**1.逆元求解**

//a/b%c ==> a\*b'%c(b'为b模c的逆元)

//逆元求解: a模b的逆元

1.a,b互质，可以用拓展的欧几里得定理

ll e\_gcd(ll a,ll b,ll&x,ll&y) (x为所求逆元)

{

if(b == 0){

x = 1; y = 0;

return a;

}

ll ans = e\_gcd(b,a%b,x,y);

ll temp = x;

x = y;

y = temp-a/b\*y;

return ans;

}

2.b为质数，由费马小定理a^(b-1) == 1(mod b),a^(b-2)为逆元

用快速幂求

ll fast\_mi(ll x,ll n)

{

ll ans = 1;

while(n){

if(n&1) ans = ans\*x%mod;

x = x\*x%mod;

n >>= 1;

}

return ans;

}

3.通用情况 a/b%c ==> a%(b\*c)/b;

4.逆元打表:1-p模p的所有逆元,p为奇质数 inv[i] = (M-M/i)\*inv[M/i]%M;

inv[1] = 1;

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

{

if(i >= mod) break;

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

}

**例1.uva11174(递推+逆元)**

村民排队，村子里现有n个人，有多少种方法可以把他们排成一列，使得没有人能排在他们的父亲前。

#include <iostream>

#include <cstring>

#include <string>

#include <algorithm>

#include <cmath>

using namespace std;

typedef long long ll;

const int mod = 1e9+7;

const int N = 40100;

struct Edge{

int node;

Edge\*next;

}m\_edge[N\*2];

int Ecnt,son[N],parent[N];

Edge\*head[N];

void init()

{

Ecnt = 0;

fill(head,head+N,(Edge\*)0);

}

void mkEdge(int a,int b)

{

m\_edge[Ecnt].node = b;

m\_edge[Ecnt].next = head[a];

head[a] = m\_edge+Ecnt++;

}

void dfs( int u )

{

son[u] = 1;

for( Edge\*p = head[u] ; p ; p = p->next ){

int v = p->node;

dfs(v);

son[u] += son[v];

}

}

ll inv[N],fac[N]; //逆元,阶乘打表

//费马小定理求逆元

//ll fast\_mi(ll x,ll n)

//{

// ll res = 1;

// while(n){

// if(n&1) res = (res\*x)%mod;

// x = x\*x%mod;

// n >>= 1;

// }

// return res;

//}

//拓展的欧基里得求逆元

ll e\_gcd(ll a,ll b,ll&x,ll&y)

{

if(b == 0){

x = 1; y = 0;

return a;

}

ll ans = e\_gcd(b,a%b,x,y);

ll temp = x;

x = y;

y = temp-a/b\*y;

return ans;

}

//预处理

void pretreat()

{

fac[0] = 1LL;

for( int i = 1 ; i <= 40000 ; ++i ){

fac[i] = (fac[i-1]\*i)%mod;

}

// for( int i = 1 ; i <= 40000 ; ++i ){

// inv[i] = fast\_mi(i,(ll)(mod-2));

// }

ll x,y;

for( int i = 1 ; i <= 40000 ; ++i ){

e\_gcd((ll)i,(ll)mod,x,y);

x %= mod;

if(x < 0) x += mod;

inv[i] = x;

}

}

int main()

{

pretreat();

int T;

scanf("%d",&T);

while(T--){

init();

int n,m;

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

init();

int a,b;

memset(parent,0,sizeof(parent));

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

scanf("%d%d",&a,&b);

mkEdge(b,a);

parent[a] = b;

}

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

if(parent[i] == 0) mkEdge(0,i);

}

dfs(0);

ll ans = fac[n];

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

ans = (ans\*inv[son[i]])%mod;

}

printf("%lld\n",ans);

}

return 0;

}

**例2.poj1845(二分等比数列求和)**

给定两个正整数http://img.blog.csdn.net/20140613112543656和http://img.blog.csdn.net/20140613112553093，求http://img.blog.csdn.net/20140613112622265的所有因子和对**9901**取余后的值。

