**数学**

2020.09.05

目录

[数学 3](#_Toc50217218)

[快速乘 3](#_Toc50217219)

[快速幂 3](#_Toc50217220)

[多项式专题 3](#_Toc50217221)

[FFT 3](#_Toc50217222)

[NTT 5](#_Toc50217223)

[求原根 6](#_Toc50217224)

[多项式求逆 7](#_Toc50217225)

[多项式除法和取模 8](#_Toc50217226)

[多项式开方 8](#_Toc50217227)

[多项式求导和积分 8](#_Toc50217228)

[多项式exp 8](#_Toc50217229)

[多项式快速幂 8](#_Toc50217230)

[FWT 8](#_Toc50217231)

[从高斯消元到矩阵树定理 9](#_Toc50217232)

[高斯消元 9](#_Toc50217233)

[矩阵的逆 13](#_Toc50217234)

[行列式及特殊矩阵行列式求法 14](#_Toc50217235)

[矩阵树定理 14](#_Toc50217236)

[线性基 15](#_Toc50217237)

[最大公因数 16](#_Toc50217238)

[最大公因数 16](#_Toc50217239)

[扩展欧几里得 16](#_Toc50217240)

[类欧几里得算法 16](#_Toc50217241)

[中国剩余定理 17](#_Toc50217242)

[模数互质 17](#_Toc50217243)

[模数不互质 17](#_Toc50217244)

[从拓展卢卡斯定理到卢卡斯定理 18](#_Toc50217245)

[拓展卢卡斯定理 18](#_Toc50217246)

[卢卡斯定理 21](#_Toc50217247)

[高次同余方程 23](#_Toc50217248)

[指数型方程（BSGS） 23](#_Toc50217249)

[幂函数方程 24](#_Toc50217250)

[莫比乌斯反演 24](#_Toc50217251)

[大质数探测与分解 25](#_Toc50217252)

[MillerRabin 25](#_Toc50217253)

[Pollardrho 26](#_Toc50217254)

[扩展欧拉定理 27](#_Toc50217255)

[Polya和Burnside 27](#_Toc50217256)

[容斥原理 28](#_Toc50217257)

[齐次线性递推 28](#_Toc50217258)

[生成函数 28](#_Toc50217259)

[二次剩余 28](#_Toc50217260)

[特殊数列 28](#_Toc50217261)

[组合数 28](#_Toc50217262)

[卡特兰数 29](#_Toc50217263)

[斯特林数 29](#_Toc50217264)

[斐波那契数列 29](#_Toc50217265)

[分拆数 30](#_Toc50217266)

[一些结论 31](#_Toc50217267)

[LGV引理（统计不相交路径数量） 31](#_Toc50217268)

# 数学

## 快速乘

**//a\*b%p，当a和b比较大的时候用**

ll add(ll a, ll b, ll p){

return a + b >= p ? a + b - p : a + b;

}

ll mul(ll a, ll b, ll p){

ll c = 0;

a %= p;

while(b){

if(b & 1)

c = add(c , a , p);

a = add(a , a , p);

b >>= 1;

}

return c;

}

## 快速幂

ll fpw(ll o, ll t, ll p){

ll c = 1;

while(t){

if(t & 1)

c = mul(c , o , p);

o = mul(o , o , p);

t >>= 1;

}

return c;

}

## 多项式专题

### FFT

#include<cmath>

typedef double db; **//这里可以换成long double**

const db pai = acos(-1);

class Complex{

public:

db r , i;

Complex () {r = i = 0;}

Complex (db \_r , db \_i) : r(\_r) , i(\_i) {}

Complex operator+ (Complex c){ return Complex(r + c.r , i + c.i);}

Complex operator- (Complex c){ return Complex(r - c.r , i - c.i);}

Complex operator\* (Complex c){ return Complex(r \* c.r - i \* c.i , r \* c.i + i \* c.r);}

Complex operator/ (db c){ return Complex(r / c , i / c);}

Complex conj(){ return Complex(r , -i);}

};

const int maxn = 6e5 + 10; **//设置大小**

Complex a[maxn] , b[maxn] , c[maxn];

Complex wu1[maxn] , wu2[maxn];

int n , m , d;

int rev[maxn];

void getrev(int L){ **//L = n + m**

d = 1;

while(d < L)

d <<= 1;

int half = (d >> 1);

for(int i = 1 ; i < d ; ++ i)

rev[i] = ((rev[i >> 1] >> 1) | (i & 1 ? half : 0));

