ACM-ICPC

D-Tesla 模板

2018

dengtesla

2018/3/26

目录

1 数论

1.1 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;
    }
}
```

1.2 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);
        }
    }
}
```

1.3 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;
}
```

1.4 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;
    }
}
```

1.5 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)
```

1.6 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：

$$\begin{cases} x = a_1 \pmod{n_1} \\ x = a_2 \pmod{n_2} \end{cases} \quad (0)$$

$$\begin{aligned} x &= n_1 k_1 + a_1 \\ x &= n_2 k_2 + a_2 \end{aligned} \quad (1)$$

$$\begin{aligned} n_1 k_1 + a_1 &= n_2 k_2 + a_2 \\ n_1 k_1 &= (a_2 - a_1) + n_2 k_2 \\ n_1 k_1 &= (a_2 - a_1) \pmod{n_2} \end{aligned}$$

显然，要想有解，必有 gcd(n1,n2)|(a2-a1)。设 gcd(n1,n2)=d，$c = a_2 - a_1$，则有：

$$\frac{n_1}{d} k_1 = \frac{c}{d} \pmod{\frac{n_2}{d}}$$

$$k_1 = \frac{c}{d} * (\frac{n_1}{d})^{-1} \pmod{\frac{n_2}{d}}$$

令 $K = \frac{c}{d} * (\frac{n_1}{d})^{-1}$，则 $k_1 = y\frac{n_2}{d} + K$，将其带入(1)式得：

$$\begin{aligned} x &= n_1 (y\frac{n_2}{d} + K) + a_1 \\ &= \frac{n_1 n_2}{d} y + n_1 K + a_1 \end{aligned}$$

即：

$$x = n_1 K + a_1 \pmod{\frac{n_1 n_2}{d}}$$

$$x = a \pmod{n} \quad (2)$$

式(2)中：

$$a = n_1 K + a_1, \quad n = \frac{n_1 n_2}{d}$$

这样，成功的将(0)式的两个方程合并为式(2)的一个方程。

最终，合并 k 个方程的最小 x 的值为 a%n。

```
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;
}
```

2 字符串

2.1 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;
}
```

3 数据结构

3.1 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++;
  }
}
```

3.2 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(1,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);
}
```

3.3 zkw 线段树

3.4 主席树

3.5 树状数组