#include <iostream>

#include <cstring>

#include <string>

#include <algorithm>

#include <cstdio>

using namespace std;

typedef long long ll;

const int N = 10000;

const int mod = 9901;

int isNotPrime[N],isPrime[N],num\_prime;

void getPrime()

{

memset(isNotPrime,0,sizeof(isNotPrime));

num\_prime = 0;

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

if(!isNotPrime[i]) isPrime[num\_prime++] = i;

for( int j = 0 ; j < num\_prime && i\*isPrime[j] < N ; ++j ){

isNotPrime[i\*isPrime[j]] = 1;

if(i%isPrime[j] == 0) break;

}

}

}

ll fast\_mi(ll x,ll n)

{

ll ans = 1;

while(n){

if(n&1) ans = ans\*x%mod;

x = x\*x%mod;

n >>= 1;

}

return ans;

}

//using dichotomy to calculate the sum of geometric sequence

ll sum(ll a,ll n)

{

if(n == 1) return a%mod;

ll s = sum(a,n/2);

if(n&1){

s = (((1+fast\_mi(a,(n-1)/2))\*s)%mod+fast\_mi(a,n))%mod;

}else{

s = ((1+fast\_mi(a,n/2))\*s)%mod;

}

return s;

}

ll solve(ll a,ll b)

{

ll cnt,ans = 1;

for( int i = 0 ; isPrime[i]\*isPrime[i] <= a && i < num\_prime ; ++i ){

if(a%isPrime[i] == 0){

cnt = 0;

while(a%isPrime[i] == 0){

cnt++;

a /= isPrime[i];

}

ans = ans\*(sum(isPrime[i],cnt\*b)+1)%mod;

}

}

if(a > 1)

ans = ans\*(sum(a,b)+1)%mod;

return ans;

}

int main()

{

getPrime();

ll A,B;

while(~scanf("%lld%lld",&A,&B)){

if(B == 0) {printf("%d\n",1); continue;}

printf("%lld\n",solve(A,B));

}

return 0;

}

**通用方法求逆元**

#include <iostream>

#include <algorithm>

#include <cstdio>

using namespace std;

typedef long long ll;

const int N = 10000;

const int mod = 9901;

int isNotPrime[N],isPrime[N],num\_prime;

void getPrime()

{

memset(isNotPrime,0,sizeof(isNotPrime));

num\_prime = 0;

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

if(!isNotPrime[i]) isPrime[num\_prime++] = i;

for( int j = 0 ; j < num\_prime && i\*isPrime[j] < N ; ++j ){

isNotPrime[i\*isPrime[j]] = 1;

if(i%isPrime[j] == 0) break;

}

}

}

ll fast\_mult(ll x,ll n,ll mod)

{

ll ans = 0;

while(n){

if(n&1) ans = (ans+x)%mod;

x = (x+x)%mod;

n >>= 1;

}

return ans;

}

ll fast\_mi(ll x,ll n,ll mod)

{

ll ans = 1;

while(n){

if(n&1) ans = fast\_mult(ans,x,mod); //Multiplication may overflow int64

x = fast\_mult(x,x,mod);

n >>= 1;

}

return ans;

}

ll solve(ll a,ll b)

{

ll cnt,ans = 1;

for( int i = 0 ; isPrime[i]\*isPrime[i] <= a && i < num\_prime ; ++i ){

if(a%isPrime[i] == 0){

cnt = 0;

while(a%isPrime[i] == 0){

cnt++;

a /= isPrime[i];

}

ll M = (ll)(isPrime[i]-1)\*mod;

ans \*= (fast\_mi(isPrime[i],cnt\*b+1,M)-1+M)/(isPrime[i]-1);

ans %= mod;

}

}

if(a > 1){

ll M = (a-1)\*mod;

ans \*= (fast\_mi(a,b+1,M)-1+M)/(a-1);

ans %= mod;

}

return ans;

}

int main()

{

getPrime();

ll A,B;

while(~scanf("%lld%lld",&A,&B)){

if(B == 0) {printf("%d\n",1); continue;}

printf("%lld\n",solve(A,B));

}

return 0;