}

void init(){

d = MAXNUM; **//这里d可以设置为数据范围最大值，一次全部赋值**

int half = (d >> 1);

for(int i = half ; i < d ; ++ i)

wu1[i] = Complex(cos((i - half) \* pai / half) , sin((i - half) \* pai / half));

for(int i = half - 1 ; i ; -- i)

wu1[i] = wu1[i << 1];

for(int i = 1 ; i < d ; ++ i)

wu2[i] = wu1[i].conj();

}

void fft\_core(Complex \*c , Complex \*wu){

for(int i = 1 ; i < d ; ++ i)

if(rev[i] > i)

swap(c[i] , c[rev[i]]);

for(int i = 1 ; i < d ; i <<= 1)

for(int j = 0 ; j < d ; j += (i << 1))

for(int k = 0 ; k < i ; ++ k){

Complex c1 = c[j + k] , c2 = c[i + j + k] \* wu[i + k];

c[j + k] = c1 + c2;

c[i + j + k] = c1 - c2;

}

}

void fft(Complex \*c , Complex \*wu){

fft\_core(c , wu);

}

void ifft(Complex \*c , Complex \*wu){

fft\_core(c , wu);

for(int i = 0 ; i < d ; ++ i)

c[i] = c[i] / d;

}

### NTT

#include<vector>

typedef long long ll;

typedef vector <int> Poly;

const int maxn = 8e5 + 10 , p = 998244353 , g = 3;

**//设置maxn为数组最大值，p是模数，g是原根**

int wu1[maxn] , wu2[maxn];

int rev[maxn];

int n , m , d;

int fpw(int x , int t){

if(!t) return 1;

int o = fpw(x , t >> 1);

o = 1ll \* o \* o % p;

if(t & 1) o = 1ll \* o \* x % p;

return o;

}

void getrev(int L){ **//L = n + m**

d = 1;

while(d < L)

d <<= 1;

int half = (d >> 1);

for(int i = 1 ; i < d ; ++ i)

rev[i] = ((rev[i >> 1] >> 1) | (i & 1 ? half : 0));

}

void ntt\_core(Poly &a , int\* wu){

for(int i = 0 ; i < d ; ++ i)

if(rev[i] > i)

swap(a[i] , a[rev[i]]);

for(int i = 1 ; i < d ; i <<= 1)

for(int j = 0 ; j < d ; j += (i << 1))

for(int k = 0 ; k < i ; k ++){

int a1 = a[j + k] , a2 = 1ll \* a[j + k + i] \* wu[i + k] % p;

a[j + k] = (a1 + a2) % p;

a[j + k + i] = (a1 + p - a2) % p;

}

}

void intt(Poly &a , int\* wu){

ntt\_core(a , wu);

int c = fpw(d , p - 2);

for(int i = 0 ; i < d ; ++ i)

a[i] = 1ll \* a[i] \* c % p;

}

void ntt(Poly &a , int\* wu){

ntt\_core(a , wu);

}

void init(){

d = MAXNUM;

int c = fpw(g , (p - 1) / d) , half = (d >> 1);

wu1[half] = 1;

for(int i = half + 1 ; i < d ; ++ i)

wu1[i] = 1ll \* wu1[i - 1] \* c % p;

for(int i = half - 1 ; i ; -- i)

wu1[i] = wu1[i << 1];

c = fpw(c , p - 2);

wu2[half] = 1;

for(int i = half + 1 ; i < d ; ++ i)

wu2[i] = 1ll \* wu2[i - 1] \* c % p;

for(int i = half - 1 ; i ; -- i)

wu2[i] = wu2[i << 1];

}

Poly mul(Poly a , Poly b){

int L = (int)a.size() + (int)b.size() - 1;

getrev(L);

a.resize(d) , ntt(a , wu1);

b.resize(d) , ntt(b , wu1);

Poly c(d);

for(int i = 0 ; i < d ; ++ i)

c[i] = 1ll \* a[i] \* b[i] % p;

intt(c , wu2);

c.resize(L);

return c;

}

**//调试部分**

void print(Poly p){

for(int i = 0 ; i < (int)p.size() ; ++ i)

cout << p[i] << " ";

cout << endl;

}

### 求原根

int p , n = p - 1; **//p不是质数时n要换成φ(p)**

int r[30] , cnt = 0; **//r数组记录的是不同的质因数**

for(int i = 2 ; i \* i <= n ; ++ i){

if(n % i == 0){

r[++ cnt] = i;

while(n % i == 0)

n /= i;

}

}

if(n > 1)

r[++ cnt] = n;

for(int i = 1 ; i <= p ; ++ i){

bool flag = 1;

for(int j = 1 ; j <= cnt ; ++ j)

if(fpw(i , p / r[j]) == 1){

flag = 0;

break;

}

if(flag){

cout << i << endl;

break;

