头文件

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <time.h>
#include <set>
#include <map>
#include <stack>
```

```
#include <queue>
#include <string>
#include <bitset>
#include <vector>
#include <deque>
#include <list>
#include <sstream>
#include <iostream>
#include <functional>
#include <numeric>
#include <algorithm>
using namespace std;
template<class T>inline T iabs(const T& v) {return v<0 ? -v : v;}
template<class T>inline T strTo(string s){istringstream is(s);T v;is>>v;return
v;}
template<class T>inline string toStr(const T& v){ostringstream os;os<<v;return
os.str();}
template<class T>inline int cMin(T& a, const T& b){return b<a?a=b,1:0;}
template<class T>inline int cMax(T& a, const T& b){return a<b?a=b,1:0;}
template<class T>inline int cBit(T n){return n?cBit(n&(n-1))+1:0;}
#define ep 1E-10
#define CLR(arr,v)   memset(arr, v, sizeof(arr))
#define SQ(a)  ((a)*(a))
#define DEBUG(a)     printf("%s = %s\n", #a, toStr(a).c_str())
#define FOR(i,s,e) for( u64  (i)=(s); (i) < (e) ; i++)
Const u64 inf3 = 0x15fffffffffffffll;
int inf4 = 0x7fffffffl;
typedef long long u64;
```

求素数[1,n],返回素数个数

```
bool is[1000010];//用来求素数[1,130000]
int prm[100000];//用来保存素数
int totleprm;//记录素数总个数
void getprm(int n)
{
    int i;
    memset(is,1,sizeof(is));
    is[1]=0;
    prm[totleprm++]=2;
    for (i=4; i<=n; i+=2) is[i]=0;
    for (i=3; i*i<=n; i+=2)
    if (is[i])
    {
        prm[totleprm++]=i;
        for (int s=2*i,j=i*i; j<=n; j+=s)
        is[j]=0;
    }
    for (; i<=n; i++)
    if (is[i]) prm[totleprm++]=i;
}
```

得到区间 [a ,b] 之间的素数

注意：需要前 6000 个素数 （一般）

```
    int subprm[100010];
    int getSubPrm(u64 a,u64 b)
```

```
    {
        u64 totl = 0,i,j;
        bool sign[1000010];
        memset(sign,true,sizeof(sign));
        if (a < 2) a = 2;
        u64 l = b - a + 1;
        for ( i=0; i<totleprm;i++)
        {
            if ((j=prm[i]*(a/prm[i]))<a) j += prm[i];
            if (j<prm[i]*prm[i]) j = prm[i] * prm[i];
            for ( ; j<=b ; j+=prm[i]) sign[j-a] = false;
        }
        for (i=0;i<l;i++)
        if (sign[i]) subprm[totl++] = a + i;
        return totl; }
```

求 a*b%c
要求:a,b 的范围在 hugeint 范围的一般以内，在 hugeint 为 unsigned __int64 时，a,b 需要是__int64 能表示的数

```
    u64  power_mod(u64 A, u64  B, u64 C)
{
      u64  R = 1, D = A;
    while (B )
    {
        if (B&1) R = (R*D)%C;
        D = (D*D)%C;
        B >>=1;
    }
    return R;
}
```

求 a^b%c
要求:a,b 的范围在__int64 范围的一般以内，在__int64 为 unsigned __int64 时，a,b 需要是__int64 能表示的数

```
    u64  power_mod(u64 A, u64  B, u64 C)
    {
        u64  R = 1, D = A;
        while (B )
        {
            if (B&1) R = product_mod(R, D, C);
            D = product_mod(D, D, C);
            B >>=1;
        }
        return R;
    }
```

求 x 的欧拉函数值
注意：需要素数表

```
u64  euler(u64  x)
{
    int i ;
    u64  res = x;
    for (i = 0; prm[i] < (u64 )sqrt(x * 1.0) + 1 && i < totleprm; i++)
    if (x % prm[i] == 0)
    {
```

```
        res = res / prm[i] *( prm[i] - 1);

        while (x % prm[i] == 0 ) x/=prm[i];
        }
if (x > 1) res = res / x * (x-1);
    return res;
    }
```

求 x 的欧拉函数值

```
    u64 phi[1000001];
    void euler(int maxn)
    {
        int i;
        int j;
        for (i = 2; i <= maxn; i++)
        phi[i] = i;
        for (i = 2; i <= maxn; i+=2)
        phi[i] /=2;
        for (i = 3; i <= maxn; i+=2)
        if (phi[i] == i)
        {
            for (j = i; j<=maxn; j+=i)
            phi[j] = phi[j] / i * (i-1);
        }
    }
```

快速求欧拉函数和素数。
注意：返回[1,maxn]中所有素数和每个数字的欧拉函数。
注意：prime[0] 存储区间内素数个数，prime[1] = 2;

```
u64 phi[150010];
char ok[150010]={0};
u64 prm[150010]={0};
u64 sum[150010]={0};
int totleprm;
int eulerAndPrm(int maxn)
{
    u64 i,j;
    int totleprm=0;
    phi[1]=1;
     for(i=2;i<=maxn;i++)
     {
      if(ok[i]==0)
      {
       prm[totleprm++]=i;1
       phi[i]=i-1;
      }
      for(j=0;j<totleprm && prm[j]*i<= maxn;j++)
      {
       ok[prm[j]*i]=1;
       if(i%prm[j]==0)
       {
         phi[i*prm[j]]=phi[i]*prm[j];
         break;
```

