算法基础（二十五）：数学基础 - 组合数 - 卢卡斯定理

我们回忆一下前两种求组合数的方法

当`a，b`的取值在`2000`以内的时候我们采用递推的方式求解，这时算法的时间复杂度是`O(N^2)`

当`a，b`的取值在`100000`以内的时候我们采用快速幂的方式求解，这时算法的时间复杂度是`O(NlogN)`

而当`a，b`的取值在`1 ~ 10 ^ 18`次方的时候，再使用前两种方法仍然会超时，所以这时得用卢卡斯定理求解

基本原理

卢卡斯定理：

$$C_a^b \equiv C_{amodp}^{bmodp} * C_{a/p}^{b/p} (modp)$$

b/p在这里是取整的意思

证明：

首先，`a`和`b`都可以表示成`p`进制的形式：

$$a = a_k * p^k + a_{k-1} * p^{k-1} + ... + a_0 * p^0$$

$$b = b_k * p^k + b_{k-1} * p^{k-1} + ... + b_0 * p^0$$

然后，由于二项式定理可得：

$$(1+x)^p = C_p^0 + C_p^1 x + C_p^2 x^2 + ... + C_p^p x^p$$

由于除了第一项和最后一项，其余项中都含有`p`这个因数，所以有第一个引理：

$$(1+x)^p \equiv 1 + x^p (modp)$$

令$m = k1p + r1$则可以得到：

$$(1+x)^m \equiv (1+x)^{k1p+r1} \equiv (1+x)^{k1p} (1+x)^{r1} \equiv [(1+x)^p]^{k1} (1+x)^{r1} (modp)$$

由上面的第一个引理得到：

$$(1+x)^m \equiv [1+x^p]^{k1} (1+x)^{r1} (modp)$$

用二项式定理展开得到：

$$(1+x)^m \equiv [1+x^p]^{k1} (1+x)^{r1} \equiv \Sigma_0^{k1} C_{k1}^i x^{ip} * \Sigma_0^{r1} C_{r1}^j x^j (modp)$$

由这个式子，我们可以得到：$m = k1p + r1$，也就是$k1 = m/p$，$r1 = mmodp$

所以有：

$$(1+x)^m \equiv \Sigma_{k=0}^m C_m^k x^k \equiv \Sigma_0^{m/p} C_{m/p}^i x^{ip} * \Sigma_0^{mmodp} C_{mmodp}^j x^j (modp)$$

我们对比一下上面这个式子两边的系数，当$k=ip+j$的时候，也就是x同次幂的时候，左左右两边的系数是相等的，根据模运算规则，也就是：

$$C_m^k \equiv C_{\lfloor m/p \rfloor}^i * C_{mmodp}^j (modp) \equiv C_{\lfloor m/p \rfloor}^{\lfloor k/p \rfloor} * C_{mmodp}^{kmodp} (modp)$$

令$k=b$，$m=a$，即得：

$$C_a^b \equiv C_{amodp}^{bmodp} * C_{a/p}^{b/p} (modp)$$

b/p在这里是取整的意思

证毕。

代码实现

给定 n 组询问，每组询问给定三个整数 a, b, p，其中 p 是质数，请你输出 $C_a^b \bmod p$ 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, b, p。

输出格式

共 n 行，每行输出一个询问的解。

数据范围

$1 \le n \le 20$,
$1 \le b \le a \le 10^{18}$,
$1 \le p \le 10^5$,

输入样例：

```
3
5 3 7
3 1 5
6 4 13
```

输出样例：

```
3
3
2
```

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

typedef long long LL;

//模数，定义为全局变量
int p;

//快速幂
```

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

//直接用定义的方式求组合数
int C(int a, int b)
{
    int res = 1;

    for(int i = 1, j = a; i <= b; i ++, j --)
    {
        //当i = b 的时候 j = a - b + 1
        //利用a*(a - 1) *...(a - b + 1)/b!来计算组合数
        res = (LL)res * j % p;
        res = (LL)res * qmi(i , p - 2) % p;
    }
    return res;

}

//递归的方式求解组合数
int lucas(LL a, LL b)
{
    //递归结束的条件
    if(a < p && b < p) return C(a, b);//小于p了之后直接从定义出发计算

    return (LL)C(a % p, b % p) * lucas(a / p, b / p) % p;

}

int main()
{
    int n;
    cin >> n;
    while(n --)
    {
        LL a, b;
        cin >> a >> b >> p;
        cout << lucas(a, b) << endl;

    }

    return 0;
}
```

时间复杂度：

卢卡斯递归的过程中总的递归次数不会超过log_pN，N是指数据的最大范围，题目中是10^{18}，p是模数的最大值，然后每次递归的时候都用普通的方式求解组合数，求一次最多不会超过p次，求组合数的时候每次循环都用快速幂，次数不会超过$logp$，所以总的时间复杂度就是$log_pN*p*logp$，大约为$p*log_pN$（因为log_pN与$logp$不会同时取得最大值，对比前两种方式，这种方式将`N`放进了`log`里面，进一步降低了复杂度

而如果我们使用预处理阶乘和逆元的方式求解组合数的话，由于求组合数的数范围不会超过`p`，所以预处理的时间复杂度是$plogp$，总的时间复杂度大约就是$p+log_pN$，可以将复杂度进一步降低。