}

}

### 多项式求逆

**//当使用多项式求逆时，数组大小需要再乘2倍**

Poly polyinv(Poly a , int o){ **//o表示数组长度**

d = 1;

while(d < (o << 1)) d <<= 1;

Poly ans(d) , b(d) , c(d);

a.resize(d);

ans[0] = fpw(a[0] , p - 2);

if(o == 1){

ans.resize(o);

return ans;

}

for(int d1 = 2 ; ; d1 <<= 1){

d = (d1 << 1);

for(int i = 0 ; i < d1 ; ++ i)

b[i] = a[i];

int half = (d >> 1);

for(int i = 1 ; i < d ; ++ i)

rev[i] = ((rev[i >> 1] >> 1) | (i & 1 ? half : 0));

ntt(ans , wu1); ntt(b , wu1);

for(int i = 0 ; i < d ; ++ i)

c[i] = 1ll \* ans[i] \* (2 + p - 1ll \* ans[i] \* b[i] % p) % p;

intt(c , wu2);

for(int i = 0 ; i < d1 ; ++ i)

ans[i] = c[i];

for(int i = d1 ; i < d ; ++ i)

ans[i] = b[i] = 0;

if(d1 >= o) break;

}

ans.resize(o);

return ans;

}

### 多项式除法和取模

**//div是除法，mod是取模**

Poly div(Poly a , Poly b){

int o = a.size() - b.size() + 1;

if(o < 1)

return Poly(1);

polyrev(a) , polyrev(b);

a.resize(o) , b.resize(o);

a = mul(a , polyinv(b , o));

a.resize(o);

polyrev(a);

return a;

}

Poly mod(Poly &a , Poly &b , Poly &c){

Poly g = mul(b , c) , ret = a;

for(int i = 0 ; i < a.size() ; ++ i)

ret[i] = (ret[i] + p - g[i]) % p;

ret.resize(b.size() - 1);

return ret;

}

### 多项式开方

### 多项式求导和积分

### 多项式exp

### 多项式快速幂

### FWT

const int p = 998244353 , inv2 = (p + 1) / 2;

const int N = 1 << 18;

int d;

int Plus(int x, int y){

return x + y >= p ? x + y - p : x + y;

}

int Minus(int x , int y){

return x - y >= 0 ? x - y : x - y + p;

}

int Mul(int x , int y){

return 1ll \* x \* y % p;

}

**//下面所有tp=1是正变换，tp=-1是逆变换**

void FWT\_or(int \*a, int tp){

for(int i = 1 ; i < d ; i <<= 1)

for(int j = 0 ; j < d ; j += (i << 1))

for(int k = 0 ; k < i ; ++ k)

a[i + j + k] = tp == 1 ? Plus(a[j + k] , a[i + j + k]) : Minus(a[i + j + k] , a[j + k]);

}

void FWT\_and(int \*a , int tp){

for(int i = 1 ; i < d ; i <<= 1)

for(int j = 0 ; j < d ; j += (i << 1))

for(int k = 0 ; k < i ; ++ k)

a[j + k] = tp == 1 ? Plus(a[j + k] , a[i + j + k]) : Minus(a[j + k] , a[i + j + k]);

}

void FWT\_xor(int \*a , int tp){

for(int i = 1 ; i < d ; i <<= 1)

for(int j = 0 ; j < d ; j += (i << 1))

for(int k = 0 ; k < i ; ++ k){

int A1 = a[j + k] , A2 = a[i + j + k];

if(tp == 1){

a[j + k] = Plus(A1 , A2); a[i + j + k] = Minus(A1 , A2);

}

else{

a[j + k] = Mul(Plus(A1 , A2) , inv2) , a[i + j + k] = Mul(Minus(A1 , A2) , inv2);

}

}

}

int main(){

FWT\_and(A , 1); FWT\_and(B , 1);

for(int i = 0 ; i < d ; ++ i)

C[i] = Mul(A[i] , B[i]);

FWT\_and(C , -1);

}

## 从高斯消元到矩阵树定理

### 高斯消元

前置技能，枚举主元法的高斯消元，即消第n个元的时候，每次选出绝对值最大一个方程对其他的进行消元。

但是常见的高斯消元模板并不能很好的解决，哪些元是变元以及哪些是确定解的问题。

假设方程有个未知数，个方程，那么我们在高斯消元的同时记录一个变量，表示使用了个方程，注意考虑到第个方程发现第个变元可能无解，这时不++，因为它可能还有用。

如果使用的方程数小于未知变量数，说明可能无解或者存在自由元。

判断无解即变量用完了方程没用完 ，这是看剩下的方程存不存在

判断一个变量是不是自由元，其实这个时候能解的一步就可以解出来，判断一下这一行是否有能解出来的即可。

#### 在实数域下高斯消元

#include<iostream>  
#include<stdio.h>  
#include<string.h>  
#include<algorithm>  
#define maxn 405  
using namespace std;  
  
