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

|  |
| --- |
| dengtesla  2018/3/26 |

目录

[1 数论 1](#_Toc512455032)

[1.1 线性筛打质数表，质因数分解 1](#_Toc512455033)

[1.2 莫比乌斯函数，欧拉函数 2](#_Toc512455034)

[1.3 Miller-Rabin素性测试 4](#_Toc512455035)

[1.4 快速幂，矩阵快速幂 5](#_Toc512455036)

[1.5 卢卡斯、大组合数取模 6](#_Toc512455037)

[1.6 中国剩余定理（不互质情况） 7](#_Toc512455038)

[2 字符串 11](#_Toc512455039)

[2.1 KMP字符串匹配 11](#_Toc512455040)

[3 数据结构 13](#_Toc512455041)

[3.1 并查集 13](#_Toc512455042)

[3.2 线段树 14](#_Toc512455043)

[3.3 zkw线段树 15](#_Toc512455044)

[3.4 主席树 15](#_Toc512455045)

[3.5 树状数组 15](#_Toc512455046)

[4 计算几何 16](#_Toc512455047)

[4.1 基础模板 16](#_Toc512455048)

# 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

int poww(int a,int b){

int ans=1,base=a;

**while**(b!=0){

**if**(b&1!=0)

ans\*=base;

base\*=base;

b>>=1;

}

**return** ans;

}

///矩阵快速幂

**const** int N=10;

int tmp[N][N];

void multi(int a[][N],int b[][N],int n)

{

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

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

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

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

tmp[i][j]+=a[i][k]\*b[k][j];

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

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

a[i][j]=tmp[i][j];

}

int res[N][N];

void Pow(int a[][N],int n)

{

memset(res,0,**sizeof** res);*//n是幂，N是矩阵大小*

**for**(int i=0;i<n;i++) res[i][i]=1;

**while**(n)

{

**if**(n&1)

multi(res,a,N);*//res=res\*a;复制直接在multi里面实现了；*

multi(a,a,N);*//a=a\*a*

n>>=1;

}

}

## 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]);

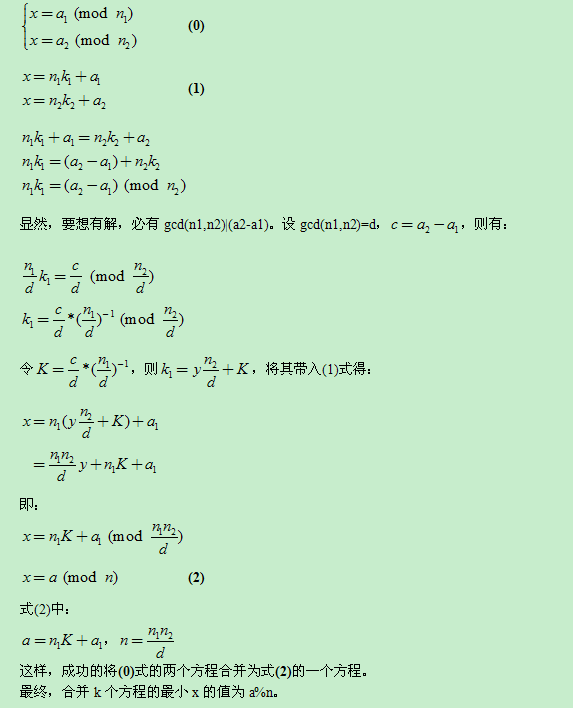
printf("Case %d: %I64d**\n**",cas,ex\_crt(m,r,n));

}

**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;

}

# 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;

}

# 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++;

}

}

## 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

# 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);

}