```
      }
      else
        phi[i*prm[j]]=phi[i]*(prm[j]-1);
     }
    }
    return totleprm;
}
```

求解 a * x = b mod n
注意：解存在数组 ans[] 中。 返回解的个数

```
int modeq(u64 a, u64 b, u64 n, u64 ans[])
{
    u64 e, i, d, x, y;
    d = extgcd(a, n, x, y);
    if (b % d >0) return 0;
    else
    {
        e =  (x * (b / d)) %(n/d);
        if (e < 0) e +=n/d;
        for(i=0; i<1; i++)
            ans[i] = (e + (i * (n/d) ))%(n);
        return d;
    }
}
```

求解 gcd (a,b) = a * x + b * y (切忌 unsigned)

```
u64 extgcd(u64 a,u64 b,u64 &x,u64 &y)
{
    if (b == 0) {x=1;y=0;return a;}
    u64 d = extgcd(b, a%b,x,y);
    u64 t = x; x = y; y = t - a/b * y;
    return d;
}
```

x = a[i] mod m[i] .
求解 x.无解时返回-1.

```
u64 china(int k, u64 a[],u64 m[])
{
    bool flag = false;
    u64 e, x, y, i,d;
    u64 result;
    u64 a1,m1;
    u64 a2,m2;
    m1 = m[0]; a1 = a[0];
```

```
   FOR(i,1,k)
    {
        m2 = m[i]; a2 = a[i];
        d =  extgcd( m1, m2, x, y );
        if ( (a2-a1) %  d !=   0  )
        flag = 1;
        result =  ( x *  ((a2-a1) /  d ) %  m2 +  m2 ) %  m2;
        a1 = a1 + m1 * result;  //对于求多个方程
        m1 = (m1 * m2) / d;     //lcm(m1,m2)最小公倍数；
        a1 = (a1 % m1 + m1) % m1;
    }
    if (flag)
    return -1;
    else
    return a1;
}
```

二分计算(p^0 + p^1 + p^2 + p^3 ... p^n) mod m

```
u64 getSum(u64 p, u64 n, u64 m)
{
    if (n == 1)
    return (p%m);
    u64 k = getSum(p,(n)/2,m);
    if (n % 2 == 0)
    {
        u64 k_help = power_mod(p,n/2,m);
        return (k+product_mod(k_help,k,m) )%m;
    }
    else
    if (n % 2 == 1)
    {
        u64 k_help = power_mod(p,n/2+1,m);
       return (k+product_mod(k_help,k,m)+k_help )%m;
    }

}
```

求约数和模板，fac[]存储所求数的因数分解式

```
u64 getDivSum(factor fac[],int n)
{
    u64 ans = 1;
```

```
    FOR(i,0,n)
        {
            ans *= getSum(fac[i].p,fac[i].b,9901);
             ans = ans % 9901;

        }
   return ans;
}
```

给出随机数，可以简单的用 rand()代替

```
__int64 rAndom()
{ __int64 a;
        a  = rand();
        a *= rand();
        a *= rand();
        a *= rand();
        return a;
}
```

rabinmiller 方法测试 n 是否为质数

```
int pri[]={2,3,5,7,11,13,17,19,23,29};
bool isprime(__int64 n)
{
    if(n<2)
        return false;
    if(n==2)
        return true;
    if(!(n&1))
        return false;
    __int64 k = 0, i, j, m, a;
    m = n - 1;
    while(m % 2 == 0)
        m = (m >> 1), k++;
    for(i = 0; i < 10; i ++)
    {
        if(pri[i]>=n)return 1;
        a = power_mod( pri[i], m, n );
        if(a==1)
            continue;
        for(j = 0; j < k; j ++)
        {
            if(a==n-1)break;
            a = product_mod(a,a,n);
        }
        if(j < k)
```

```
            continue;
        return false ;
    }
    return true;
}
```

pollard_rho 分解，给出 N 的一个非 1 因数，
返回 N 时为一次没有找到

```
__int64 pollard_rho(__int64 C, __int64 N)
{
    __int64 I, X, Y, K, D;
    I = 1;
    X = rand() % N;
    Y = X;
    K = 2;
    do
    {
        I++;
        D = gcd(N + Y - X, N);
        if (D > 1 && D < N) return D;
        if (I == K) Y = X, K *= 2;
        X = (product_mod(X, X, N) + N - C) % N;
    }while (Y != X);
    return N;
}
```

找出 N 的最小质因数

```
__int64 rho(__int64 N)
{
    if (isprime(N)) return N;
    do
    {
        __int64 T = pollard_rho(rand() % (N - 1) + 1, N);
        if (T < N)
        {
              __int64 A, B;
              A = rho(T);
              B = rho(N / T);
              return A < B ? A : B;
        }
    }
    while (true);
}
```

用中国剩余定理解同余方程组 a=bi（modni）