double eps=1e-7;  
double myabs(double now)  
{  
 return now>0.0 ? now : -now;  
}  
int n,m;// n个方程m个未知数   
int vis[maxn];  
double ans[maxn];  
double a[maxn][maxn];  
int gauss()  
{  
 int now=1;//考虑到了第now个方程了   
 for(int i=1;i<=n && now<=m;i++,now++)  
 {  
 int row=now;  
 for(int j=now+1;j<=m;j++)  
 if(myabs(a[j][i])>myabs(a[row][i]))  
 row=j;  
   
 if(myabs(a[row][i])<=eps)   
 {  
 now--;//这里是因为第i个变量可能解不出来了，但是第now个方程依然可能有用   
 continue;  
 }  
   
 if(row!=i)  
 for(int j=1;j<=n+1;j++)  
 swap(a[i][j],a[row][j]);  
   
   
 for(int j=1;j<=m;j++)//全消了  
 if(myabs(a[j][i])>eps && j!=now)   
 {  
 double temp=a[j][i]/a[now][i];  
 for(int k=i;k<=n+1;k++)  
 a[j][k]-=temp\*a[now][k];  
 }  
 }  
   
 //考虑无解   
 for(int i=now;i<=m;i++)  
 {  
 if(myabs(a[i][n+1])>eps)  
 {  
 return 0;  
 }  
 }  
   
 if(now<=n)//用了的方程少于变元数   
 {  
 for(int i=1;i<=now-1;i++)  
 {  
 int cnt=0,pos=0;  
 for(int j=1;j<=n;j++)  
 if(myabs(a[i][j])>eps && !vis[j])  
 {  
 cnt++;  
 pos=j;  
 }   
 if(cnt>1) continue;  
   
 ans[pos]=a[i][n+1];  
 for(int j=1;j<=n;j++)  
 {  
 if(j!=pos && myabs(a[j][i])>1e-6)  
 {  
 ans[pos]-=a[i][j]\*ans[j];  
 }  
 }  
 ans[pos]=ans[pos]/a[i][pos];  
 vis[pos]=1;  
 }  
 }  
 else  
 {  
 for(int i=1;i<=n;i++)  
 ans[i]=a[i][n+1]/a[i][i],vis[i]=1;  
 }   
 return 1;  
}  
  
int main()  
{  
 scanf("%d",&n); m=n;  
   
 for(int i=1;i<=m;i++)  
 for(int j=1;j<=n+1;j++)  
 scanf("%lf",&a[i][j]);  
   
 if(!gauss())   
 {  
 printf("No Solution\n");  
 return 0;  
 }  
   
 for(int i=1;i<=n;i++)  
 {  
 if(!vis[i])  
 {  
 printf("%d not determined\n",i);  
 }  
 else  
 {  
 printf("%.2lf\n",ans[i]);  
 }  
 }  
}

#### 在质数域下的高斯消元（行列式求值）

int a[M] , b[M];

int f[M][M];

int n , m;

int fpw(int x , int t){

if(!t) return 1;

int s = fpw(x , t >> 1);

s = 1ll \* s \* s % p;

if(t & 1) s = 1ll \* s \* x % p;

return s;

}

int guass(){

for(int i = 1 ; i <= m ; ++ i){

if(!f[i][i])

for(int j = i + 1 ; j <= m ; ++ j)

if(!f[j][i]){

for(int k = i ; k <= m ; ++ k)

swap(f[i][k] , f[j][k]);

break;

}

if(!f[i][i]) continue;

int inv = fpw(f[i][i] , p - 2);

for(int j = i + 1 ; j <= m ; ++ j){

int c = 1ll \* inv \* (p - f[j][i]) % p;

for(int k = i ; k <= m ; ++ k)

f[j][k] = (f[j][k] + 1ll \* c \* f[i][k]) % p;

}

}

int ans = 1;

for(int i = 1 ; i <= m ; ++ i)

ans = 1ll \* ans \* f[i][i] % p;

return ans;

}

#### 在异或域下的高斯消元

### 矩阵的逆

学过高等代数的人都知道，给原矩阵右边添上一个单位矩阵，将原矩阵高斯消元为单位矩阵，右边的矩阵即为原矩阵的逆。

### 行列式及特殊矩阵行列式求法

学过高等代数的人都知道，将矩阵用线性变换的方法化成对角矩阵之后，对角线元素相乘，即为行列式。但是是用高斯消元的时候会交换两行，这样的话会使得行列式的值乘，记录一下即可。

