**Pdev数论模板**

**Ver 2.0 Alpha**

**Content：**

**---------数论部分----------**

**0. 几个常用tips**

**1. 同余**

**2. 逆元**

**3. CRT（解同余方程组）**

**4. gcd\extend\_gcd（解同余方程、解ax+by=c）**

**5. 欧拉函数**

**6. 唯一分解定理**

**7. 快速幂取模及一个奇怪的大数据算法**

**8. O(sqrt(n))的时间枚举n的所有约数 --（Testing）**

**9. 莫比乌斯反演**

**10.Miller\_Rabin素数测试**

**11.** **pollard\_rho算法分解质因数**

**---------矩阵部分-------------**

1. **矩阵快速幂**
2. **高斯消元**

**---------博弈论部分------------**

**---------概率论部分------------**

# A.数论部分

### 0.几个常用tips

**1. gcd(x,n)=i的充要条件是gcd(x/i,n/i)=1**

**2. gcd(a,b)=1 –> a和b互质 -> 可以想办法套欧拉函数 -> 满足条件的a有phi(b)个**

**3. 如果找n的每一个约数i会有点慢，可以枚举i，令n=2\*i，3\*i，........（n是i的所有倍数且小于MAXN）**

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

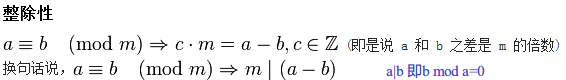
　 　for (int n=i\*2;n<=MX;n+=i)

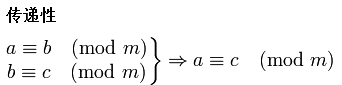
　　 　　f[n]=。。。。。（i是n的约数）

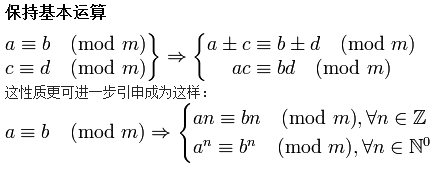
## 1.同余

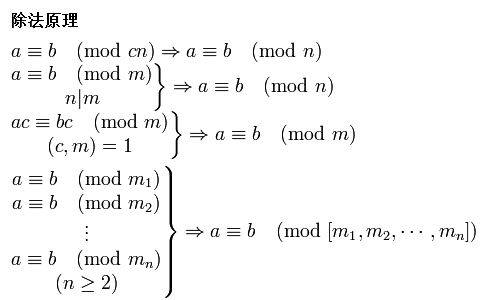
**同余：a≡b (mod m)，表示a % m==b % m**

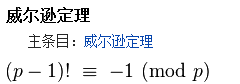
**同余式的运算法则：**

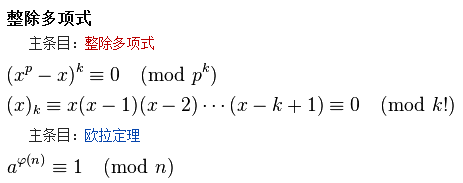


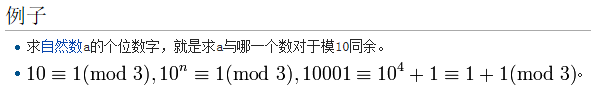












## 2.逆元

**HDU 1576**

**逆元的定义：例如a\*b≡1 (mod m)，则b就是a关于m的逆元。**

**若a和b互为逆元，则(x/a) mod m=(x\*b) mod m**

**同余方程中，若a\*x≡b (mod n)，左乘a的逆元A，得x≡A\*b(mod n)**

**求逆元方法也很简单，用扩展欧几里得解这个方程即可。**

void calc\_inv(LL n) //求1--n关于MOD的逆元，inv[i]=i的逆元

{

inv[1] = 1;

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

{

if(i >= MOD) break;

inv[i] = (MOD - MOD / i) \* inv[MOD % i] % MOD;

}

}

## 3. 中国剩余定理

**POJ 1006**

**中国剩余定理：求解同余方程组**

**n≡a[1] (mod m[1])**

**n≡a[2] (mod m[2])**

**......**

**n≡a[i] (mod m[i])**

**该方程组有解的条件：m[i]两两互质**

int CRT(int a[],int m[],int n)