```
long solmodu(long z, long b[], long n[])
{
    int i;
```

```
    long a,m,x,y,t;
    m=1 ;a=0;
    for(i=0; i<z; i++)  m*=n[i];
    for(i=0; i<z; i++)
    {
        t=m/n[i];
        exEuclid(n[i],t,x,y);
        a=(a+t*y*b[i])%m;
    }
    return (a+m)%m;
}
```

进制转换

将一个 10 进制数转为 m 进制。

返回转换后数的长度，存在数组 a 中，并且，第 i 位上的数字表示 a[i] * (m ^ i)

```
int changExquisite(int a[],u64 n, int m)
{
    int i = 0;
    while (n > 0)
    {
        a[i++] = n % m;
        n = n / m;
    }
    return i;
}
```

牛顿迭代法求开方

注意：实际是二分

```
double NT_sqrt(double n)
{
    double m = 1;
     while(fabs(m-(m+n/m)/2)>1e-6){
    m=(m+n/m)/2;
    }
    return m;
}
```

字符串 s 表示的数字对一个整数取摸

```
u64 strmod(char *s,u64 t)
{
    u64 sum=0;
       int i,len=strlen(s);
       for(i=0;i<len;i++)
    {
             sum=sum*10+s[i]-'0';
             while(sum>=t)
                    sum-=t;
    }
       return sum;
}
```

得到素数 m 的原根,需要素数表
Phi 为 m 的欧拉函数值
注意：原跟存在条件 2,4,p^m , 2*p^m

```
u64 getPrmRoot(u64 m,u64 phi)
{
    if (m == 2)
    return 1;
    u64 fac[1000];
    u64 n = phi;
    int e = (int)(sqrt(0.0 + n) + 1);
    int length = 0;
    for (int i=0; i<totleprm && prm[i]*prm[i] <= phi; i++)
    if (n % prm[i] == 0)
    {
        fac[length++] = phi/prm[i];
 while ( n % prm[i] == 0) n/=prm[i];
    }
    if (n!=1)
    fac[length++] = phi/n;
    u64 g = 2;
    while (g < m)
    {
        if (gcd(g,m) != 1)
        {g++;continue;}
        bool sign = true;
        FOR(i,0,length)
        if (power_mod(g,fac[i],m) == 1 )
        {
            sign = false;
            break;
        }
        if (sign)
        break;
        g++;
    }
//表示原根不存在
    if (g == m)
    return -1;
    return g;
}
```

满足 gcd(n,i) = 1 并且 i <=m成立的 i 的个数
初始化素数表

```
int getEul(int n , int m)
{
    int factor[100];
    int num = 0;
    int e = (int)(sqrt(0.0 + n) + 1);
    for (int i=0; i<totleprm && prm[i] <= e; i++)
```

```
    if ( n % prm[i] == 0)
    {
        factor[num++] = prm[i];
        while ( n % prm[i] == 0 && n != 1)
        n = n / prm[i];
    }
    if (n != 1)
    factor[num++] = n;
    int ans = 0;
    FOR(i,1,(1<<num))
    {
        int sign = i;
        int lcm = 1;
        int time = 0;
        int j = 0;
        while (sign > 0)
        {
            if (sign % 2 == 1)
            {
                lcm = lcm * factor[j];
                time++;
            }
            sign = sign >> 1;
            j++;
        }
        if (time % 2 == 1)
        ans += m/lcm;
        else
        ans -= m/lcm;
    }
    return m - ans;
}
```

求 A^B

```
    u64  power(u64 A, u64  B)
    {
        u64  R = 1, D = A;
        while (B )
        {
            if (B&1) R = R * D;
            D = D * D;
            B >>=1;
        }
        return R;
    }
```

一个数字的二进制表达式中 1 的个数

```
int getOneNum(u64 n)
{
    int sum = 0
    while (n > 0)
    {
        if (n & 1)
         sum ++;
         n = n >> 1;
    }
    return sum;
}
```

得到一个数字的二进制表达式的第 i 位

```
int isOne(u64 n, int i)
{
    if ((n & (1 << i) ) == 0)
    return 0;
    return 1;
}
```

求第 k 个与 n 互素的数,需要素数表

```
u64 kThPrim(u64 n, u64 k)
{
    u64 nn = n;
    u64 factor[100];
    int num = 0;
    for (int i=0; i<totleprm && prm[i]*prm[i] <= n; i++)
    if (nn % prm[i] == 0)
    {
        factor[num++] = prm[i];
        while (nn % prm[i] == 0 && nn != 1)
        nn/=prm[i];
    }
    if (nn != 1)
    factor[num++] = nn;

    u64 l = 1;
    //这里注意，上界必须足够大
    u64 h = 1000000000;
    u64 m;
    while ( l < h)
    {
        m = ( l + h) >> 1;
        u64 ans = m ;
        FOR(i,1,(1<<num))
        {

            u64 lcm = 1;
            u64 ans_help = 0;
            int sum = 0;
            FOR(j,0,num)
             if ((i & (1 << i) ) != 0)
                {
                    sum++;
                    lcm = lcm * factor[j];
                }
            if (sum & 1 )
            ans -= m / lcm;
            else
            ans += m / lcm;
        }


         if (ans  ==  k)
        {
        while (gcd(m,n) != 1) m--;
        return m;
```

```
        }
        if (ans < k)
        l = m + 1;
        else
        h = m;
    }
    return 0;
}
```

求解 x^2 = a mod (n) 。n 为素数.
注意：有解时.返回较小解 x . 另外一解为 n-x.