行列式可以按行展开，公式为

#### 特殊矩阵行列式求法

利用范德蒙行列式

三对角行列式，直接展开得到递推关系

箭型行列式，即只有两条边和对角线上有值的行列式。把第列加到第1行，把第1列的第i行消成0。

除对角线外其他元素相等，或其他元素按行成比例，用第一行乘比例系数去减其他行。

两条线带几个点的行列式，直接展开递推即可。

### 矩阵树定理

矩阵树定理主要求解三类问题，给无向图求生成树数量，给有向图和一个点求以这个点为根的内向树和外向树的数量。

一般需要构造两个矩阵分别为联通矩阵和度数矩阵。

#### 构造方法

给无向图求生成树数量，为**与**之间的边数，为第个点的度数

以一个点为根的内向树，为**到**之间的边数，为第个点的入度

以一个点为根的外向树，为**到**之间的边数，为第个点的出度

#### 求解方法

先得到矩阵

给无向图求生成树数量，任意删去一行一列(一般为最后一行一列)，求行列式

以一个点为根的内向树，删去给定点所在行所在列，求行列式

以一个点为根的外向树，删去给定点所在行所在列，求行列式

## 线性基

typedef long long ll;

const int B = 60; **//最大位数**

ll v[B];

int n;

**//这个是完全消干净的，每一个有值的位置只有它和非自由元**

void insert(ll \*num , long long val){

for(int i = B - 1 ; i >= 0 ; -- i)

if(val & (1ll << i)){

if(num[i])

val ^= num[i];

else{

num[i] = val;

for(int j = i - 1 ; j >= 0 ; -- j)

if(num[i] & (1ll << j))

num[i] ^= num[j];

for(int j = i + 1 ; j < B ; ++ j)

if(num[j] & (1ll << i))

num[j] ^= num[i];

break;

}

}

}

ll Query\_Max(ll \*num){ **//询问线性基能组成的最大数**

ll ans = 0;

for(int i = B - 1 ; i >= 0 ; -- i)

ans ^= num[i];

return ans;

}

## 最大公因数

### 最大公因数

int gcd(int x , int y){

if(x > y) swap(x , y);

if(!x) return y;

return gcd(y % x , x);

}

### 扩展欧几里得

**//求解形如的方程，要求**

int solveinv(int a , int &x1 , int b , int &x2){

if(!a){

x1 = 0 , x2 = 1;

return b;

}

int y1 , y2;

int g = solveinv(b % a , y1 , a , y2);

x2 = y1;

x1 = y2 - (b / a) \* x2;

return g;

}

**//求解，并使在范围内**

int getinv(int a , int p){

int x1 , x2;

int t = solveinv(a , x1 , p , x2);

if(t != 1) return -1; //if gcd(a,p) != 1, then x not exist

if(x1 < 0) x1 += p;

return x1;

}

### 类欧几里得算法

**//求，直角三角形下有多少个整数格子**

ll cal(ll a , ll b , ll c , ll n){

if(a == 0) return (b / c) \* (n + 1);

if(a >= c || b >= c)

return cal(a % c , b % c , c , n) + (a / c) \* n \* (n + 1) / 2 + (b / c) \* (n + 1);

ll m = (a \* n + b) / c , v = cal(c , c - b - 1 , a , m - 1);

return n \* m - v;

}

## 中国剩余定理

**//两个同余方程的合并**

typedef pair<int , int> pii; **//同余方程用pair存储，形如x=first(mod second)**

### 模数互质

pii crt1(pii e1 , pii e2){

pii w;

w.second = e1.second \* e2.second;

w.first = (1ll \* e1.first \* e2.second % w.second \* getinv(e2.second , e1.second) +

1ll \* e2.first \* e1.second % w.second \* getinv(e1.second , e2.second)) % w.second;

return w;

}

### 模数不互质

**//int范围内**

pii crt2(pii e1 , pii e2){

pii w;

int z = gcd(e1.second , e2.second) , t = e2.first - e1.first;

if(t % z != 0)

exit(1); **//此同余方程无解**

int p1 = e1.second / z , p2 = e2.second / z , x , y;

t /= z;

solveinv(p1 , x , p2 , y);

w.second = e1.second / z \* e2.second;

w.first = (1ll \* x \* e1.second % w.second \* t + e1.first) % w.second;

if(w.first < 0) w.first += w.second;

return w;

}

**//long long范围内**

pii crt2(pii e1 , pii e2){

pii w;

ll z = gcd(e1.second , e2.second) , t = e2.first - e1.first;

if(t % z != 0)

exit(1); **//无解**

ll p1 = e1.second / z , p2 = e2.second / z , x , y;

t /= z;

solveinv(p1 , x , p2 , y);

if(x < 0) x += p2;

w.second = e1.second / z \* e2.second; t = add(t , w.second , w.second);

w.first = add(mul(mul(x , e1.second , w.second) , t , w.second) , e1.first , w.second);

return w;