}

**2.质数，欧拉函数，因子个数，因子和打表**

**质数的快速线性筛法，不会重复筛选**

1.对于(质数\*质数)的情况：不会出现重复筛的情况

2.对于(质数\*合数)的情况：就会出现重复筛，例如4\*3,6\*2

由于任意个合数可以分解为质数的乘积，p=p1\*p2...\*pn(p1为最小的质数)

当一个合数乘上一个比他最小的质数还要小或等于的质数时不会重复筛，所以当p%p1=0时，停止筛选

int Prime[N],isNotPrime[N],num\_prime = 0;

void getPrime( int n )

{

fill( isNotPrime , isNotPrime+N , 0 );

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

if( !isNotPrime[i] ) Prime[num\_prime++] = i;

for( int j = 0 ; j < num\_prime && i\*Prime[j] < n ; ++j ){

isNotPrime[i\*Prime[j]] = 1;

if( i%Prime[j] == 0 ) break;

}

}

}

**欧拉函数线性打表**

欧拉函数是小于n且与n互质的数的个数，f(1)=1，例如f(8)=4，有1,3,5,7

设n=p1\*p2...\*pn(pi是n的质数)，则f(n)=n\*(1-1/p1)\*(1-1/p2).....(1-1/pn)

int euler[N];

void getEuler( int n )

{

for( int i = 0 ; i < n ; ++i ) euler[i] = i;

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

//i为质数

if( euler[i] == i ){

for( int j = 1 ; j\*i < n ; ++j ){

euler[j\*i] -= euler[j\*i]/i;

}

}

}

}

**求某数正因子个数打表**

算数基本定理求正因子个数

设n=p1^a1\*p2^a2...\*pn^an(pi是n的质数,ai是质数的次幂)

f(n)=(a1+1)\*(a2+1)...\*(a3+1)

int num[N];

void getNumDivisor( int n )

{

fill( num , num+N , 0 );

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

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

num[i\*j] += 2;

if( i == j ) num[i\*j]--;

}

}

}

**求某数所有正因子和打表**

算数基本定理求所有正因子个数和

设n=p1^a1\*p2^a2...\*pn^an(pi是n的质数,ai是质数的次幂)

f(n)=(1+p1...p1^a1)\*(1+p2...+p2^a2)...\*(1+pn...+pn^an)

等比数列乘积

int sum[N];

void getSumdivisor( int n )

{

fill( sum , sum+N , 0 );

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

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

sum[i\*j] += i+j;

if( i == j ) sum[i\*j] -= i;

}

}

}

**3．求菲波拉契数的前4位和后4位**

#include <iostream>

#include <algorithm>

#include <cstring>

#include <string>

#include <cmath>

using namespace std;

const int mod = 10000;

//求后面4位，对10000取模

//求前面4位，使用菲波拉契数的公式，取对数

struct Matrax{

int a[2][2];

Matrax(){

memset(a,0,sizeof(a));

}

};

Matrax Mult\_Matrax(Matrax x,Matrax s)

{

Matrax ans;

for( int i = 0 ; i < 2 ; ++i ){

for( int j = 0 ; j < 2 ; ++j ){

for( int k = 0 ; k < 2 ; ++k ){

ans.a[i][j] = (ans.a[i][j]+x.a[i][k]\*s.a[k][j]%mod)%mod;

}

}

}

return ans;

}

Matrax Fast\_Matrax(Matrax x,int n)

{

Matrax ans;

for( int i = 0 ; i < 2 ; ++i ) ans.a[i][i] = 1;

while(n){

if(n&1) ans = Mult\_Matrax(ans,x);

x = Mult\_Matrax(x,x);

n >>= 1;

}

return ans;

}

int main()

{

int fabo[50];

fabo[1] = fabo[2] = 1;

fabo[0] = 0;

for( int i = 3 ; i < 40 ; ++i ){

fabo[i] = fabo[i-1]+fabo[i-2];

}

int n;

while(~scanf("%d",&n)){

if(n <= 39){printf("%d\n",fabo[n]);continue;}

int L4,F4;

//后4位取模

Matrax s;

s.a[0][0] = s.a[0][1] = s.a[1][0] = 1;

s = Fast\_Matrax(s,n-1);

L4 = s.a[0][0];