```
 u64 ModSqrt(u64 a,u64 n){
    u64 b,k,i,x;
    if(n==2)return a%n;
    if(power_mod(a,(n-1)/2,n)==1){
        if(n%4==3)x=power_mod(a,(n+1)/4,n);
        else{
            for(b=1;power_mod(b,(n-1)/2,n)==1;b++);
            i=(n-1)/2;k=0;
            do{
                i/=2;
                k/=2;

    if( (power_mod(a,i,n)*(u64)power_mod(b,k,n)+1)%n==0 )k+=(n-1)/2;
            }while(i%2==0);
            x=(power_mod(a,(i+1)/2,n)*(u64)power_mod(b,k/2,n))%n;
        }
        if(x*2>n)x=n-x;return x;
    }return -1;
}
```

求解 pell 方程

```
struct Pell
{
llong a,b,c;//a+b*sqrt(c)
Pell():a(0),b(0),c(0){}
Pell(llong aa,llong bb,llong cc){a=aa;b=bb;c=cc;}
Pell operator*(const Pell &p)
{
Pell ret;
ret.a=a*p.a+b*p.b*c;
ret.b=a*p.b+b*p.a;
ret.c=c;
return ret;
}
Pell operator^(int k)
{
if(k==1)return *this;
--k;
Pell ret=*this,tmp=ret;
while(k)
```

```
{
if(k&0x1)
ret=ret*tmp;
tmp=tmp*tmp;
k>>=1;
}
return ret;
}
inline void showans()
{
printf("%10lld%10lld\n",b,(a-1)>>1);
}
};
```

求 pell 方程的第 i 个解

```
/*
a * x ^ 2 – b * y ^ 2 = c;
x1,y1 为方程的一个最小特解.。
x0, y0 为 x ^ 2 – aby^2 = 1 的最小特解
矩阵 unit 为
解为 x,y
*/
void pell(Mat UNIT, int i,u64 x1, u64 y1, u64 &x, u64 &y)
{
      Mat k = po(UNIT,i);
      x = k.matrix[0][0] * x1 + k.matrix[0][1] * y1;
      y = k.matrix[1][0] * x1 + k.matrix[1][1] * y1;
}
```

求解 x^2 - a*b*y^2 = 1。的最小解 (pell)

```
void pell_continue(int n, u64 &x0, u64 &y0)
{
    int con[1000];
```

```
    int num = 0,c;
    double k = sqrt(n);
    con[num++] =  (int)k;
    k = k - (int)k;
    while (1)
    {

        con[num++] = (int)(1 / k);
        k = (1/k)-(int)(1/k);
        x0= 1;
        y0= con[num-1];
        for (int i=num-2; i>=0; i--)
        {
            x0 += con[i]*y0;
            c = x0;
            x0 = y0;
            y0 = c;
        }
        c = x0;
        x0 = y0;
        y0 = c;
        if (x0 * x0 - n * y0 * y0  == 1)
        return ;
    }
}
```

求离散对数 A^x = B mod C 模板 c 不一定要素数

```
#include<iostream>
#include<map>
#include<cmath>
using namespace std;
typedef long long LL;
const int maxn = 65535;
struct hash{
    int a,b,next;
}Hash[maxn << 1];
int flg[maxn];
int top,idx;
void ins(int a,int b){
    int k = b & maxn;
    if(flg[k] != idx){
        flg[k] = idx;
        Hash[k].next = -1;
        Hash[k].a = a;
        Hash[k].b = b;
        return ;
```

```
    }
    while(Hash[k].next != -1){
        if(Hash[k].b == b) return ;
        k = Hash[k].next;
    }
    Hash[k].next = ++ top;
    Hash[top].next = -1;
    Hash[top].a = a;
    Hash[top].b = b;
}
int find(int b){
    int k = b & maxn;
    if(flg[k] != idx) return -1;
    while(k != -1){
        if(Hash[k].b == b) return Hash[k].a;
        k = Hash[k].next;
    }
    return -1;
}
int gcd(int a,int b){return b?gcd(b,a%b):a;}
int ext_gcd(int a,int b,int& x,int& y){
    int t,ret;
    if (!b){x=1,y=0;return a;}
    ret=ext_gcd(b,a%b,x,y);
    t=x,x=y,y=t-a/b*y;
    return ret;
}
int Inval(int a,int b,int n){
    int x,y,e;
    ext_gcd(a,n,x,y);
    e=(LL)x*b%n;
    return e<0?e+n:e;
}
int pow_mod(LL a,int b,int c)
{
LL ret=1%c;a%=c;
while(b)
{
if(b&1)
ret=ret*a%c;
a=a*a%c;
b>>=1;
}return ret;
}
int BabyStep(int A,int B,int C){
    top = maxn; ++ idx;
    LL buf=1%C,D=buf,K;
    int i,d=0,tmp;
    for(i=0;i<=100;buf=buf*A%C,++i)if(buf==B)return i;
    while((tmp=gcd(A,C))!=1){
        if(B%tmp)return -1;
        ++d;
        C/=tmp;
        B/=tmp;
        D=D*A/tmp%C;
    }
    int M=(int)ceil(sqrt((double)C));
    for(buf=1%C,i=0;i<=M;buf=buf*A%C,++i)ins(i,buf);
```

