|  |
| --- |
| ACM-ICPC |
| D-Tesla模板 |
| 2018 |

|  |
| --- |
| dengtesla  2018/11/8 |

目录

[1 数论 1](#_Toc529480637)

[1.1 线性筛打质数表，质因数分解 1](#_Toc529480638)

[1.2 莫比乌斯函数，欧拉函数 2](#_Toc529480639)

[1.3 Miller-Rabin素性测试 4](#_Toc529480640)

[1.4 快速幂，矩阵快速幂 5](#_Toc529480641)

[1.5 卢卡斯、大组合数取模 6](#_Toc529480642)

[1.6 中国剩余定理（不互质情况） 7](#_Toc529480643)

[1.7 对前n个数分解质因数 12](#_Toc529480644)

[1.8 将根式转换为连分数形式 13](#_Toc529480645)

[1.9 Meissel-Lehmer算法（求1e11内质数个数） 14](#_Toc529480646)

[1.10 Min\_25自动机（亚线性处理积性函数前缀和） 16](#_Toc529480647)

[1.11 求解同余方程(x^2=a(mod p)) 21](#_Toc529480648)

[1.12 类欧几里得算法 23](#_Toc529480649)

[1.13 预处理狄利克雷卷积（O(nlnn)） 25](#_Toc529480650)

[2 字符串 26](#_Toc529480651)

[2.1 KMP字符串匹配 26](#_Toc529480652)

[2.2 Manacher(回文串计算) 28](#_Toc529480653)

[3 数据结构 29](#_Toc529480654)

[3.1 并查集 29](#_Toc529480655)

[3.2 链表 30](#_Toc529480656)

[3.3 线段树 31](#_Toc529480657)

[3.4 zkw线段树 32](#_Toc529480658)

[3.5 主席树 32](#_Toc529480659)

[3.6 树状数组 33](#_Toc529480660)

[3.7 树链剖分 34](#_Toc529480661)

[4 计算几何 39](#_Toc529480662)

[4.1 基础模板 39](#_Toc529480663)

[4.2 最大空凸包 39](#_Toc529480664)

[5 杂项 42](#_Toc529480665)

[5.1 罗马-数字转换 42](#_Toc529480666)

[5.2 Bm算法求解线性递推 44](#_Toc529480667)

[5.3 FFT求多项式乘法 47](#_Toc529480668)

[5.4 拉格朗日插值 49](#_Toc529480669)

[5.5 高速大数（longlong）运算 50](#_Toc529480670)

[5.6 自己的FFT 53](#_Toc529480671)

# 数论

## 线性筛打质数表，质因数分解

*#define MAXIMUM 26*

int prime[1000000];

bool isprime[1000000];

void get\_prime(int listsize)

{

int primesize=1;

memset(isprime,1,**sizeof**(isprime));

isprime[1] = false;

**for**(int i=2;i<=listsize;i++)

{

**if**(isprime[i]) prime[primesize++]=i;

**for**(int j=1;i\*prime[j]<=listsize&&j<=primesize;j++)

{

isprime[i\*prime[j]] = false;

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

}

}

}

**struct** p

{

int value;

int time;

}p[MAXIMUM];

void prime\_factorization(long long n)

{

memset(p,0,**sizeof**(p));

long long psize=0;

long long a = n;

**for**(int t = 1;1LL\*prime[t]\*prime[t]<=n;t++)

{

**if**(a%prime[t]==0) p[++psize].value = prime[t];

**while**(a%prime[t]==0)

{

p[psize].time += 1;

a = a / prime[t];

}

**if**(a<=90000)

**if**(isprime[a])

{

p[++psize].value = a;

p[psize].time += 1;

a = 1;

**break**;

}

**if**(a==1) **break**;

}

**if**(a!=1)

{

p[++psize].value = a;

p[psize].time = 1;

}

}

## 莫比乌斯函数，欧拉函数

**const** int MAX = 101000;

int mu[MAX+10];

bool isprime[MAX+10];

int prime[MAX+10];

void get\_mobius(int n)

{

mu[1] = 1;

int tot = 0;

memset(isprime,1,**sizeof**(isprime));

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

{

**if**(isprime[i])

{

prime[++tot] = i;

mu[i] = -1;

}

**for**(int j=1;prime[j]\*i<=n;j++)

{

isprime[prime[j]\*i] = 0;

**if**(i%prime[j]==0)

{

mu[prime[j]\*i] = 0;

**break**;

}

mu[prime[j]\*i]=-mu[i];

}

}

}

int euler[MAX+10];

void get\_euler(int n)

{

euler[1] = 1;

int tot = 0;

memset(isprime,1,**sizeof**(isprime));

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

{

**if**(isprime[i])

{

prime[++tot] = i;

euler[i] = i-1;

}

**for**(int j=1;prime[j]\*i<=n;j++)

{

isprime[prime[j]\*i] = 0;

**if**(i%prime[j]==0)

{

euler[prime[j]\*i] = prime[j]\*euler[i];

**break**;

}

euler[prime[j]\*i] = euler[i]\*(prime[j]-1);

}

}

}

## Miller-Rabin素性测试

**typedef** long long LL;

LL iprime[6] = {2, 3, 5, 233, 331};

LL qmul(LL x, LL y, LL mod) { *// 乘法防止溢出， 如果p \* p不爆LL的话可以直接乘； O(1)乘法或者转化成二进制加法*

**return** (x \* y - (long long)(x / (long double)mod \* y + 1e-3) \*mod + mod) % mod;

}

LL qpow(LL a, LL n, LL mod) {

LL ret = 1;

**while**(n) {

**if**(n & 1) ret = qmul(ret, a, mod);

a = qmul(a, a, mod);

n >>= 1;

}

**return** ret;

}

bool Miller\_Rabin(LL p) {

**if**(p < 2) **return** 0;

**if**(p != 2 && p % 2 == 0) **return** 0;

LL s = p - 1;

**while**(! (s & 1)) s >>= 1;

**for**(int i = 0; i < 5; ++i) {

**if**(p == iprime[i]) **return** 1;

LL t = s, m = qpow(iprime[i], s, p);

**while**(t != p - 1 && m != 1 && m != p - 1) {

m = qmul(m, m, p);

t <<= 1;

}

**if**(m != p - 1 && !(t & 1)) **return** 0;

}

**return** 1;

}

## 快速幂，矩阵快速幂

ll poww(ll a,ll b,ll mod)

{

ll ans = 1,base = a;

**while**(b)

{

**if**(b&1)

{

ans \*= base;

ans %= mod;

}

base \*= base;

base %= base;

b>>=1;

}

**return** ans;

}

**namespace** matrix

{

**const** int N = 100;

**struct** matrix

{

ll v[N][N];

};

matrix multi(matrix a,matrix b)

{

matrix ans;

memset(ans.v,0,**sizeof**(ans.v));

**for**(int i=0;i<N;i++)

**for**(int j=0;j<N;j++)

**for**(int k=0;k<N;k++)

ans.v[i][j] += a.v[i][k]\*b.v[k][j];

**return** ans;

}

matrix mat\_pow(matrix a,ll b)

{

matrix ans;

memset(ans.v,0,**sizeof**(ans.v));

**for**(int i=0;i<N;i++) ans.v[i][i] = 1;

**while**(b)

{

**if**(b&1) ans = multi(ans,a);

a = multi(a,a);

b>>=1;

