算法基础（二十四）：数学基础 - 组合数 - 快速幂求法

基本思想

当组合数的数据进一步增大，比如下面这个例子：

> 给定 n 组询问，每组询问给定两个整数 a，b，请你输出 $C_a^b \bmod (10^9+7)$ 的值。
>
> **输入格式**
>
> 第一行包含整数 n。
>
> 接下来 n 行，每行包含一组 a 和 b。
>
> **输出格式**
>
> 共 n 行，每行输出一个询问的解。
>
> **数据范围**
>
> $1 \le n \le 10000$,
> $1 \le b \le a \le 10^5$
>
> **输入样例：**
>
> ```
> 3
> 3 1
> 5 3
> 2 2
> ```
>
> **输出样例：**
>
> ```
> 3
> 10
> 1
> ```

上一篇中的查询次数为：`10000`，`a，b`的范围为：`2000`采用的方法是利用公式：$C_a^b = C_{a-1}^{b-1} + C_{a-1}^{b}$进行递推。在这里的查询次数是：`10000`，`a，b`的范围为：`100000`由于递推的时间复杂度是`O(N^2)`， 当数据的规模达到本次的情况时仍然较大，所以得换一个方式。

我们注意到组合数的求解公式：

$$C_a^b = \frac{a*(a-1)*...(a-b+1)}{1*2*3*...b} = \frac{a!}{b!(a-b)!}$$

我们需要注意的是：

$$C_a^b mod(10^7+7) = \frac{a!}{b!(a-b)!} mod(10^9+7) \neq \frac{a! mod(10^9+7)}{b!(a-b)! mod(10^9+7)}$$

根据模运算法则我们可以得到：

$$C_a^b mod(10^9+7) = \frac{a!}{b!(a-b)!} mod(10^9+7) = (a! mod(10^9+7) * \frac{1}{b!(a-b)!} mod(10^9+7)) mod(10^9+7)$$

可以将括号内乘法的第二项看作一个整体运算结果不变：

$$\frac{1}{b!(a-b)!}mod(10^9+7) = (\frac{1}{b!}mod(10^9+7) * \frac{1}{(a-b)!}mod(10^9+7))mod(10^9+7)$$

令$10^9+7=k$代入可以得到：

$$(a!modk * \left\{(\frac{1}{b!}modk * \frac{1}{(a-b)!}modk)modk\right\})modk$$

由于逆元的定义：

$$b! * \frac{1}{b!} \equiv 1modk$$

所以：$\frac{1}{b!} = b!^{-1}$

那么有：

$$C_a^b modk = (a!modk * \left\{(b!^{-1}modk * (a-b)!^{-1}modk)modk\right\})modk$$

令$fact(i) = i!mod(1e9+7)$，令$infact(i) = (i!)^{-1}mod(1e9+7)$，也就是`i!`的逆元模`1e9 + 7`

那么就有：

$$(fact(a) * \{(infact(b) * infact(a-b))modk\})modk$$

最左边可以加上一个`modk`，得到：

$$(fact(a)modk * \{(infact(b) * infact(a-b))modk\})modk$$

由于两项都取余，所以可以将后面的中括号去掉，化简得到：

$$(fact(a) * infact(b) * infact(a-b))modk$$

由费马小定理：

$$b!^{-1} = b^{k-2}modk$$

所以：

$$infact(b)modk = b!^{k-2}modk$$

由于快速幂求逆元的时候时间复杂度是`Olog(N)`，那么我们预处理所有`1 ~ 100000`的阶乘和逆元的时候总的操作次数最多不会超过`100000 * log(1000000)`，而第一种方案的复杂度是`100000 * 100000`， 这样就大大减少了时间复杂度

代码实现

```
#include<iostream>
#include<algorithm>
using namespace std;

const int N = 100010, mod = 1e9 + 7;
//定义1 ~ 100010所有数的阶乘以及阶乘逆元的数组
int fact[N], infact[N];
typedef long long LL;
```

```

//快速幂求a的k次方
int qmi(int a, int k, int p)
{
    int res = 1;
    while(k)
    {
        if(k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}


int main()
{
    //0的阶乘以及阶乘的逆元是1
    fact[0] = infact[0] = 1;

    //从1开始处理
    for(int i = 1; i < N; i ++)
    {
        //预处理每一个数的阶乘
        fact[i] = (LL)fact[i - 1] * i % mod;
        //预处理每个阶乘的逆元，这个式子展开就是((i - 1)^(mod - 2) % mod * i^(mod - 2) % mod)  % mod
        //也就是i^(mod - 2) % mod
        infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;

    }

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

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

    }

    return 0;
}
```

为什么要使用`LL`

注意题目中数的阶乘取模后的最大值是`10 ^ 9`，而`int`类型的数据最大是`2147483647`，所以如果两个数相乘得到的结果是可能超出`int`范围的，同理，如果三个数相乘，而`long long`的类型最大是`922 3372 0368 5477 5807 (922*10^16)`，所以三个数相乘是可能超出范围的，所以在最后一步，需要将

$$(fact(a) * infact(b) * infact(a-b)) mod k$$

化为：

$$(\{fact(a) * infact(a-b)\}modk * infact(b)modk)modk$$

展开可以得到：

$$(\{fact(a)modk * infact(a-b)modk\}modk * infact(b)modk)modk$$

在代码中$fact(i)$就已经代表的是$(i)!modk$，所以，代码中就可以省去第一个$modk$，第二个$modk$也可以省去，第四个也可以省， 然后`c`语言默认的计算顺序是从左到右的，这样可以简化为`(LL)fact[a] * infact[b] % mod * infact[a - b] % mod`，这样两两相乘的结果会先取模，然后再与第三项相乘，这样就不会超出范围了

补充一些模运算的知识：

在数学上进行乘法与取模混合的时候计算顺序是从左到右计算的，比如式子：`5 * 4 mod 3`，结果是`2`而不是`5`，但是如果是乘法中两项都带有取模符号，比如`5 mod 3 * 4 mod 3`，这时不管是从左到右计算，还是先对`4，5`取模再相乘结果都是一样的，所以在上面的推导中可以将后面的$\frac{1}{b!(a-b)!}mod(10^9+7)$绑在一起进行代换，但是在`c`语言中统一都是从左到右计算，并且取模的优先级是高于加减的。

模运算是没有除法的展开规则的，遇到$(a/b)modk$的时候需要化为$(a*b^{-1})modk$，而计算逆元的时候需要用到费马小定理以及求快速幂的公式

另外需要注意的是在模运算中$(a*b*c)modk \neq amodk*bmodk*cmodk$

假如有四个数相乘后`mod k`，比如：`(a * b * c * d) mod k`

转化就是：`(((a mod k * b mod k)mod k * c mod k) mod k * d mod k)mod k`

在`c`语言中可以写成：`(((a mod k * b )mod k * c ) mod k * d )mod k`