```
    for(i=0,K=pow_mod((LL)A,M,C);i<=M;D=D*K%C,++i){
        tmp=Inval((int)D,B,C);int w ;
        if(tmp>0&&(w = find(tmp)) != -1)return i*M+w+d;
    }
    return -1;
}



int main(){
    int A,B,C;
    while(scanf("%d%d%d",&A,&C,&B)!=EOF,A || B || C){
        B %= C;
        int tmp=BabyStep(A,B,C);
        if(tmp<0)puts("No Solution");else printf("%d\n",tmp);
    }
    return 0;
}
```

求最小 x 使得 a^x = 1 mod n
其中 gcd(a,n)=1

```
u64 getRemainOne(u64 a, u64 n)
{
    u64 phi,min;
    phi = euler(n);
    min = phi;
    for (int i=1; i*i <= phi; i++)
        if (phi % i== 0)
        {
            if (power_mod(2,i,n) == 1 && i < min)
                min = i;
            if (power_mod(2,phi/i,n) == 1 && phi/i < min)
                min = phi/i;
        }
        return min;
}
```

保证 pn<=L,pd<=L 并且 pn/pd 最接近 A.

```
void AppNum(double A, long L, long &pn, long &pd)
{
    double min;
    long i;
    long j;
    long N;
    long D;

        pn = -999999;
        pd = 1;
```

```
        min = 9999999;
        for (D=1; D<=L; ++D)
        {
            N = (long)(D * A);

            if (N > L)
            {
                break;
            }

            for (i=0; i<=1; ++i)
            {
                if (fabs(min - A) > fabs((double)(N+i)/(double)D - A))
                {
                    pn = N+i;
                    pd = D;
                    min = (double)(N+i)/(double)D;
                }
            }
        }

        if (pn == -999999)
        {
            for (D=1; D<=L; ++D)
            {
                for (N=1; N<=L; ++N)
                {
                    if (fabs(min - A) > fabs((double)N/(double)D - A))
                    {
                        pn = N;
                        pd = D;
                        min = (double)N/(double)D;
                    }
                }
            }
        }

}
```

m<10，一个这样的数:M = mmm...mm,要求 M%k=0.
返回 M 的位数，不存在返回-1

```
int calNumMul(u64 k,int m)
{
        u64 L = k;
        L = 9 * L /gcd(9*L,m);
        u64 phi = euler(L);
        u64 min = phi;
        for (int i=1; i*i <= phi; i++)
        if (phi % i == 0)
        {
            if (power_mod(10,i,L) == 1 && i < min)
            min = i;
            if (power_mod(10,phi/i,L) == 1 && i < min)
            min = phi/i;
        }
        if (min < phi)
        return min;
        return -1;
}
```

求 n 所有的约数和

注意：最快是打表

```
/*
  int val[500001] = {0};
    for(int i = 1; i <= 250000; i++)
        for(int j = i<<1; j <= 500000; j += i)
            val[j] += i;
*/
u64 getdiv(u64 num){
    u64 divs[10000], divn[10000];
    u64 d = 1, top = u64(sqrt(double(num))), divl = 0;
    while(d <= top){
        if(d < 3) d++;
        else d += 2;
        u64 cnt = 0;
        while(num % d == 0){
            cnt++;
            num /= d;
        }
        if(cnt){
            divs[divl] = d;
            divn[divl++] = cnt;
            top = int(sqrt(double(num)));
        }
    }
    if(num > 1){
        divs[divl] = num;
        divn[divl++] = 1;
    }
    u64 ret = 1;
    for(u64 i = 0; i < divl; i++){
        u64 j = divn[i];
        u64 k = 1, l = divs[i];
        while(j--){
            k += l;
            l *= divs[i];
        }
        ret *= k;
    }
    return ret;
}
```

状态背包+梅森素数

给定 k 个数 ,p1,p2...pk,计算 n=p1^e1*p2^e2*...*pi^ei*...*pk^ek (0<=ei<=10, no all ei==0)

m是n的因子和，如果m是2的幂指数次即存在x使得m=2^x，求最大的x，不存在输出NO
把n写成质因子的幂指数成绩，底数均为质数，n=x1^a1*x2^a2....
m=(1+x1+x1^2...x1^a1)*(1+x2+x2^2...x2^a2)....要使得m=2^x,显然a1,a2,a3...都必须为1
那么x1,x2,x3就必须为梅森数了，对于输入的数只能是若干不同的梅森素数的积，如果某个数不是，那么把他踢出掉
标记下它能够组合的状态，找个和最大的即可

```
#include <stdio.h>
#include <string.h>
#include <iostream>
using namespace std;
#define FOR(i,s,e) for (int (i)=(s);(i)<(e);i++)
int MePrm[]={(1<<2)-1,(1<<3)-1,(1<<5)-1,(1<<7)-1,(1<<13)-1,(1<<17)-1,(1<<19)-1,(1<<31)-1};
int num[]={2,3,5,7,13,17,19,31};
int sum[300];
int use[300];
void init()
{
    memset(sum,0,sizeof(sum));
    FOR(i,1,256)
        FOR(j,0,8)
        if ((i & (1<<j)) !=0)
        sum[i]+=num[j];
}
```