//中国剩余定理，返回方程x≡a[1..n] (mod m[1..n])的解x

{

int M=1;

for (int i=1;i<=n;i++) M\*=m[i];

int ret=0;

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

{

int x,y;

int tm=M/m[i];

extend\_gcd(tm,m[i],x,y);

ret=(ret+tm\*x\*a[i])%M;

}

return (ret+M)%M;

}

**补充：扩展中国剩余定理**

**用于计算方程组n≡a[i] (mod m[i])的一个特解，其中不要求所有的m[i]两两互质。此时用的方法是方程两两合并的方法**

//POJ2891

\_\_int64 extend\_gcd(\_\_int64 a,\_\_int64 b,\_\_int64 &x,\_\_int64 &y){

if (b==0){

x=1;y=0;

return a;

}

else{

\_\_int64 r=extend\_gcd(b,a%b,y,x);

y=y-x\*(a/b);

return r;

}

}

\_\_int64 gcd(\_\_int64 a,\_\_int64 b)

{

if (b==0) return a;

return gcd(b,a%b);

}

\_\_int64 lcm(\_\_int64 a,\_\_int64 b)

{

\_\_int64 tm=gcd(a,b);

if (tm==0) return 0;

else return (a\*b/tm);

}

LL CRT\_ex(LL n,LL a[],LL m[]) //n==a[i](mod m[i])

{ //n%m[i]==a[i]

bool sol=true;

LL X,Y,G,C,C1;

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

{

LL a1=a[i-1],m1=m[i-1],a2=a[i],m2=m[i];

G=extend\_gcd(a1,a2,X,Y);

G=gcd(a1,a2);

C=m2-m1;

if (C%G!=0)

{ sol=false; return -1; }

X=X\*(C/G); Y=Y\*(C/G);

LL rq=a2/G; //xi的通解公式 xi=x1+k\*(b/d)

X=(X%rq+rq)%rq; //需要最小非负解

C1=a1\*X+m1;

m[i]=C1; a[i]=lcm(a1,a2);

}

if (sol==true) return C1; else return -1;

}//-1：无解

## 4.欧几里得：gcd、extern\_gcd

**POJ 1061、2142、2891（迭代解同余方程组）**

**对于同余方程A≡B (mod L)，**

**令A=x\*L+m，B=y\*L+m，稍微YY下即可转化为x\*L-y\*L=A-B，即ax+by=c的形式。**

**对于方程ax+by=c**

**设G=gcd(a,b)**

**若c%G!=0，则该方程无整数解。STOP**

**否则，先用扩展欧几里得求出方程ax+by=G的解x0、y0**

**那么有a\*x0+b\*y0=G**

**令x1=x0\*(c/G)，y1=y0\*(c/G)**

**则a\*x1+b\*y1=c**

**x1、y1即原方程的一个特解**

**这个方程的通解：xi=x1+k\*(b/G)，yi=y1-k\*(a/G)**

**对于不定方程ax+by=c**

**用类似的方法易求出一个特解x1、y1**

**那么通解xi=x1+k\*(b/d)，yi=y1-k\*(a/d) 其中d=gcd(a,b)**

**另：如果要求yi的最小非负解？令r=a/tm，则解y2=(y1%r+r)%r**

**其他类似的要求自己用纸算一算，YY一下就行了**

**求使……最大/最小的解：一般把……构造成一个函数，讨论单调性**

**因为方程中左边有两个数a、b，讨论的时候为简化问题可以人为规定a>b或a<b**

**输入输出的时候别忘了相应swap一下就行了~**

int gcd(int a,int b){ //辗转相除法，返回gcd(a,b)

if (b==0) return a;

return gcd(b,a%b);

}

int extend\_gcd(int a,int b,int &x,int &y){

//扩展欧几里得，求解a\*x+b\*y=gcd(a,b)，返回值为gcd(a,b)

if (b==0){

x=1;y=0;

return a;