}

## 从拓展卢卡斯定理到卢卡斯定理

### 拓展卢卡斯定理

拓展卢卡斯定理是在不是质数但是将质因数分解后得到 之后，都很小的情况下  
解决的一种方法。

由于各个，最后再用中国剩余定理合并。

考虑到，我们可以分别求各个的阶乘的答案，再合并起来求。

#### 算法过程

假设当前考虑的是

由于阶乘中有很多数有p这个因子，那么我们得先把p这个因子提出来

拿下面的数列做例子，当前考虑的是3^2，我们把3的倍数筛出来变成了

显然可以发现，在同一层个构成了一个循环，且每乘的倍数，内部的也是和第一层类似且循环的。

**统计是的倍数的p的幂次方之和**

对于 以内是的倍数的p的幂次方之和，可以得到至少是1次幂之和， 以此类推即可

**统计不是p的倍数的乘积**

每一层统计循环的乘积，统计循环了多少次，再统计单独的即可。

#### 代码

#include<iostream>  
#include<stdio.h>  
#include<string.h>  
#include<algorithm>  
#define maxn 100005  
using namespace std;  
typedef long long ll;  
ll p,n,m,tot;  
ll quickpow(ll x,ll k,const ll mod)  
{  
 ll ans=1;  
 while(k)  
 {  
 if(k&1)  
 ans=(ans\*x)%mod;  
 x=(x\*x)%mod;  
 k>>=1;  
 }  
 return ans;  
}  
void exgcd(ll a,ll b,ll &x,ll &y)  
{  
 if(!b)  
 x=1,y=0;  
 else  
 exgcd(b,a%b,y,x),y-=a/b\*x;  
}  
ll getinv(ll a,ll mod)  
{  
 if(!a) return 0;  
 ll x,y;  
 exgcd(a,mod,x,y);  
 x=(x%mod+mod)%mod;  
 return !x ? mod : x;  
}  
ll pri[maxn],mi[maxn];  
void fenjie(ll now)  
{  
 ll temp=now;  
 for(ll i=2;i\*i<=now;i++)  
 {  
 if(temp%i==0)  
 {  
 pri[++tot]=i; mi[tot]=1;  
 while(temp%i==0)  
 mi[tot]=(mi[tot]\*i)%p,temp/=i;  
 }  
 }  
 if(temp!=1)  
 pri[++tot]=temp,mi[tot]=temp;  
}  
ll cal(ll now,ll p1,ll mod)  
{  
 if(!now) return 1;  
 ll ans=1;  
 if(now/mod)  
 {  
 for(int i=2;i<=mod;i++)//这里可以改成前缀和的形式  
 if(i%p1)  
 ans=(ans\*i)%mod;  
 ans=quickpow(ans,now/mod,mod);  
 }  
 for(int i=2;i<=now%mod;i++)//这里可以改成前缀和的形式   
 if(i%p1)  
 ans=(ans\*i)%mod;  
 return (ans\*mul(now/p1,p1,mod))%mod;  
}  
ll c(ll n,ll m,ll p1,ll mod)  
{  
 if(m>n) return 0;  
 ll a=cal(n,p1,mod),b=cal(m,p1,mod),c=cal(n-m,p1,mod);  
 ll k=0;  
 for(ll i=n;i;i/=p1) k+=i/p1;  
 for(ll i=m;i;i/=p1) k-=i/p1;  
 for(ll i=(n-m);i;i/=p1) k-=i/p1;  
 return a\*getinv(b,mod)%mod\*getinv(c,mod)%mod\*quickpow(p1,k,mod)%mod;  
}  
ll china(ll n,ll m)  
{  
 ll ans=0;  
 for(int i=1;i<=tot;i++)  
 {  
 ll res=c(n,m,pri[i],mi[i]);  
 res=(res\*(p/mi[i])%p);  
 res=(res\*getinv(p/mi[i],mi[i]))%p;  
 ans=(ans+res)%p;  
 }  
 return ans;  
}  
int main()  
{  
 scanf("%lld%lld%lld",&n,&m,&p);  
 fenjie(p);  
 ll ans=china(n,m);  
 return printf("%lld\n",ans),0;  
}