}

**return** ans;

}

}

## 卢卡斯、大组合数取模

LL PowMod(LL a,LL b,LL MOD){

LL ret=1;

**while**(b){

**if**(b&1) ret=(ret\*a)%MOD;

a=(a\*a)%MOD;

b>>=1;

}

**return** ret;

}

LL fac[100005];

LL Get\_Fact(LL p){

fac[0]=1;

**for**(LL i=1;i<=p;i++)

fac[i]=(fac[i-1]\*i)%p; *//预处理阶乘*

}

LL Lucas(LL n,LL m,LL p){

LL ret=1;

**while**(n&&m){

LL a=n%p,b=m%p;

**if**(a<b) **return** 0;

ret=(ret\*fac[a]\*PowMod(fac[b]\*fac[a-b]%p,p-2,p))%p;

n/=p;

m/=p;

}

**return** ret;

}

int main(){

int t;

scanf("%d",&t);

**while**(t--){

LL n,m,p;

scanf("%I64d%I64d%I64d",&n,&m,&p);

Get\_Fact(p);

printf("%I64d**\n**",Lucas(n,m,p));

}

**return** 0;

}

*卢卡斯定理*

*O(logp(n)\*p)*

## 中国剩余定理（不互质情况）

**using** **namespace** std;

**const** int maxn=100005;

**const** int inf=0x7fffffff;

**typedef** long long ll;

void ex\_gcd(ll a,ll b,ll &d,ll &x,ll &y)*//扩展欧几里得*

{

**if**(!b) {d=a;x=1;y=0;}

**else**{

ex\_gcd(b,a%b,d,y,x);

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

}

}

ll ex\_crt(ll \*m,ll \*r,int n)

{

ll M=m[1],R=r[1],x,y,d;

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

ex\_gcd(M,m[i],d,x,y);

**if**((r[i]-R)%d) **return** -1;

x=(r[i]-R)/d\*x%(m[i]/d);

R+=x\*M;

M=M/d\*m[i];

R%=M;

}

**return** R>0?**R**:R+M;

}

int main()

{

int t,n;

scanf("%d",&t);

**for**(int cas=1;cas<=t;cas++){

scanf("%d",&n);

ll m[maxn],r[maxn];*//m除数，r余数*

**for**(int i=1;i<=n;i++) scanf("%lld",&m[i]);

**for**(int i=1;i<=n;i++) scanf("%lld",&r[i]);

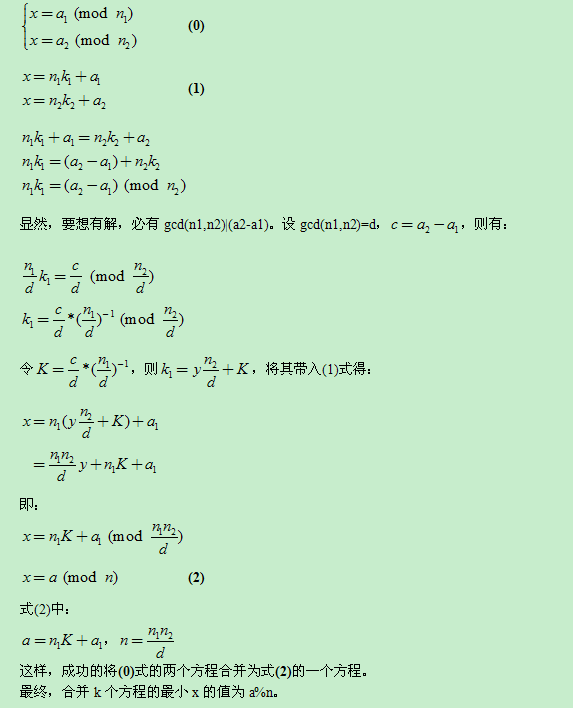
printf("Case %d: %I64d**\n**",cas,ex\_crt(m,r,n));

}

**return** 0;

}

关于原理的推导：



**typedef** **\_\_int64** int64;

int64 Mod;

int64 gcd(int64 a, int64 b)

{

**if**(b==0)

**return** a;

**return** gcd(b,a%b);

}

int64 Extend\_Euclid(int64 a, int64 b, int64&x, int64& y)

{

**if**(b==0)

{

x=1,y=0;

**return** a;

}

int64 d = Extend\_Euclid(b,a%b,x,y);

int64 t = x;

x = y;

y = t - a/b\*y;

**return** d;

}

*//a在模n乘法下的逆元，没有则返回-1*

int64 inv(int64 a, int64 n)

{

int64 x,y;

int64 t = Extend\_Euclid(a,n,x,y);

**if**(t != 1)

**return** -1;

**return** (x%n+n)%n;

}

*//将两个方程合并为一个*

bool merge(int64 a1, int64 n1, int64 a2, int64 n2, int64& a3, int64& n3)

{

int64 d = gcd(n1,n2);

int64 c = a2-a1;

**if**(c%d)

**return** false;

c = (c%n2+n2)%n2;

c /= d;

n1 /= d;

n2 /= d;

c \*= inv(n1,n2);

c %= n2;

c \*= n1\*d;

c += a1;

n3 = n1\*n2\*d;

a3 = (c%n3+n3)%n3;

**return** true;

}

*//求模线性方程组x=ai(mod ni),ni可以不互质*

int64 China\_Reminder2(int len, int64\* a, int64\* n)

{

int64 a1=a[0],n1=n[0];

int64 a2,n2;

**for**(int i = 1; i < len; i++)

{

int64 aa,nn;

a2 = a[i],n2=n[i];

**if**(!merge(a1,n1,a2,n2,aa,nn))

**return** -1;

a1 = aa;

n1 = nn;

}

Mod = n1;

**return** (a1%n1+n1)%n1;

}

int64 a[1000],b[1000];

int main()

{

int i;

int k;

**while**(scanf("%d",&k)!=EOF)

{

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

scanf("%I64d %I64d",&a[i],&b[i]);

printf("%I64d**\n**",China\_Reminder2(k,b,a));

}

**return** 0;

}

## 对前n个数分解质因数

*#include<bits/stdc++.h>*

**using** **namespace** std;

*#define N 2000000*

vector <pair<long long,int>> d[2000004];

void init()

{

d[1].push\_back({1,1});

**for**(long long i=2;i<=N;i++)

{

**if**(d[i].empty())

{

**for**(long long j=i;j<=N;j+=i)

d[j].push\_back({i,1});

long long w=i\*i;

**while**(w<=N)

{

**for**(long long j=w;j<=N;j+=w)

d[j][d[j].size()-1].second++;

w\*=i;

}

}

}

}

int main()

{

init();

int t;

**while**(cin >> t)

**for**(int i=0;i<d[t].size();i++)

{

cout << "factor " << d[t][i].first << " is " << d[t][i].second << endl;

}

}

## 将根式转换为连分数形式

*#include<bits/stdc++.h>*

**using** **namespace** std;

vector<int> a;

vector<int> b;

vector<int> c;

void get\_fractions(int n)

{

a.clear();

b.clear();

c.clear();

int AA;

AA = floor(sqrt(n));

int c0 = n-AA\*AA;

int a0 = (sqrt(n)+AA)/(n-AA\*AA);

int b0 = a0\*(n-AA\*AA)-AA;

c.push\_back(c0);

a.push\_back(a0);

b.push\_back(b0);

int i=0;