```
int main()
{
    init();
    int n;
    int t;
    while (scanf("%d",&t)!=EOF)
    {
        memset(use,0,sizeof(use));
        use[0]=1;
        FOR(i,0,t)
        {
            bool sign = true;
            scanf("%d",&n);
            //初始状态设00000000
            int k = 0;
            FOR(j,0,8)
            if (n%MePrm[j] == 0)
            {
```

```
                int time = 0;
                while (n%MePrm[j] == 0&& n!=1 )
                {n/=MePrm[j];time++;}
                //符合要求，含有第 j 个梅森素数的 1 次方。修改状态
                if (time == 1)
                k|=(1<<j);
                else
                {sign = false;break;}
            }
            if (!sign || n > 1 || k==0 ) continue;
            FOR (j,1,256)
                //假设 j=10100110,k=00000110.必须下列条件，j 状态合法
(1^1=0.1^0=1.)
                if ((j&k) == k && use[(j^k)] == 1) {use[j]=1;}
        }
            int max = 0;
            FOR(j,1,256)
                //根据枚举的所有可能的状态，求该状态的最大值
                if (use[j] && max < sum[j]) max=sum[j];
            if (max == 0)
                printf("NO\n");
            else
                printf("%d\n",max);
    }
    return 0;
}
```

求 b^b^b…^b mod m (n 个)

注意这个函数

```
u64 newMod(u64 n, u64 m)
{
    if (n < m) return n;
     return n%m+m;
}
u64 product_mod(u64 a,u64 b,u64 c)
{
    u64 ret=0,tmp=newMod(a,c);
    while(b)
    {
        if(b&0x1)
            if((ret+=tmp)>=c)
                ret-=c;
            if((tmp<<=1)>=c)
                tmp-=c;
            b>>=1;
    }
    return ret;
}
u64  power_mod(u64 A, u64  B, u64 C)
{
     if (B == 0) return 1%C;
    u64  R = 1, D = A;
    while (B )
     {
    //注意这行，也可以调用 product_mod
        if (B&1) R =newMod(R*D,C);
        D = newMod(D*D,C);
        B >>=1;
    }
```

```
    return R;
}
u64 f(u64 b_v,u64 n, u64 m_v)
{
           if (m_v == 1) return 0;
           if (n == 1) return newMod(b_v,m_v);
           u64 ph=euler(m_v);
           u64 k = f(b_v,n-1,ph);
           return power_mod(b_v,k,m_v);
}
```

求最大的并且不大于 n 的最大反素数

反素数：对于 i<x ，必有 g(i)<g(x).g(x)表示 x 的约数个数

性质：为 p0^a0*p1^a1…pi^ai.并且 a0<=a1<=…<=ai

```
int prm[]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
u64  sum;
u64 max_un_prm;
void bfs(u64 n, int k,u64 value, u64 max_value,int max_time)
{
    if (n < value) return ;
    if (max_value > sum || max_value == sum && value < max_un_prm)
    {sum = max_value;max_un_prm = value;}
    u64 n_help = prm[k];
    int time = 1;
    while (value <= n && time <= max_time)
    {
         value*=prm[k];
         bfs(n,k+1,value,max_value*(time+1),time);
         time++;
    }
}
int main()
{
    u64 n;
    while (cin >> n)
    {
        sum = 0;
        bfs(n,0,1,1,100);
       cout << max_un_prm<<endl;;
    }
    return 0;
}
```

整数 HASH ，用以统一每个数字出现的次数

```
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <string.h>
using namespace std;
```

```
inline int IABS(int t) {return t>0?t:-1*t;}
int a, b, c, d, e;
int ans;
int hash[20000];
int key[20000];
int NUM = 16383, CH = 1111;
inline void MakeHash(int value)
{
int t =IABS(value);
t &= NUM;

while (true)
{

if (hash[t] == 0)
{
hash[t] = 1;
key[t] = value;
break;
}
else
{
if (key[t] == value)
{
hash[t]++;
break;
}
else
{
t = (t + CH) & NUM;
}
}
}
}


//求 value 出现次数
inline void GetHash(int value)
{
int t = IABS(value);
t &= NUM;
while (true)
{
if (hash[t] == 0)
{
break;
}
else
{
if (key[t] == -value)
{
ans += hash[t];
break;
}
else
{
t = (t + CH) & NUM;
}
```

```
}
}
}

int main()
{
int i, j;
while (scanf("%d %d %d %d", &a, &b, &c, &d ) !=EOF)
{
memset (hash, 0, sizeof(hash));
memset (key, 0, sizeof(key));
if (a > 0 && b > 0 && c > 0 && d > 0)
    {printf("0\n");continue;}
for (i = -100; i <= 100; i++)
{
for (j = -100; j <= 100; j++)
{
if (i && j)
{
MakeHash(a * i * i + b * j  * j);
}
}
}
ans = 0;
for (i = -100; i <= 100; i++)
{
for (j = -100; j <= 100; j++)
{
{
if (i && j  )
{
GetHash(c * i * i  + d * j  * j );
}
}
}
}
cout << ans << endl;
}
return 0;
}
```

同上，速度较快的一种