### 卢卡斯定理

为什么先拓展再本身呢，因为卢卡斯定理和拓展卢卡斯好像没什么肉眼可见的关系

做法很简单，就每次将递归后的值乘上对当前n，m取模之后求组合数的值。

#include<iostream>  
#include<stdio.h>  
#include<string.h>  
#include<algorithm>  
#define maxn 200005  
using namespace std;  
typedef long long ll;  
ll n,m,p;  
ll quickpow(ll pp,ll k)  
{  
 ll ans=1;  
 while(k)  
 {  
 if(k&1)  
 ans=(ans\*pp)%p;  
 pp=(pp\*pp)%p;  
 k>>=1;  
 }  
 return ans;  
}  
ll fac[maxn],inv[maxn];  
ll c(ll n1,ll m1)  
{  
 if(n1<m1) return 0;  
 return fac[n1]\*inv[m1]%p\*inv[n1-m1]%p;  
}  
ll lucas(ll n1,ll m1)  
{  
 if(n1<m1) return 0;  
 if(!n1) return 1;  
 return c(n1%p,m1%p)\*lucas(n1/p,m1/p)%p;  
}  
int main()  
{  
 int \_;  
 scanf("%d",&\_);  
 while(\_--)  
 {  
 scanf("%lld%lld%lld",&n,&m,&p);  
 fac[0]=1;  
 for(int i=1;i<=p-1;i++)  
 fac[i]=(fac[i-1]\*i)%p;  
 inv[p-1]=quickpow(fac[p-1],p-2);  
 for(int i=p-1;i>=1;i--)  
 inv[i-1]=(inv[i]\*i)%p;  
 inv[0]=1;  
 printf("%lld\n",lucas(n+m,m));  
 }  
}

## 高次同余方程

### 指数型方程（BSGS）

//求，其中

class Hash{ **//哈希表**

private:

static const int M = 99989 , N = 1e6; **//M是哈希模数，N是插入的元素**

int first[M + 12] , nx[N + 10];

int id[N + 10] , tot;

public:

Hash(){

for(int i = 0 ; i < M ; ++ i)

first[i] = 0;

tot = 0;

}

void insert(int v , int x){

int pos = v % M;

++ tot; nx[tot] = first[pos];

first[pos] = tot;

id[tot] = v;

}

bool find(int v){

int pos = v % M;

for(int x = first[pos] ; x ; x = nx[x])

if(id[x] == v)

return 1;

return 0;

}

void clear(){

tot = 0;

}

void undo(int v){

int pos = v % M;

first[pos] = 0;

}

}S;

int p;

int BSGS(int a , int b , int p){

int m = (int)sqrt((double)p) , r = fpw(a , p - 2) , v = b;

for(int i = 0 ; i < m ; ++ i , v = 1ll \* v \* r % p)

S.insert(v , i);

int u = 1 , ans = -1;

v = fpw(a , m);

for(int i = 0 ; i \* m < p ; ++ i , u = 1ll \* u \* v % p)

if(S.find(u) == 1){

for(int k = 0 ; k < m ; ++ k , u = 1ll \* u \* a % p){

if(u == b){

ans = i \* m + k;

break;

}

}

break;

}

v = b;

for(int i = 0 ; i < m ; ++ i , v = 1ll \* v \* r % p)

S.undo(v);

return ans; **//如果返回-1则表示无解**

}

### 幂函数方程

**//求**

int G; **//G是p的原根**

int cal(int a , int b , int p){

int c = BSGS(G , b , p) , d = gcd(p - 1 , a);

if(c % d != 0) return -1;

int A = a / d , B = (p - 1) / d , C = c / d;

int x1 , x2;

solveinv(A , x1 , B , x2);

x1 = 1ll \* x1 \* C % B;

if(x1 < 0) x1 += B;

return fpw(G , x1);

}

## 莫比乌斯反演

暂时有问题无法上传

## 大质数探测与分解

### MillerRabin

ll add(ll a, ll b, ll p){ **//加法**

return a + b >= p ? a + b - p : a + b;

}

ll mul(ll a, ll b, ll p){ **//乘法**

ll c = 0;

a %= p;

while(b){

if(b & 1)

c = add(c , a , p);

a = add(a , a , p);

b >>= 1;

}

return c;

}

ll fpw(ll o, ll t, ll p){ **//乘方**

ll c = 1;

while(t){

if(t & 1)

c = mul(c , o , p);

o = mul(o , o , p);

t >>= 1;

}

return c;

}

ll rand(ll s, ll t){ **//随机数生成器**

return rand() % (t - s + 1) + s;

}

bool testprime(ll v){ **//判断v是不是质数，是返回1**

if(v <= 2) return v - 1;

else if(!(v & 1)) return 0;

ll d = v-1 , x , y;

while(!(d & 1))

d >>= 1;

for(int i = 1 ; i < 10 ; ++ i){ **//这里为了准确度可以提高一下i的最大值**

x = fpw(rand(1 , v - 1) , d , v);

ll d1 = d;

while(d1 != v-1){

y = mul(x , x , v);

if(y == 1 && x != 1 && x != v - 1)

return 0;

x = y;

d1 <<= 1;