**do**

{

int ccc = (n - b[i]\*b[i])/c[i];

int aaa = (sqrt(n)+b[i])/ccc;

int bbb = aaa\*ccc-b[i];

**if**(a[0]==aaa&&b[0]==bbb&&c[0]==ccc)

**break**;

c.push\_back(ccc);

a.push\_back(aaa);

b.push\_back(bbb);

i++;

}**while**(1);

printf("[%d;(",AA);

vector<int>::iterator it;

**for**(it=a.begin();it!=a.end();it++)

cout<<\*it<<(it==a.end()-1?")]":",");

*//for(auto& x : a) cout << x;*

cout << endl;

}

int main()

{

int n;

**while**(cin >> n)

get\_fractions(n);

**return** 0;

}

## Meissel-Lehmer算法（求1e11内质数个数）

*//Meisell-Lehmer*

*//G++ 218ms 43252k*

*#include<cstdio>*

*#include<cmath>*

**using** **namespace** std;

*#define LL long long*

**const** int N = 5e6 + 2;

bool np[N];

int prime[N], pi[N];

int getprime()

{

int cnt = 0;

np[0] = np[1] = true;

pi[0] = pi[1] = 0;

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

{

**if**(!np[i]) prime[++cnt] = i;

pi[i] = cnt;

**for**(int j = 1; j <= cnt && i \* prime[j] < N; ++j)

{

np[i \* prime[j]] = true;

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

}

}

**return** cnt;

}

**const** int M = 7;

**const** int PM = 2 \* 3 \* 5 \* 7 \* 11 \* 13 \* 17;

int phi[PM + 1][M + 1], sz[M + 1];

void init()

{

getprime();

sz[0] = 1;

**for**(int i = 0; i <= PM; ++i) phi[i][0] = i;

**for**(int i = 1; i <= M; ++i)

{

sz[i] = prime[i] \* sz[i - 1];

**for**(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j / prime[i]][i - 1];

}

}

int sqrt2(LL x)

{

LL r = (LL)sqrt(x - 0.1);

**while**(r \* r <= x) ++r;

**return** int(r - 1);

}

int sqrt3(LL x)

{

LL r = (LL)cbrt(x - 0.1);

**while**(r \* r \* r <= x) ++r;

**return** int(r - 1);

}

LL getphi(LL x, int s)

{

**if**(s == 0) **return** x;

**if**(s <= M) **return** phi[x % sz[s]][s] + (x / sz[s]) \* phi[sz[s]][s];

**if**(x <= prime[s]\*prime[s]) **return** pi[x] - s + 1;

**if**(x <= prime[s]\*prime[s]\*prime[s] && x < N)

{

int s2x = pi[sqrt2(x)];

LL ans = pi[x] - (s2x + s - 2) \* (s2x - s + 1) / 2;

**for**(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];

**return** ans;

}

**return** getphi(x, s - 1) - getphi(x / prime[s], s - 1);

}

LL getpi(LL x)

{

**if**(x < N) **return** pi[x];

LL ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;

**for**(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans -= getpi(x / prime[i]) - i + 1;

**return** ans;

}

LL lehmer\_pi(LL x)

{

**if**(x < N) **return** pi[x];

int a = (int)lehmer\_pi(sqrt2(sqrt2(x)));

int b = (int)lehmer\_pi(sqrt2(x));

int c = (int)lehmer\_pi(sqrt3(x));

LL sum = getphi(x, a) +(LL)(b + a - 2) \* (b - a + 1) / 2;

**for** (int i = a + 1; i <= b; i++)

{

LL w = x / prime[i];

sum -= lehmer\_pi(w);

**if** (i > c) **continue**;

LL lim = lehmer\_pi(sqrt2(w));

**for** (int j = i; j <= lim; j++) sum -= lehmer\_pi(w / prime[j]) - (j - 1);

}

**return** sum;

}

int main()

{

init();

LL n;

**while**(~scanf("%lld",&n))

{

printf("%lld**\n**",lehmer\_pi(n));

}

**return** 0;

}

## Min\_25自动机（亚线性处理积性函数前缀和）

*#include<bits/stdc++.h>*

**using** **namespace** std;

*#define ll long long*

**const** int maxn = 2000;

**const** int N = 710000;

**const** int mod = 1e9+7;

int b[maxn],c[maxn][maxn],Inv[maxn];

ll sqr,n; */// sqr为sqrt(n)*

ll w[N],id1[N],id2[N];

int tot; *///记录对于要筛的n,sqrt(n)以内质数的个数*

int isp[N],p[N];

ll zh[N][3]; *///zh[i][k]记录(p[1])^k + (p[2])^k + ... + (p[i])^k*

ll g[N][3];

ll poww(ll a,int b)

{

ll ans = 1,base = a%mod;

**while**(b)

{

**if**(b&1)

{

ans\*=base;

ans%=mod;

}

base\*=base;

base%=mod;

b>>=1;

}

**return** ans;

}

ll sigma\_f(ll n,int k) *///得到∑i^k, i:1~n*

{

**if**(k==0) **return** n;

n++;

n%=mod;

ll tmp = n;

ll ans=0;

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

{

ans += 1LL\*c[k+1][i]\*b[k+1-i]%mod\*n%mod;

ans %= mod;

n \*= tmp%mod;

n %= mod;

}

ans \*= Inv[k+1];

ans %= mod;

ans += mod;

ans %= mod;

**return** ans;

}

void get\_p(int n,int w)

{

tot = 0;

memset(isp,1,**sizeof**(isp));

isp[0] = 0;

isp[1] = 0;

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

{

**if**(isp[i])

{

p[++tot] = i;

ll wait = 1;

**for**(int j=0;j<=w;j++)

{

zh[tot][j] = zh[tot-1][j] + wait;

zh[tot][j] %= mod;

wait \*= i;

}

}

**for**(int j=1;p[j]\*i<=n&&j<=i;j++)

{

isp[i\*p[j]] = 0;

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

}

}

}

void get\_g(ll n,int t) *///对每个x=n/i,求出∑[i是质数](i^t) (i from 1 to x)。每个对应的值存储在g[x][t]中*

{

int m = 0;

ll i=1,r;

**while**(i<=n)

{

ll len = n/i;

r = n/len;

**if**(len<=sqr) id1[len] = ++m;

**else** id2[r] = ++m;

**for**(int ww=0;ww<=t;ww++)

{

g[m][ww] = sigma\_f(len,ww)-1;

g[m][ww] %= mod;

g[m][ww] += mod;

g[m][ww] %= mod;

}

w[m] = len; *///w[i]记录了形如n/k的第i大的取值是多少*

i = r+1;

}

**for**(int i=1;i<=tot;i++)

{

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

{

**if**(1LL\*p[i]\*p[i]>w[j]) **break**;

**else**

{

int op;

**if**(w[j]/p[i]<=sqr) op = id1[w[j]/p[i]];

**else** op = id2[n/(w[j]/p[i])];

**for**(int ww=0;ww<=t;ww++)

{

g[j][ww] = g[j][ww] - poww(p[i],ww)\*((g[op][ww]-zh[i-1][ww])%mod);

g[j][ww] %= mod;

g[j][ww] += mod;

g[j][ww] %= mod;

}

}

}

}

}

**inline** ll get\_value(int wz,int k)

{

ll w = (g[wz][2] + 2\*g[wz][1] - g[wz][0]) - (zh[k-1][2] + 2\*zh[k-1][1] - zh[k-1][0]);

w %= mod;

w += mod;

w %= mod;

