46     {  
47         res.push_back(t % 10);  
48         t /= 10;  
49     }  
50     return res;  
51 }  
52  
53 int main()  
54 {  
55     int a, b;  
56     scanf("%d%d", &a, &b);  
57     get_primes(a);  
58  
59     // 阶乘分解，记录每个质因数的次数  
60     for(int i = 0; i < cnt; i ++)  
61     {  
62         int p = primes[i];  
63         sum[i] = get_sum(a, p) - get_sum(b, p) - get_sum(a - b, p);  
64     }  
65  
66     // 高精度乘法  
67     vector<int> ans;  
68     ans.push_back(1);  
69     for(int i = 0; i < cnt; i ++)  
70         for(int j = 1; j <= sum[i]; j ++)  
71             ans = mul(ans, primes[i]);  
72  
73     for(int i = ans.size() - 1; i >= 0; i --)  
74         printf("%d", ans[i]);  
75  
76     puts("");
```



```
77     return 0;  
78 }
```

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。

Chapter 8

容斥原理

Chapter 9

博弈论