}

if(y != 1) return 0;

}

return 1;

}

### Pollardrho

ll h[300]; int tot = 0; **//h数组存分解后的所有质因数，tot是质因数个数**

ll gcd(ll x , ll y){

return !x ? y : gcd(y % x , x);

}

ll w1,w2;

ll nx(ll x){

return add(w1 , mul(x , x , w2) , w2);

}

ll Pollar(ll v){

int check = 0;

w2 = v;

while(++ check <= 10000){

w1 = rand(1 , v - 1);

ll x = rand(1 , v - 1) , y = nx(x);

while(true){

ll t = gcd(y - x + v , v);

if(t == v) break;

else if(t > 1)

return t;

x = nx(x); y = nx(nx(y));

}

}

return 0;

}

void find(ll v){

if(testprime(v)){

h[++ tot] = v;

return ;

}

if(v % 2 == 0){

h[++ tot] = 2;

find(v / 2);

return ;

}

ll u = Pollar(v);

if(!u) assert(1);

find(u);

find(v / u);

}

## 扩展欧拉定理

**//这里面是欧拉函数，的求法可以用大质数分解，也可以用筛法。**

## Polya和Burnside

Polya定理：，其中是操作群，是一个群对序列操作后划分的等价类数量。

Burnside引理：，其中v是排列，相当于求在下的不动点数量。

## 容斥原理

## 齐次线性递推

## 生成函数

## 二次剩余

## 特殊数列

### 组合数

#### 递推预处理

const int N = 2020 , p = 998244353; **//p不一定是质数**

int C[N][N];

void init(){

C[0][0] = 1;

for(int i = 1 ; i < N ; ++ i){

C[i][0] = C[i][i] = 1;

for(int j = 1 ; j < i ; ++ j)

C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % p;

}

}

#### 阶乘预处理

const int N = 1e6+10 , p = 998244353;

int fac[N] , ifac[N];

int C(int n , int m){

if(m < 0 || m > n) return 0;

return 1ll \* fac[n] \* ifac[m] % p \* ifac[n - m] % p;

}

void init(){

fac[0] = fac[1] = 1;

for(int i = 2 ; i < N ; ++ i)

fac[i] = 1ll \* fac[i - 1] \* i % p;

ifac[0] = ifac[1] = 1;

for(int i = 2 ; i < N ; ++ i)

ifac[i] = p - 1ll \* (p / i) \* ifac[p % i] % p;

for(int i = 2 ; i < N ; ++ i)

ifac[i] = 1ll \* ifac[i - 1] \* ifac[i] % p;

}

### 卡特兰数

#### 表达式

#### 组合意义

1.长度为2n的括号序列个数。

2.n个节点二叉树的形态数。

3.对凸n+2边形进行不同的三角形分割的方案数。

### 斯特林数

#### 第一类斯特林数

#### 第二类斯特林数

### 斐波那契数列

#### 表达式

#### 通项计算

##### 矩阵乘法

构造矩阵，直接进行矩阵快速幂即可，复杂度

##### 质数域（5是p的二次剩余）

求出使得，这样就变成正常数列求和了，复杂度。

#### 皮萨诺周期

斐波那契数列在模p意义下周期是皮萨诺周期。

当p是质数时，

n是正整数时，()，其中

#### 齐肯多夫定理

**任何正整数都可以表示为若干项不同且不相邻的斐波那契数之和，且表示方法唯一。**

即构成了一个齐肯多夫拉系

，并且不存在的情况。

### 分拆数

利用五边形数和生成函数，得到n个数拆成若干个整数的方案数（数之间没有区别且可以相同）

const int maxn = 1e5 + 10 , p = 1e9 + 7;

int f[maxn]; **//f(n)表示的就是分拆数**

void init(){

f[0] = 1;

for(int i = 1 ; i <= 100000 ; ++ i){

for(int k = 1 ; k <= i ; ++ k){

int u = k \* (3 \* k - 1) / 2;

if(u <= i)

f[i] = (f[i] + (k & 1 ? f[i - u] : p - f[i - u])) % p;

u = k \* (3 \* k + 1) / 2;

if(u <= i)

f[i] = (f[i] + (k & 1 ? f[i - u] : p - f[i - u])) % p;

else

break;

}

}

for(int i = 1 ; i <= 20 ; ++ i)

printf("%d\n" , f[i]);

}

## 一些结论

### LGV引理（统计不相交路径数量）

有n个点，他们分别从点出发，终点是，走出的路径互不相交的方案数是

这是个行列式，其中表示做为起点，做为终点的方案数。