*//ll w = (g[wz][1]-g[wz][0])-(zh[k-1][1]-zh[k-1][0]);*

**return** w; *///自己填写f(x)的表达式（在质数时）*

*///比如f(x) = x^2 + 2\*x - 1，就写(g[wz][2] + 2\*g[wz][1] - g[wz][0]) - (zh[k-1][2] + 2\*zh[k-1][1] - zh[k-1][0])*

}

ll f(ll p,ll k) *///计算f(p^k)处的值*

{

**if**(k==1) **return** (p\*p+2\*p-1)%mod;

**return** -3; *///自己填写*

}

ll get\_s(ll x,int k)

{

**if**(x<=1||p[k]>x) **return** 0;

int wz;

**if**(x<=sqr) wz = id1[x];

**else** wz = id2[n/x];

ll ans = get\_value(wz,k);

*//if(k==1) ans += 2;*

**for**(int i=k;i<=tot&&1LL\*p[i]\*p[i]<=x;++i)

{

**for**(ll l=p[i],e=1;l\*p[i]<=x;l=l\*p[i],++e)

{

ans = ans + (get\_s(x/l,i+1)\*f(p[i],e)%mod)%mod+f(p[i],e+1);

ans %= mod;

}

}

ans += mod;

ans %= mod;

**return** ans;

}

void init()

{

c[0][0]=1;

**for** (int i=1;i<maxn;i++)

{

**for** (int j=1;j<=i;j++) c[i][j]=(c[i-1][j-1]+c[i-1][j]) % mod;

c[i][0]=1;

}

Inv[1]=1;

**for** (int i=2;i<maxn;i++) Inv[i]=1LL\*Inv[mod % i] \* (mod-mod/i) % mod;

b[0]=1;

**for** (int i=1;i<maxn;i++)

{

b[i]=0;

**for** (int k=0;k<i;k++) b[i]=(b[i]+1LL\*c[i+1][k]\*b[k] % mod) % mod;

b[i]=(1LL\*b[i]\*(-Inv[i+1]) % mod+mod)%mod;

}

}

void solve(ll n)

{

init();

sqr = sqrt(n);

get\_p(sqr,2);

get\_g(n,2);

ll ans = get\_s(n,1)+1;

cout << ans << endl;

}

int main()

{

**while**(cin >> n)

{solve(n);}

}

## 求解同余方程(x^2=a(mod p))

ll poww(ll a,ll b,ll p)

{

ll ans = 1,base = a;

**while**(b)

{

**if**(b&1)

{

ans\*=base;

ans %= p;

}

base\*= base;

base %= p;

b>>=1;

}

**return** ans;

}

ll b[20],m[20];

ll shank(ll a,ll p) *///get x^2=a(mod p)*

{

**if**(!isprime[p]) **return** -1;

a%=p;

a+=p;

a%=p;

**if**(poww(a,(p-1)/2,p)!=1) **return** -1;

ll q = p-1;

ll k = 0;

**while**(q%2==0)

{

k++;

q/=2;

}

ll f;

**for**(int i=2;i<p;i++)

{

**if**(poww(i,(p-1)/2,p)!=1)

{

f = i;

**break**;

}

}

ll g = poww(f,q,p);

b[1] = poww(a,q,p);

int i = 1;

**while**(b[i]!=1)

{

m[i] = 1;

int wa = 2;

**while**(poww(b[i],wa,p)!=1)

{

m[i]++;

wa \*= 2;

}

i++;

b[i] = b[i-1]%p\*poww(g,poww(2,k-m[i-1],p-1),p)%p;

}

int r = i-1;

ll x = poww(a,(q+1)/2,p);

ll times = 0;

**for**(int i=1;i<=r;i++)

{

times += poww(2,k-1-m[i],p-1);

}

x \*= poww(g,times,p);

x %= p;

**return** x;

}

## 类欧几里得算法

求解 : 

ll f\_sqr(ll a,ll b,ll c,ll n,ll r)

{

double w = sqrt(r);

**if**(n==0) **return** 0;

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

ll gg = \_\_gcd(a,\_\_gcd(b,c));

a/=gg,b/=gg,c/=gg;

ll res = (ll)(w\*a + 1.0\*b)/(1.0\*c);

**if**(res==0)

{

ll gcd = \_\_gcd(a\*c,\_\_gcd(b\*c,a\*a\*r-b\*b));

ll nn = (w\*a + 1.0\*b)/(1.0\*c)\*n;

**return** nn\*n - f\_sqr(a\*c/gcd,b\*c\*(-1)/gcd,(a\*a\*r-b\*b)/gcd,nn,r);

}

**else** **return** n\*(n+1)/2\*res + f\_sqr(a,b-res\*c,c,n,r);

}

求解 : 

ll f(ll a,ll b,ll c,ll n) ///Sigma\_{i=0}^{n} floor((a\*i+b)/c)

{

ll m = (a\*n+b)/c;

**if**(n==0||m==0) **return** (b/c);

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

**if**(a<c&&b<c) **return** m\*n - f(c,c-b-1,a,m-1);

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

}

求解 : 

ll g(ll a,ll b,ll c,ll n) ///Sigma\_{i=0}^{n} i\*floor((a\*i+b)/c)

{

ll m = (a\*n+b)/c;

**if**(n==0||m==0) **return** 0;

**if**(a<c&&b<c) **return** ((n+1)\*n\*m - f(c,c-b-1,a,m-1) - h(c,c-b-1,a,m-1))/2;

**else** **return** g(a%c,b%c,c,n) + (a/c)\*n\*(n+1)\*(2\*n+1)/6 + (b/c)\*n\*(n+1)/2;

}

求解 : 

ll h(ll a,ll b,ll c,ll n) ///Sigma\_{i=0}^{n} (floor((a\*i+b)/c))^2

{

ll m = (a\*n+b)/c;

**if**(n==0||m==0) **return** (b/c)\*(b/c);

**if**(a<c&&b<c) **return** n\*m\*(m+1) - g(c,c-b-1,a,m-1)\*2 - f(c,c-b-1,a,m-1)\*2 - f(a,b,c,n);

**else** **return** h(a%c,b%c,c,n) + (a/c)\*(a/c)\*n\*(n+1)\*(2\*n+1)/6 + (a/c)\*(b/c)\*n\*(n+1) + (b/c)\*(b/c)\*(n+1) + (a/c)\*g(a%c,b%c,c,n)\*2 + (b/c)\*f(a%c,b%c,c,n)\*2;

}

## 预处理狄利克雷卷积（O(nlnn)）

int f[MAXN],g[MAXN],h[MAXN]={0};

void calc(int n)

{

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

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

**if**(j==i)h[i\*j]+=f[i]\*g[i];

**else** h[i\*j]+=f[i]\*g[j]+f[j]\*g[i];

}

# 字符串

## KMP字符串匹配

**const** int MAX = 200;

int nextt[MAX];

void getNext(string t)

{

int j, k;

memset(nextt,0,**sizeof**(nextt));

j = 0; k = -1; nextt[0] = -1;

**while**(j < t.length())

{

**if**(k == -1 || t[j] == t[k])

nextt[++j] = ++k;

**else**

k = nextt[k];