}

else{

int r=extend\_gcd(b,a%b,y,x);

y=y-x\*(a/b);

return r;

}

}

## 5.欧拉函数

**关于gcd的8题.txt**

**·欧拉函数：一般记作φ(n)，表示1-n中与n互质的数的数量。**

**·欧拉函数是积性函数，即φ(m\*n)=φ(m)\*φ(n)**

**//这条定理基友面试时还遇到了= =**

**·欧拉函数的值φ(n)=n\*(1-p[1])\*(1-p[2])\*...\*(1-p[n])**

**//p[i]是小于等于n的所有素数**

**·若n是m的倍数，则小于等于n且与m互质的数的个数为(n/m)\*φ(m)**

**//证明不难理解：设k小于等于m且与m互质，则k+m、k+2m......也与m互质**

**·若n是质数p的k次幂，则φ(n)=(p-1)\*(p^(k-1))**

**//关于欧拉函数wikipedia上讲的很详细，此处不赘述了**

void calc\_phi(int n) //求1--n的欧拉函数，phi[i]=φ(i)

{

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

phi[i]=0;

phi[1]=1;

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

if (!phi[i])

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

{

if (!phi[j]) phi[j]=j;

phi[j]=phi[j]/i\*(i-1);

}

}

long long euler(long long n) //返回phi(n)

{

long long ret = n;

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

if (n % i == 0)

{

ret = ret / i \* (i - 1);

while (n % i == 0)

n /= i;

}

if (n > 1)

ret = ret / n \* (n - 1);

return ret;

}

## 6.整数的唯一分解定理：

**POJ 1845**

**任意正整数都有且只有一种方式写出其素因子的乘积表达式。**

**A=(p1^k1)\*(p2^k2)\*(p3^k3)\*....\*(pn^kn) 其中pi均为素数**

## 7.快速幂取模

**就是(A^B) mod MOD啦**

LL pow\_mod(LL a,LL b)

{

if (a==1) return 1;

LL t=a,ans=1;

while(b)

{

if (b&1)

ans=ans\*t%MOD;

t=t\*t%MOD;

b>>=1;

}

return ans;

}

||||-----------萌萌哒分割线-------------||||

//这个是给可能超出long long范围的变态大数据用的

//hdu5183

LL func(LL a,LL b,LL c) //a\*b%c

{

long long ret = 0;

while (b)

{

if (b & 1)

ret = (ret + a) % c;

a = 2 \* a % c;

b >>= 1;

}

return ret;

}

LL pow\_mod(LL a,LL b,LL MOD)

{

if (a==1) return 1;

LL t=a%MOD,ans=1;

while(b)

{

if (b&1)

ans=func(ans,t,MOD);

t=func(t,t,MOD);

b>>=1;

}

return ans;

}

## 8. O(sqrt(n))的时间枚举n的所有约数 --（Testing）

注意一个性质：若n%i==0,则有n%(n/i)=0

for (int i=1;i\*i<=N;i++)

　　if (N%i==0)

　　{

　　　　//i是约数，N/i也是约数

　　　　balabalabala...

　　}

## 10.Miller\_Rabin素数测试

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Miller\_Rabin算法进行素数测试

\*速度快，可以判断一个 < 2^63的数是不是素数

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int S = 8; //随机算法判定次数，一般8~10就够了

//计算ret = (a\*b)%c

long long mult\_mod(long long a,long long b,long long c)

{

a %= c;

b %= c;

long long ret = 0;

long long tmp = a;

while(b)

{

if(b & 1)

{

ret += tmp;

if(ret > c)ret -= c;//直接取模慢很多

}

tmp <<= 1;

if(tmp > c)tmp -= c;

b >>= 1;

}

return ret;

}

//计算 ret = (a^n)%mod

long long pow\_mod(long long a,long long n,long long mod)

{

long long ret = 1;

long long temp = a%mod;

while(n)

{

if(n & 1)ret = mult\_mod(ret,temp,mod);

temp = mult\_mod(temp,temp,mod);

n >>= 1;

}

return ret;