//前4位取对数

double s1 = 1.0/sqrt(5.0);

double s2 = (1.0+sqrt(5.0))/2.0;

double as = log10(s1)+n\*log10(s2);

as = as-floor(as);

as = pow(10,as);

F4 = floor(as\*10000)/10;

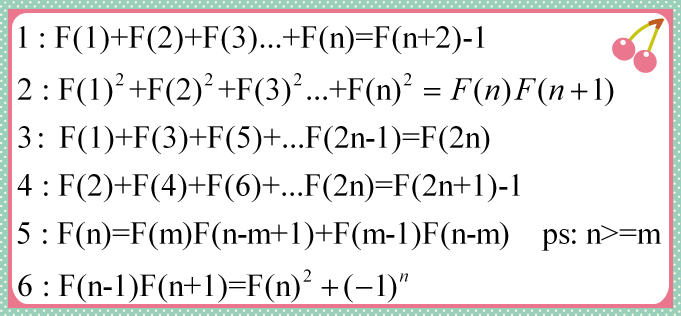
printf("%d...%04d\n",F4,L4); //后面4位不足要补0

}

return 0;

}

菲波拉契数的一些性质



**4.拓展的大步小步算法**

#include <cstdio>

#include <cmath>

#include <map>

using namespace std;

typedef long long ll;

int fast\_mi(ll x, ll n, ll mod\_v)

{

ll ans = 1;

while(n)

{

if(n&1) ans = ans\*x% mod\_v;

x = x\*x%mod\_v;

n >>= 1;

}

return ans;

}

int gcd(int a, int b)

{

return b ? gcd(b, a % b) : a;

}

int baby\_step\_giant\_step(int a, int b, int p)

{

a %= p, b %= p;

if(b == 1) return 0;

int cnt = 0;

ll t = 1;

for(int g = gcd(a, p); g != 1; g = gcd(a, p))

{

if(b % g) return -1;

p /= g, b /= g, t = t \* a / g % p;

++cnt;

if(b == t) return cnt;

}

map<int, int> hash;

int m = int(sqrt(1.0 \* p) + 1);

ll base = b;

for(int i = 0; i != m; ++i)

{

hash[base] = i;

base = base \* a % p;

}

base = fast\_mi(a, m, p);

ll now = t;

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

{

now = now \* base % p;

if(hash.count(now))

return i \* m - hash[now] + cnt;

}

return -1;

}

int main()

{

int a, b, p;

while(scanf("%d %d %d", &a, &p, &b), p)

{

int ans = baby\_step\_giant\_step(a, b, p);

if(ans == -1) puts("No Solution");

else printf("%d\n", ans);

}

return 0;

}

**5.中国剩余定理模板**

#include <cstdio>

#include <cmath>

#include <map>

using namespace std;

typedef long long ll;

//中国剩余定理，求解x==ai(mod mi),所有mi互素

ll e\_gcd(ll a,ll b,ll&x,ll&y)

{

if(b == 0){

x = 1; y = 0;

return a;

}

ll ans = e\_gcd(b,a%b,x,y);

ll temp = x;

x = y;

y = temp-a/b\*y;

return ans;

}

ll china(int n,int a[],int m[])

{

ll M = 1,x,y,ans = 0;

for( int i = 0 ; i < n ; ++i ) M \*= m[i];

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

ll w = M/m[i];

e\_gcd(w,m[i],x,y);

ans = (ans+x\*w\*a[i])%M;

}

return (x+M)%M;

}

int main()

{

return 0;

}

#### 6.大质数检测，Miller-Rabin质数测试

#include<cstdio>

#include<iostream>

#include<cmath>

#include<algorithm>

using namespace std;

typedef long long ll;

ll myrand()

{

ll a = rand();

a=a\*rand();

return a;

}

ll fast\_mult(ll x,ll n,ll mod) //快速乘

{

ll ans = 0;

while(n){

if(n&1) ans = (ans+x)%mod;

x = (x+x)%mod;

n >>= 1;

}

return ans;

}

ll fast\_mi(ll x,ll n,ll mod) //快速幂

{

ll ans = 1;

while(n){

if(n&1) ans = fast\_mult(ans,x,mod);

x = fast\_mult(x,x,mod);

n >>= 1;