}

}

*/\**

*返回模式串t在主串s中首次出现的位置*

*返回的位置是从0开始的。*

*\*/*

int KMP\_Index(string t,string s)

{

int i = 0, j = 0;

getNext(t);

**while**(i < s.length() && j < t.length())

{

**if**(j == -1 || s[i] == t[j])

{

i++; j++;

}

**else**

j = nextt[j];

}

**if**(j == t.length())

**return** i - t.length();

**else**

**return** -1;

}

*/\**

*返回模式串t在主串s中出现的次数*

*\*/*

int KMP\_Count(string t,string s)

{

int ans = 0;

int i, j = 0;

**if**(s.length() == 1 && t.length() == 1)

{

**if**(s[0] == t[0])

**return** 1;

**else**

**return** 0;

}

getNext(t);

**for**(i = 0; i < s.length(); i++)

{

**while**(j > 0 && s[i] != t[j])

j = nextt[j];

**if**(s[i] == t[j])

j++;

**if**(j == t.length())

{

ans++;

j = nextt[j];

}

}

**return** ans;

}

## Manacher(回文串计算)

char s[500000];

char s\_new[1000000+10];

int pos[1000000+10];

int manacher(char \*s){

int len = strlen(s);

s\_new[0] = '$';

s\_new[1] = '#';

**for**(int i=0;i<len;i++){

s\_new[2\*(i+1)] = s[i];

s\_new[2\*(i+1)+1] = '#';

}

s\_new[2\*(len+1)] = '\0';

int len2 = strlen(s\_new);

int mx = 0,ans = 0,po = 0;

**for**(int i=1;i<len2;i++){

**if**(i<mx) pos[i] = min(pos[2\*po-i],mx-i);

**else** pos[i] = 1;

**while**(s\_new[i-pos[i]]==s\_new[i+pos[i]]) pos[i]++;

**if**(mx<i+pos[i]){

po = i;

mx = i+pos[i];

}

ans = max(ans,pos[i]-1);

}

**return** ans;

}

int main(){

**while** (printf("请输入字符串：**\n**")){

scanf("%s", s);

printf("最长回文长度为 %d**\n\n**", manacher(s));

**for**(int i=0;i<strlen(s\_new);i++) cout << s\_new[i]; cout << endl;

**for**(int i=0;i<strlen(s\_new);i++) cout << pos[i]; cout << endl;

}

**return** 0;

}

# 数据结构

## 并查集

*#define MAX 10000;*

**struct** UF

{

int ranking;

int parent;

}UF[MAX];

void init(int n)

{

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

{

UF[i].parent=i;

UF[i].ranking=0;

}

}

int get\_parent(int x)

{

**if**(UF[x].parent==x) **return** x;

**return** get\_parent(UF[x].parent);

}

void Union(int a,int b)

{

a=get\_parent(a);

b=get\_parent(b);

**if**(UF[a].rank>UF[b].rank) UF[b].parent = UF[a].parent;

**else**

{

UF[a].parent = UF[b].parent;

**if**(UF[a].rank==UF[b].rank) UF[a].rank++;

}

}

## 链表

**const** int SIZE = 500000+5;

int tot,head,tail;

**struct** node

{

int value;

int prev,next;

}node[SIZE];

int \_init()

{

tot = 2;

head = 1,tail = 2;

node[head].next = tail;

node[tail].prev = head;

}

int \_insert(int p,int val)

{

int q = ++tot;

node[q].value = val;

node[node[p].next].prev = q;

node[q].next = node[p].next;

node[p].next = q;

node[q].prev = p;

}

void \_remove(int p)

{

node[node[p].prev].next = node[p].next;

node[node[p].next].prev = node[p].prev;

}

void \_clear()

{

memset(node,0,**sizeof**(node));

head = tail = tot = 0;

}

## 线段树

**const** int maxn = 100007;

**struct** Tree

{

int l,r,sum;

int vis;

}t[maxn<<2];

void push\_up(int step)

{

t[step].sum = t[step\*2].sum + t[step\*2+1].sum;

}

void push\_down(int step)

{

**if**(!t[step].vis) **return**;

t[step\*2].vis += t[step].vis;

t[step\*2+1].vis += t[step].vis;

t[step\*2].sum += t[step].vis\*(t[step\*2].r-t[step\*2].l+1);

t[step\*2+1].sum += t[step].vis\*(t[step\*2+1].r-t[step\*2+1].l+1);

t[step].vis = 0;

}

void build(int l,int r,int step)

{

t[step].l = l,t[step].r = r,t[step].sum = t[step].vis = 0;

**if**(l==r) **return**;

int mid = (l+r)/2;

build(l,mid,step\*2);

build(mid+1,r,step\*2+1);

}

void update(int l,int r,int val,int step)

{

**if**(l==t[step].l&&r==t[step].r)

{

t[step].vis += val;

t[step].sum += (r-l+1)\*val;

**return**;

}

int mid = (t[step].l+t[step].r)/2;

push\_down(step);

**if**(r<=mid) update(l,r,val,step\*2);

**else** **if**(l>mid) update(l,r,val,step\*2+1);

**else** update(l,mid,val,step\*2),update(mid+1,r,val,step\*2+1);

push\_up(step);

}

int query(int l,int r,int step)

{

**if**(l==t[step].l&&r==t[step].r)

**return** t[step].sum;

int mid = (t[step].l+t[step].r)/2;

push\_down(step);

**if**(r<=mid) **return** query(l,r,step\*2);

**else** **if**(l>mid) **return** query(l,r,step\*2+1);

**else** **return** query(l,mid,step\*2)+query(mid+1,r,step\*2+1);

}

## zkw线段树

## 主席树

## 树状数组

*#include<bits/stdc++.h>*

**using** **namespace** std;

**const** int maxn = 100007;

int c[maxn];

int lowbit(int n)

{

**return** n & (-n);

}

int query(int x)

{

int res = 0;

**while**(x>0)

{

res += c[x];

x -= lowbit(x);

}

**return** res;

}

int update(int x,int val)

{

**while**(x<maxn)

{

c[x] += val;

x += lowbit(x);

}

}

int main()

{

int n;

cin >> n;

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

{

int x,y;

cin >> x >> y;

update(x,y);

}

cout << "give me your question!" << endl;

int l,r;

**while**(cin >> l >> r)

{

cout << "In [" << l << "," << r << "] ,all the number's sum is: " << query(r)-query(l-1) << endl;

}

}

## 树链剖分

**const** int MAX = 100000+5;

**const** int mod = 1e9+7;

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

int head[MAX]; *///head[x]存储以x为起点的第一条边的编号 如果没有以x为起点的边，head[x]=0;*

int cnt = 0;

**struct** edge

{

int next; *///e[i].next存储与i号边同起点的下一条有向边的编号，如果i号边已经是最后一条边,e[i].next=0;*

int to; *///e[i].to存储i号边所指向的终点(点的编号)*

}e[MAX<<1];

void add\_edge(int x,int y)

{

e[++cnt].next = head[x];*///建新边，标号为i = cnt+1，让当前以x为起点的第一条边作为他的下一条边(为接下来将他设为以x为起点的第一条边做铺垫)*

e[cnt].to = y; *///这条边指向了y节点*

head[x] = cnt; *///将这条边作为以x为起点的第一条边*

}

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

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

int f[MAX],d[MAX],sz[MAX],heavy\_son[MAX];

int rk[MAX],top[MAX],id[MAX];

void dfs1(int u,int fa,int depth) *///当前节点 他的爸爸 当前深度*

{

f[u] = fa;

d[u] = depth;

sz[u] = 1;