}

//通过 a^(n-1)=1(mod n)来判断n是不是素数

// n-1 = x\*2^t中间使用二次判断

//是合数返回true,不一定是合数返回false

bool check(long long a,long long n,long long x,long long t)

{

long long ret = pow\_mod(a,x,n);

long long last = ret;

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

{

ret = mult\_mod(ret,ret,n);

if(ret == 1 && last != 1 && last != n-1)return true;//合数

last = ret;

}

if(ret != 1)return true;

else return false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Miller\_Rabin(long long n)

//是素数返回true,(可能是伪素数)

//不是素数返回false

{

if( n < 2)return false;

if( n == 2)return true;

if( (n&1) == 0)return false;//偶数

long long x = n - 1;

long long t = 0;

while( (x&1)==0 )

{

x >>= 1;

t++;

}

rand();/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

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

{

long long a = rand()%(n-1) + 1;

if( check(a,n,x,t) )

return false;

}

return true;

}

## 11. pollard\_rho算法分解质因数

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// pollard\_rho算法进行质因素分解

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

long long factor[100];//质因素分解结果（刚返回时是无序的）

int tol;//质因素的个数，编号0~tol-1

long long gcd(long long a,long long b)

{

long long t;

while(b)

{

t = a;

a = b;

b = t%b;

}

if(a >= 0)return a;

else return -a;

}

//找出一个因子

long long pollard\_rho(long long x,long long c)

{

long long i = 1, k = 2;

srand(time(NULL));

long long x0 = rand()%(x-1) + 1;

long long y = x0;

while(1)

{

i ++;

x0 = (mult\_mod(x0,x0,x) + c)%x;

long long d = gcd(y - x0,x);

if( d != 1 && d != x)return d;

if(y == x0)return x;

if(i == k)

{

y = x0;

k += k;

}

}

}

//对 n进行素因子分解，存入factor. k设置为107左右即可

void findfac(long long n,int k)

{

if(n == 1)return;

if(Miller\_Rabin(n))

{

factor[tol++] = n;

return;

}

long long p = n;

int c = k;

while( p >= n)

p = pollard\_rho(p,c--);//值变化，防止死循环k

findfac(p,k);

findfac(n/p,k);

}

使用方法：（要对n分解质因数）

tol = 0;

findfac(n,107);

//此时factor[0..tol-1]存放的就是所有质因数（无序）

# B．矩阵部分

## 0．矩阵快速幂

typedef unsigned long int ULL;

typedef vector<ULL> vec;

typedef vector<vec> mat;

const ULL P=9973;

mat mul(mat &A,mat &B) //return A\*B

{

mat C(A.size(),vec(B[0].size()));

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

{

for (int k=0;k<(int)B.size();k++)

{

for (int j=0;j<(int)B[0].size();j++)

{

C[i][j]=(C[i][j]+A[i][k]\*B[k][j])%P;

}

}

}

return C;

}

mat m\_pow(mat A,int m) //return A^m

{

mat B(A.size(),vec(A.size()));

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

B[i][i]=1;

while (m>0)

{

if (m&1) B=mul(B,A);

A=mul(A,A);

m>>=1;

}

return B;

}

int main()

{

cin>>n>>m;

mat A(n,vec(n));

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

for (int j=0;j<n;j++)

cin>>A[i][j];

A=m\_pow(A,m);

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

{

for (int j=0;j<n;j++)

cout<<A[i][j]<<" ";

cout<<endl

}

return 0;

}

## 1. 高斯消元

**USACO ratios**

void solve(Matrix A,int n) //A为增广矩阵，n维

{

int i,j,k,r;

for (i=0;i<n;i++)

{

r=i;

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

if (fabs(A[j][i])>fabs(A[r][i]))

r=j;

if (r!=i)

for (j=0;j<=n;j++)

swap(A[r][j],A[i][j]);

for (k=i+1;k<n;k++)

{

double f=A[k][i]/A[i][i];

for (j=i;j<=n;j++)

A[k][j]-=f\*A[i][j];

}

}

for (i=n-1;i>=0;i--)

{

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

A[i][n]-=A[j][n]\*A[i][j];

A[i][n]/=A[i][i];

}

}