```
#define tmax 3000001
struct NODE
{
    int val;
    int tot;
    bool used;
}hash[tmax];
 //定位函数
int locate(int s)
{
    int temp;
    temp =s;
    while (temp < 0) temp+=tmax;
    while (temp >= tmax)temp%=tmax;
    while (hash[temp].used && hash[temp].val != s)
    {
        temp++;
        if (temp >= tmax ) temp-=tmax;
    }
    return temp;
}
 //此为 hash 函数
void insert_in_hash(int value)
{
    int s = locate(value);
    hash[s].used = true;
    hash[s].val = value;
    hash[s].tot++;
}
void GetHash(int s)
{
    s=-1*s;
    int t =locate(s);
    ans+=hash[t].tot;
    return ;
}
```

Sum(gcd(x,y)) 1<=x,y<=n

需要[1…n]的前 n 项和欧拉函数表,

```
u64 gcdExtreme(u64 n)
{
     u64 sum = sum_phi[n];
          for (u64 i=2; i< (n)/2+1; )
          {
```

```
            u64 k = n / i;
            u64 j = n / k;
            j++;
            u64 num = (i+j-1)*(j-i)/2;
            sum += num * sum_phi[n/i];
            i = j ;
         }
    return sum;
}
```

大整数因数分解

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cstdio>
using namespace std;
typedef     __int64 u64;
const int MAX = 100;
const int MAXN = 30;
#define INF 1000000000

 u64 gcd(u64 a, u64 b) {
    if (a < b) return gcd(b,a);
    u64 c;
    while (b) {c=a%b;a=b;b=c;}
    return a;
}

 u64 product_mod(u64 a,u64 b,u64 c)
{
    u64 ret=0,tmp=a%c;
    while(b)
    {
        if(b&0x1)
            if((ret+=tmp)>=c)
                ret-=c;
            if((tmp<<=1)>=c)
                tmp-=c;
```

```
            b>>=1;
    }
    return ret;
}

u64 random() {
      u64 a;
      a = rand();
      a *= rand();
      a *= rand();
      a *= rand();
      return a;
}

u64 f(u64 x, u64 n) {
      return (product_mod(x, x, n) + 1) % n;
}
u64 Pollard_Rho(u64 n) {
      if(n <= 2) return 0;
      if(! (n & 1)) return 2;
      u64 i, p, x, xx;
      for(i = 1; i < MAX; i ++) {
           x = random() % n;
           xx = f(x, n);
           p = gcd((xx + n - x) % n, n);
           while(p == 1) {
                x = f(x, n);
                xx = f(f(xx, n), n);
                p = gcd((xx + n - x) % n, n) % n;
           }
           if(p) return p;
      }
      return 0;
}

u64 power_mod(u64 x, u64 c, u64 n) {
      u64 z = 1;
      while(c) {
           if(c & 1) z =product_mod(z, x, n);
           x = product_mod(x, x, n);
           c >>= 1;
      }
      return z;
}

bool Miller_Rabin(u64 n) {
      if(n < 2) return false;
      if(n == 2) return true;
      if(! (n & 1)) return false;
      u64 i, j, k, m, a;
      m = n - 1;
      k = 0;
      while(m % 2 == 0) {
           m >>= 1; k ++;
      }
      for(i = 0; i < MAX; i ++) {
           a = power_mod(random() % (n - 1) + 1, m, n);
           if(a == 1) continue;
```

```
        for(j = 0; j < k; j ++) {
            if(a == n - 1) break;
            a = product_mod(a, a, n);
        }
        if(j == k) return false;
    }
    return true;
}


u64 Prime(u64 a) {
    if(Miller_Rabin(a)) return 0;
    u64 t = Pollard_Rho(a);
    u64 p = Prime(t);
    if(p) return p;
    return t;
}


u64 factor[MAXN];
u64 facNum[MAX];
```
直接调用即可。因子存在 factor 中，相应系数存在 facNum 中，返回素因子个数.
```
int BigNumRes(u64 a)
{
    int m = 0;
    int time=0;
    u64 t;
    while(a > 1)
    {
        if(Miller_Rabin(a)) break;
        t = Prime(a);
        factor[m ++] = t;
        time=0;
        while (a%t==0 && a!=1) {time++;a/=t;}
        facNum[m-1]=time;
    }
    if(a != 1) {factor[m ++] = a;facNum[m-1]=1;}
    return m;
}
```

三分模板

对于满足凸性的函数
```
double Calc(Type a)
{
    /* 根据题目的意思计算 */
}
void Solve(void)
{
    double Left, Right;
    double mid, midmid;
    double mid_value, midmid_value;
    Left = MIN; Right = MAX;
    while (Left + EPS < Right)
    {
```

```
        mid = (Left + Right) / 2;
        midmid = (mid + Right) / 2;
        mid_area = Calc(mid);
        midmid_area = Calc(midmid);
        // 假设求解最大极值.
        if (mid_area >= midmid_area) Right = midmid;
        else Left = mid;
    }
}
```

哥德巴赫猜想

.每个不小于 6 的偶数都可以表示为两个奇素数之和；2.每个不小于 9 的奇数都可以表示为三个奇素数之和。

梅森合数的因数分解

将一个梅森合数因素分解。

这里用到一个性质，2^p-1 的梅森合数，约数形式必须时候 2*h*p +1 .