**for**(int i=head[u];i!=0;i=e[i].next)

{

int v = e[i].to;

**if**(v==fa) **continue**;*///根*

dfs1(v,u,depth+1);

sz[u] += sz[v];

**if**(sz[v]>sz[heavy\_son[u]])

heavy\_son[u] = v; *///重儿子的size最大*

}

}

int cnt2 = 0;

void dfs2(int u,int t) *///当前节点 重链的顶端*

{

top[u] = t;

id[u] = ++cnt2; *///标记dfs序*

rk[cnt2] = u; *///构造一个从新dfs序的标号 -> 原始节点标号的映射*

**if**(heavy\_son[u]==0) **return**;*///根*

dfs2(heavy\_son[u],t);

**for**(int i=head[u];i!=0;i=e[i].next)

{

int v=e[i].to;

**if**(v!=heavy\_son[u]&&v!=f[u]) dfs2(v,v);

}

}

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

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

int dis(int x,int y)

{

int res = 0;

int fx = top[x],fy = top[y];

**while**(fx!=fy)

{

**if**(d[fx]>=d[fy])

{

res += d[x] - d[fx];

res += d[fx] - d[f[fx]];

x = f[fx];

fx = top[x];

}

**else**

{

res += d[y] - d[fy];

res += d[fy] - d[f[fy]];

y = f[fy],fy = top[y];

}

}

*//cout << id[x] << " " << id[y] << endl;*

**if**(id[x]<=id[y])

{

res += id[y]-id[x];

}

**else** res += id[x] - id[y];

**return** res;

}

ll sum(int x,int y)

{

ll ans = 0;

int fx=top[x],fy=top[y];

**while**(fx!=fy) *///两点不在同一条重链*

{

**if**(d[fx]>=d[fy])

{

ans+=query(id[fx],id[x],1);

ans %= mod; *///线段树区间求和，处理这条重链的贡献*

x=f[fx],fx=top[x]; *///将x设置成原链头的父亲结点，走轻边，继续循环*

}

**else**

{

ans+=query(id[fy],id[y],1);

ans %= mod;

y=f[fy],fy=top[y];

}

}

*///循环结束，两点位于同一重链上，但两点不一定为同一点，统计这两点之间的贡献*

**if**(id[x]<=id[y])

ans+=query(id[x],id[y],1),ans%=mod;

**else**

ans+=query(id[y],id[x],1),ans%=mod;

**return** ans;

}

void updates(int x,int y,ll c) *///x->y的路径上第一个点+c，第二个点+2\*c，...*

{

int fx=top[x],fy=top[y];

int dis\_xy = dis(x,y);

ll lazy1 = c;

ll lazy2 = c;

**while**(fx!=fy)

{

**if**(d[fx]>=d[fy])

{

ll dd = id[x] - id[fx];

update(id[fx],id[x],(1LL\*lazy1 + (dd)\*c)%mod,(1LL\*(-1)\*lazy2)%mod,1);

x=f[fx];

lazy1 = ((1LL\*c\*(dd+1) + lazy1)%mod);

lazy1 %= mod;

}

**else**

{

ll dd = dis(x,fy);

update(id[fy],id[y],(1LL\*lazy1+lazy2\*dd)%mod,lazy2,1);

y=f[fy];

}

fx=top[x];

fy=top[y];

}

**if**(id[x]<=id[y])

{

update(id[x],id[y],lazy1,lazy2,1);

}

**else**

{

ll dd = dis(x,y);

update(id[y],id[x],(1LL\*lazy1+1LL\*lazy2\*dd%mod)%mod,(1LL\*(-1)\*lazy2)%mod,1);

}

}

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

# 计算几何

## 基础模板

**struct** spot *///存储点，也可指代向量*

{

double x;

double y;

double z;

};

spot cross(**const** spot &a,**const** spot &b) *///计算向量a和向量b的叉乘(有顺序)*

{

spot w;

w.x = a.y\*b.z-b.y\*a.z;

w.y = a.z\*b.x-a.x\*b.z;

w.z = a.x\*b.y-a.y\*b.x;

**return** w;

}

double dot(**const** spot &a,**const** spot &b) *///计算向量a和向量b的点乘积*

{

**return** a.x\*b.x+a.y\*b.y+a.z\*b.z;

}

double norm(**const** spot &a) *///计算向量a的模长*

{

**return** sqrt(a.x\*a.x+a.y\*a.y+a.z\*a.z);

}

## 最大空凸包

T组测试数据，每组n个点

求出n个点以其中若干个点为顶点的最大凸多边形面积，要求多边形内部不能有其它点

复杂度O(n^3)

*#include<bits/stdc++.h>*

**using** **namespace** std;

*#define LL long long*

*#define mod 1000000007*

**typedef** **struct** Point

{

int x, y;

Point() {}

Point(int x, int y): x(x),y(y) {}

Point **operator** + (**const** Point &b) **const** { **return** Point(x+b.x,y+b.y); }

Point **operator** - (**const** Point &b) **const** { **return** Point(x-b.x,y-b.y); }

int **operator** \* (**const** Point &b) **const** { **return** x\*b.y-y\*b.x; }

int len() **const** { **return** x\*x+y\*y; }

int **operator** < (**const** Point &a) **const**

{

**if**((\***this**)\*a>0 || (\***this**)\*a==0 && len()<a.len())

**return** 1;

**return** 0;

}

}Point;

int n;

Point s[122], p[122];

int dp[122][122];

int Jud(int m)

{

int ans, i, j, now, k, flag, S;

memset(dp, 0, **sizeof**(dp));

ans = 0;

**for**(i=2;i<=m;i++)

{

now = i-1;

**while**(now>=1 && p[i]\*p[now]==0)

now--;

flag = 0;

**if**(now==i-1)

flag = 1;

**while**(now>=1)

{

S = p[now]\*p[i];

k = now-1;

**while**(k>=1 && (p[now]-p[i])\*(p[k]-p[now])>0)

k--;

**if**(k>=1)

S += dp[now][k];

**if**(flag)

dp[i][now] = S;

ans = max(ans, S);

now = k;

}

**if**(flag==0)

**continue**;

**for**(j=1;j<=i-1;j++)

dp[i][j] = max(dp[i][j],dp[i][j-1]);

}

**return** ans;

}

int main(void)

{

int T, i, j, m, ans;

scanf("%d", &T);

**while**(T--)

{

scanf("%d", &n);

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

scanf("%d%d", &s[i].x, &s[i].y);

ans = 0;

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

{

m = 0;

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

{

**if**(s[j].y>s[i].y || s[j].y==s[i].y && s[j].x>=s[i].x)

p[++m] = s[j]-s[i];

}

sort(p+1, p+m+1);

ans = max(ans, Jud(m));

}

printf("%.1f**\n**", ans/2.0);

}

**return** 0;

}

# 杂项

## 罗马-数字转换

*#include<bits/stdc++.h>*

**using** **namespace** std;

char str[10]="IVXLCDM";

int num[10]={1,5,10,50,100,500,1000};

int roman\_to\_num(char s[])

{

int len=strlen(s);

int cnt=0,a[20];

**for**(int i=0;i<len;i++)

{

int f=0,t;