}

return ans;

}

bool Miller\_Rabin(ll n,int times)

{

ll u,x,y,a;

if(n<2)

return false;

if(n==2)

return true;

if(!(n&1))

return false;

//我们先找到的最小的a^u，再逐步扩大到a^(n-1)

for( u=n-1 ; !(u&1) ; u>>=1 );

//Miller-Rabin测试

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

a = myrand()%(n-2)+2;

x = fast\_mi(a,u,n);

while(u < n){

y=fast\_mi(x,2,n);

if(y == 1 && x != 1 && x != n-1){

return false;

}

else{

x=y;

u=u\*2;

}

}

if(x!=1)

return false;

}

return true;

}

int main()

{

int T;

scanf("%d",&T);

while(T--){

ll n;

scanf("%lld",&n);

if(Miller\_Rabin(n,20)){

cout<<"Yes"<<endl;

}else{

cout<<"No"<<endl;

}

}

return 0;

}

**7.求大数因子，Pollard\_rho[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)，hdu3864**

#include<cstdio>

#include<cstdlib>

#include<iostream>

#include<cmath>

#include<algorithm>

#include <ctime>

using namespace std;

typedef long long ll;

const int times = 20;

ll myrand()

{

ll a = rand();

a=a\*rand();

return a;

}

ll fast\_mult(ll x,ll n,ll mod) //快速乘

{

ll ans = 0;

while(n){

if(n&1) ans = (ans+x)%mod;

x = (x+x)%mod;

n >>= 1;

}

return ans;

}

ll fast\_mi(ll x,ll n,ll mod) //快速幂

{

ll ans = 1;

while(n){

if(n&1) ans = fast\_mult(ans,x,mod);

x = fast\_mult(x,x,mod);

n >>= 1;

}

return ans;

}

bool Miller\_Rabin(ll n,int times)

{

ll u,x,y,a;

if(n<2)

return false;

if(n==2)

return true;

if(!(n&1))

return false;

//我们先找到的最小的a^u，再逐步扩大到a^(n-1)

for( u=n-1 ; !(u&1) ; u>>=1 );

//Miller-Rabin测试

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

a = myrand()%(n-2)+2;

x = fast\_mi(a,u,n);

while(u < n){

y=fast\_mi(x,2,n);

if(y == 1 && x != 1 && x != n-1){

return false;

}

else{

x=y;

u=u\*2;

}

}

if(x!=1)

return false;

}

return true;

}

ll f[100],cnt;

ll gcd(ll a,ll b)

{

return b==0?a:gcd(b,a%b);

}

ll Pollard\_rho(ll n,ll c)//Pollard\_rho算法，找出n的因子

{

ll i=1,j,k=2,x,y,d,p;

x=rand()%n;

y=x;

while(1){

i++;

x=(fast\_mult(x,x,n)+c)%n;

if(y==x)return n;

if(y>x)p=y-x;

else p=x-y;

d=gcd(p,n);

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

if(i==k){

y=x;

k+=k;

}

}

}

void find(ll n)//找出n的所有因子

{

if(Miller\_Rabin(n,times)){

f[cnt++]=n;//保存所有质因子

return;

}

ll p=n;

while(p>=n)p=Pollard\_rho(p,rand()%(n-1)+1);//由于p必定为合数，所以通过多次求解必定能求得答案

find(p);

find(n/p);

}

int main()

{

srand(time(NULL));//随机数设定种子

ll n;

while(cin>>n)

{

if(n==1){cout<<"is not a D\_num"<<endl;continue;}//特判

cnt=0;

find(n);

if(cnt!=2&&cnt!=3){cout<<"is not a D\_num"<<endl;continue;}

sort(f,f+cnt);

if(cnt==2){

if(f[0]!=f[1])cout<<f[0]<<" "<<f[1]<<" "<<n<<endl;

else cout<<"is not a D\_num"<<endl;

}else{ //n是一个素数的三次方

if(f[0]==f[1]&&f[1]==f[2])cout<<f[0]<<" "<<f[0]\*f[0]<<" "<<n<<endl;

else cout<<"is not a D\_num"<<endl;

}

}

return 0;

}