枚举 p ，h 即可。

//分解梅森合数 2^k-1

 int prm[]={11,23,29,37,41,43,47,53,59};

k 为 prm[]中一个

```
int MesanCom(int k,u64 fac[])
{
    int num =0;
    u64 n = ((u64)(1)<<k)-1;
    for (int j=1; (u64)(2*k*j+1)*(u64)(2*j*k+1) <= n; j++ )
        if (n % (2*k*j+1) == 0)
        {
            fac[num++] = (u64)(2*k*j+1);
            while (n % (2*k*j+1) == 0)
                n/=(2*k*j+1);
        }
    if (n!=1) fac[num++] = n;
    return num;
}
```

计算前 n 个 Catalen 数和 mod m

```
.C[n]=(4n-2)/(n+1*)C[n]。需要素数表
const int maxN=100010;
int totle_prm_m[maxN];
int prm_m[100];//存储 m 的素因子
u64 euler_m;//存 m 欧拉函数值
int num_prm_m=0;
void init(int m)
{
    num_prm_m = 0;
    euler_m=m;
    memset(totle_prm_m,0,sizeof(totle_prm_m));
    for (int i=0; i<totleprm && prm[i] * prm[i] <= m; i++)
        if (m % prm[i] == 0)
        {
            euler_m = euler_m * (prm[i]-1)/prm[i];
            if (prm[i] <= maxN)
            prm_m[num_prm_m++]=prm[i];
            while (m!=1 && m % prm[i] == 0) m/=prm[i];
        }
        if (m!=1)
        {
            euler_m = euler_m * (m-1) / m;
            if (m <= maxN)
            prm_m[num_prm_m++]=m;
        }

 }
u64 SumCatalanModM(u64 n,u64 m)
{

    init(m);
    u64 a = 1;
    u64 b = 1;
    u64 ans = 0;
    u64 a_help;
    int time;
    int t;
    FOR(i,1,n+1)
    {
        //分子
        t = 4*i-2;
        FOR(j,0,num_prm_m)
        if (t % prm_m[j] == 0)
        {
              time = 0;
            while (t!=1 && t % prm_m[j] == 0)
            {time++;t/=prm_m[j];}
            totle_prm_m[prm_m[j]]+=time;
        }
        a = (a*t)%m;
        t = i+1;

        FOR(j,0,num_prm_m)
        if (t % prm_m[j] == 0)
        {
              time = 0;
            while (t != 1  && t % prm_m[j] == 0)
```

```
            {time++;t/=prm_m[j];}
            totle_prm_m[prm_m[j]]-=time;
        }
        //分母
        b=(b*t)%m;
        a_help=a;
        FOR(j,0,num_prm_m)
        if (totle_prm_m[prm_m[j]])
        a_help = (a_help*power_mod(prm_m[j],totle_prm_m[prm_m[j]],m) ) % m;

        ans=(ans+(a_help*power_mod(b,euler_m-1,m))%m)%m;
    }
    return ans;
}
```

矩阵相关

```
#define N 20
typedef long long u64;
struct Mat
{
    u64 matrix[N][N];
    int r,l;
};
//单位矩阵
Mat ONE;
int p;
//矩阵乘法
Mat mul(Mat a, Mat b)
{
    int i, j, k;
    Mat c;
    for (i = 0; i < a.r; i++)
        for (j = 0; j < b.l; j++)
```

```
        {
            c.matrix[i][j] = 0;
            for (k = 0; k < a.l; k++)
            {
                c.matrix[i][j] +=  (a.matrix[i][k]*b.matrix[k][j])%p;
                c.matrix[i][j]%=p;
            }
        }
    c.r = a.r ;
    c.l = b.l;
    return c;
}
//高次矩阵幂
Mat mat_Power (Mat a,u64 n)
{
    if (n == 0) return ONE;
    if (n == 1)
    return a;
    Mat ans;
    Mat k = mat_Power(a, n/2);
    ans.l=ans.r=a.l;
    if (n % 2 == 0)
    ans= mul(k,k);
    else
    ans= mul(mul(k,k),a);
    return ans;}
//矩阵加法
Mat add_mat(Mat a, Mat b)
{
    Mat c;
    FOR(i,0,a.r)
        FOR(j,0,a.l)
            c.matrix[i][j] = (a.matrix[i][j]+b.matrix[i][j])%p;
    c.l = a.l;
    c.r = a.r;
    return c;
}

//计算 a + a^2 +a^3...a^n
Mat Sum_mat(Mat a, u64 n)
{
    if (n == 1) return a;
    //ONE 是单位矩阵
    if (n == 0) return ONE;
    Mat k = Sum_mat(a,n/2);
    Mat sum_help;
    Mat ans;
    ans.r = ans.l = a.l;
    if (n % 2 == 0)
    {
        sum_help =  mat_Power(a,n/2);
        ans = add_mat(k, mul(sum_help, k) );
    }
    else
    {
        sum_help = mat_Power(a,n/2+1);
        ans = add_mat(k , add_mat(sum_help, mul(sum_help, k)) );
    }
```

```
    return ans;
}
```

外挂读入

```
double get(){
    char c;
    double ret;
    while (c=getchar(),c<'0'||c>'9');
    ret=c-'0';
    while (c=getchar(),c>='0'&&c<='9')
        ret=ret*10+c-'0';
    return ret;
}
```