**if**(s[i]==s[i+1]&&i!=len-1)

{

**if**(s[i]==s[i+2]&&i!=len-2)

{

f=2;

}

**else**

f=1;

}

**for**(int k=0;k<7;k++)

**if**(s[i]==str[k])

{

t=k;

**break**;

}

a[cnt++]=num[t]\*(f+1);

i+=f;

}

int sum=0;

**for**(int i=0;i<cnt;i++)

sum+=a[i];

**for**(int i=0;i<cnt-1;i++)

{

**if**(a[i]<a[i+1])sum-=2\*a[i];

}

**return** sum;

}

string num\_to\_roman(int num)

{

char\* digit[10] = {"","I","II","III","IV","V","VI","VII","VIII","IX"};

char\* ten[10] = {"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"};

char\* hundreds[10] = {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"};

char\* thousand[7] = {"","M","MM","MMM","MMMM","MMMMM","MMMMMM"};

string ans;

ans = string(thousand[num/1000]) + string(hundreds[num%1000/100]) + string(ten[num%100/10]) + string(digit[num%10]);

**return** ans;

}

int main()

{

char a[101];

scanf("%s",a);

cout << roman\_to\_num(a) << endl;

cout << num\_to\_roman(roman\_to\_num(a)) << endl;

}

## Bm算法求解线性递推

*#include* *<bits/stdc++.h>*

**using** **namespace** std;

*#define rep(i,a,n) for (long long i=a;i<n;i++)*

*#define per(i,a,n) for (long long i=n-1;i>=a;i--)*

*#define pb push\_back*

*#define mp make\_pair*

*#define all(x) (x).begin(),(x).end()*

*#define fi first*

*#define se second*

*#define SZ(x) ((long long)(x).size())*

**typedef** vector<long long> VI;

**typedef** long long ll;

**typedef** pair<long long,long long> PII;

**const** ll mod=1e9+7;

ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); **for**(;b;b>>=1){**if**(b&1)res=res\*a%mod;a=a\*a%mod;}**return** res;}

*// head*

long long \_,n;

**namespace** linear\_seq

{

**const** long long N=10010;

ll res[N],base[N],\_c[N],\_md[N];

vector<long long> Md;

void mul(ll \*a,ll \*b,long long k)

{

rep(i,0,k+k) \_c[i]=0;

rep(i,0,k) **if** (a[i]) rep(j,0,k)

\_c[i+j]=(\_c[i+j]+a[i]\*b[j])%mod;

**for** (long long i=k+k-1;i>=k;i--) **if** (\_c[i])

rep(j,0,SZ(Md)) \_c[i-k+Md[j]]=(\_c[i-k+Md[j]]-\_c[i]\*\_md[Md[j]])%mod;

rep(i,0,k) a[i]=\_c[i];

}

long long solve(ll n,VI a,VI b)

{ *// a 系数 b 初值 b[n+1]=a[0]\*b[n]+...*

*// printf("%d\n",SZ(b));*

ll ans=0,pnt=0;

long long k=SZ(a);

assert(SZ(a)==SZ(b));

rep(i,0,k) \_md[k-1-i]=-a[i];\_md[k]=1;

Md.clear();

rep(i,0,k) **if** (\_md[i]!=0) Md.push\_back(i);

rep(i,0,k) res[i]=base[i]=0;

res[0]=1;

**while** ((1ll<<pnt)<=n) pnt++;

**for** (long long p=pnt;p>=0;p--)

{

mul(res,res,k);

**if** ((n>>p)&1)

{

**for** (long long i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;

rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]\*\_md[Md[j]])%mod;

}

}

rep(i,0,k) ans=(ans+res[i]\*b[i])%mod;

**if** (ans<0) ans+=mod;

**return** ans;

}

VI BM(VI s)

{

VI C(1,1),B(1,1);

long long L=0,m=1,b=1;

rep(n,0,SZ(s))

{

ll d=0;

rep(i,0,L+1) d=(d+(ll)C[i]\*s[n-i])%mod;

**if** (d==0) ++m;

**else** **if** (2\*L<=n)

{

VI T=C;

ll c=mod-d\*powmod(b,mod-2)%mod;

**while** (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

L=n+1-L; B=T; b=d; m=1;

}

**else**

{

ll c=mod-d\*powmod(b,mod-2)%mod;

**while** (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

++m;

}

}

**return** C;

}

long long gao(VI a,ll n)

{

VI c=BM(a);

c.erase(c.begin());

rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;

**return** solve(n,c,VI(a.begin(),a.begin()+SZ(c)));

}

};

int main()

{

int t;

cin >> t;

**while**(t--)

{

int n;

cin >> n;

printf("%I64d**\n**",linear\_seq::gao(VI{2,24,96,416,1536,5504,18944,64000,212992,702464},n-1));

}

}

## FFT求多项式乘法

*#include* *<map>*

*#include* *<set>*

*#include* *<cmath>*

*#include* *<ctime>*

*#include* *<stack>*

*#include* *<queue>*

*#include* *<cstdio>*

*#include* *<cctype>*

*#include* *<bitset>*

*#include* *<string>*

*#include* *<vector>*

*#include* *<cstring>*

*#include* *<iostream>*

*#include* *<algorithm>*

*#include* *<functional>*

*#define fuck(x) cout<<"["<<x<<"]";*

*#define FIN freopen("input.txt","r",stdin);*

*#define FOUT freopen("output.txt","w+",stdout);*

*//#pragma comment(linker, "/STACK:102400000,102400000")*

**using** **namespace** std;

**typedef** long long LL;

**typedef** pair<int, int> PII;

**const** int MX = 3e5 + 5;

**const** int INF = 0x3f3f3f3f;

**const** int mod = 1e9 + 7;

**const** double pi = acos(-1.0);

**struct** cp {

double x, y;

cp() {}

cp (double x, double y): x(x), y(y) {}

**inline** cp **operator** + (**const** cp &b) {

**return** cp(x + b.x, y + b.y);

}

**inline** cp **operator** - (**const** cp &b) {

**return** cp(x - b.x, y - b.y);

}

**inline** cp **operator** \* (**const** cp &b) {

**return** cp(x \* b.x - y \* b.y, x \* b.y + y \* b.x);

}

} a[MX], b[MX];

int r[MX];

void fft(cp a[], int opt, int n) {

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

**if**(i < r[i]) swap(a[i], a[r[i]]);

}

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

cp wn(cos(pi / i), opt \* sin(pi / i));

**for**(int p = i << 1, j = 0; j < n; j += p) {

cp w(1, 0);

**for**(int k = 0; k < i; k++, w = wn \* w) {

cp x = a[j + k], y = w \* a[j + k + i];

a[j + k] = x + y; a[j + k + i] = x - y;

}

}

}

}

*/\*多项式a，最高次为n，多项式b，最高次为m*

*从0到n项的系数*

*卷积结果等于后来a[].x*

*复杂度O(nlogn)，最后的最高项为n+m*

*\*/*

void solve(cp a[], cp b[], int n, int m) {

int l = 0, nn, nm = n + m;

**for**(nn = 1; nn <= nm; nn <<= 1) l++;

**for**(int i = n + 1; i <= nn; i++) a[i] = cp(0, 0);

**for**(int i = m + 1; i <= nn; i++) b[i] = cp(0, 0);

n = nn; m = nm;

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

r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));

}

fft(a, 1, n); fft(b, 1, n);

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

a[i] = a[i] \* b[i];

}

fft(a, -1, n);

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

a[i].x /= n;

}

}

int main() {

int n, m; *//FIN;*

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

**for**(int i = 0; i <= n; i++) scanf("%lf", &a[i].x);

**for**(int i = 0; i <= m; i++) scanf("%lf", &b[i].x);

solve(a, b, n, m);

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

printf("%d%c", (int)(a[i].x + 0.5), i == n + m ? '\n' : ' ');

}

}

## 拉格朗日插值

*#include<iostream>*

*#include<string>*

*#include<vector>*

**using** **namespace** std;

double Lagrange(int N,vector<double>&X,vector<double>&Y,double x);

int main(){

char a='n';

**do**{

cout<<"请输入差值次数n的值："<<endl;

int N;

cin>>N;

vector<double>X(N,0);

vector<double>Y(N,0);

cout<<"请输入插值点对应的值及函数值(Xi,Yi)："<<endl;

**for**(int a=0;a<N;a++){

cin>>X[a]>>Y[a];

}

cout<<"请输入要求值x的值："<<endl;

double x;

cin>>x;

double result=Lagrange(N,X,Y,x);

cout<<"由拉格朗日插值法得出结果： "<<result<<endl;

cout<<"是否要继续？(y/n)：";

cin>>a;

}**while**(a=='y');

**return** 0;

}

double Lagrange(int N,vector<double>&X,vector<double>&Y,double x){

double result=0;

**for**(int i=0;i<N;i++){

double temp=Y[i];

**for**(int j=0;j<N;j++){

**if**(i!=j){

temp = temp\*(x-X[j]);

temp = temp/(X[i]-X[j]);

}

}

result += temp;

}

**return** result;

};

## 高速大数（longlong）运算

*#include* *<bits/stdc++.h>*

**using** **namespace** std;

*#ifndef ONLINE\_JUDGE*

*#define dbg(args...) \*

*do \*

*{ \*

*cout << "\033[32;1m" << #args << " -> "; \*

*err(args); \*

*} while (0)*

*#else*

*#define dbg(...)*

*#endif*

void err()

{

cout << "**\033**[39;0m" << endl;

}

**template** <**template** <**typename**...> **class** **T**, **typename** t, **typename**... Args>

void err(T<t> a, Args... args)

{

**for** (**auto** **x** : a) cout << x << ' ';

err(args...);

}

**template** <**typename** T, **typename**... Args>

void err(T a, Args... args)

{

cout << a << ' ';

err(args...);

}

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

**typedef** long long ll;

**typedef** unsigned long long ull;

**using** i64 = long long;

**using** u64 = unsigned long long;

**using** u128 = \_\_uint128\_t;

**struct** Mod64

{

Mod64() : n\_(0) {}

Mod64(u64 n) : n\_(init(n)) {}

**static** u64 modulus() { **return** mod; }

**static** u64 init(u64 w) { **return** reduce(u128(w) \* r2); }

**static** void set\_mod(u64 m)

{

mod = m;

assert(mod & 1);

inv = m;

**for** (int i = 0; i < 5; ++i) inv \*= 2 - inv \* m;

r2 = -u128(m) % m;

}

**static** u64 reduce(u128 x)

{

u64 y = u64(x >> 64) - u64((u128(u64(x) \* inv) \* mod) >> 64);

**return** i64(y) < 0 ? y + **mod** : y;

}

Mod64& **operator**+=(Mod64 rhs)

{

n\_ += rhs.n\_ - mod;

**if** (i64(n\_) < 0) n\_ += mod;

**return** \***this**;

}

Mod64 **operator**+(Mod64 rhs) **const** { **return** Mod64(\***this**) += rhs; }

Mod64& **operator**\*=(Mod64 rhs)

{

n\_ = reduce(u128(n\_) \* rhs.n\_);

**return** \***this**;

}

Mod64 **operator**\*(Mod64 rhs) **const** { **return** Mod64(\***this**) \*= rhs; }

u64 get() **const** { **return** reduce(n\_); }

**static** u64 mod, inv, r2;

u64 n\_;

};

u64 Mod64::mod, Mod64::inv, Mod64::r2;

int main()

{

int T;

scanf("%d", &T);

**while** (T--)

{

ull A0, A1, M0, M1, C, M;

int k;

scanf("%llu%llu%llu%llu%llu%llu%d", &A0, &A1, &M0, &M1, &C, &M, &k);

Mod64::set\_mod(M);

Mod64 a0(A0), a1(A1), m0(M0), m1(M1), c(C), ans(1), a2(0);

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

{

ans = ans \* a0;

a2 = m0 \* a1 + m1 \* a0 + c;

a0 = a1;

a1 = a2;

}

printf("%llu**\n**", ans.get());

}

}

## 自己的FFT

*#include<bits/stdc++.h>*

**using** **namespace** std;

**const** int MAX = 500000;

**const** double pi = 3.141592653589793;

**struct** c\_number

{

double real;

double imag;

};

c\_number multi(c\_number a,c\_number b)

{

c\_number c;

c.real = (a.real\*b.real) - (a.imag\*b.imag);

c.imag = (a.real\*b.imag) + (a.imag\*b.real);

**return** c;

}

int Inv[MAX+5];

void getInv(int bit){

**for**(int i=0;i<(1<<bit);i++)

{

Inv[i]=(Inv[i>>1]>>1)|((i&1)<<(bit-1));

}

}

void fft(c\_number \*a,int n,double mode)

{

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

**if**(i<Inv[i]) swap(a[i],a[Inv[i]]);

**for**(int i=2;i<=n;i<<=1)

{

c\_number w\_stage;

w\_stage.real = cos(2.0\*mode\*pi/i);

w\_stage.imag = sin(2.0\*mode\*pi/i);

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

{

c\_number w;

w.real = 1;

w.imag = 0;

**for**(int k=j;k<j+(i>>1);k++)

{

c\_number st,fly;

st = a[k];

fly = multi(a[k+(i>>1)],w);

a[k].imag = st.imag + fly.imag;

a[k].real = st.real + fly.real;

a[k+(i>>1)].imag = st.imag - fly.imag;

a[k+(i>>1)].real = st.real - fly.real;

w = multi(w,w\_stage);

}

}

}

**if**(mode==-1)

{

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

a[i].real /= n;

}

}

double a[MAX+5];

double b[MAX+5];

c\_number a\_y[MAX+5];

c\_number b\_y[MAX+5];

int main()

{

int n1,n2,n=1;

scanf("%d%d",&n1,&n2); *///输入两个多项式的次数*

int cnt = 0;

**while**(n<=n1+n2)

{

n<<=1;

cnt++;

}

getInv(cnt);

memset(a\_y,0,**sizeof**(a\_y));

memset(b\_y,0,**sizeof**(b\_y));

**for**(int i=0;i<=n1;i++)

scanf("%lf",&a\_y[i].real);

**for**(int i=0;i<=n2;i++)

scanf("%lf",&b\_y[i].real);

fft(a\_y,n,1);

fft(b\_y,n,1);

**for**(int i=0;i<n;i++) a\_y[i] = multi(a\_y[i],b\_y[i]);

fft(a\_y,n,-1);

**for**(int i=0;i<=(n1+n2);i++)

{

printf("%lld ",(long long)(a\_y[i].real+0.1));

}

}