目录

1 基本操作

1.1 离散化

1.1.1 一维线离散化

```
struct LINE{
    int l,r;
}line[SIZE*16];
int lisan[SIZE*16];
void Discrete(int N){//bl存的是编号为[1,N]的线,在每个线段的尾部插入一个断点
    int lisantot=0;
    for(int i=1;i<=N;i++){
        lisan[lisantot++]=line[i].l;
        lisan[lisantot++]=line[i].r;
        lisan[lisantot++]=line[i].r+1;
    }
    sort(lisan,lisan+lisantot);
    int lisanlen=unique(lisan,lisan+lisantot)-lisan;
    for(int i=1;i<=N;i++){
        line[i].l=lower_bound(lisan,lisan+lisanlen,line[i].l)-lisan+1;
        line[i].r=lower_bound(lisan,lisan+lisanlen,line[i].r)-lisan+1;
    }
}
```

1.1.2 二维点离散化

```
#define ll long long
const int SIZE=1e5;
struct point{
    ll x,y;
}p[SIZE];
bool cmp_x(point a,point b){return a.x < b.x;}
bool cmp_y(point a,point b){return a.y < b.y;}
void Discrete(int n){//n个点,下标[1,n]
    sort(p+1,p+n+1,cmp_x);
    int last=p[1].x,num=1;
    p[1].x = num = 1;
    for(int i=2;i<=n;i++){
        if(p[i].x == last)p[i].x=num;
        else{
            last = p[i].x;
            p[i].x=++num;
        }
```

```
    }
    sort(p+1,p+n+1,cmp_y);
    last=p[1].y,num=1;
    p[1].y=num=1;
    for(int i=2;i<=n;i++){
        if(p[i].y == last)p[i].y=num;
        else{
            last=p[i].y;
            p[i].y=++num;
        }
    }
}
```

1.2 矩阵快速幂

```
//矩阵快速幂
class mat{
public:
    int n,m;
    ll v[maxn][maxn];
    mat(int n,int m):n(n),m(m){}
    void init()
    {
        memset(v,0,sizeof(v));
    }
    void init1()
    {
        for(int i=0;i<maxn;i++)
            for(int j=0;j<maxn;j++)
                v[i][j]=(i==j); //单位矩阵
    }
    mat operator* (const mat B) const//矩阵乘法 A(n,k)*B(k,m)=C(n,m);
    {
        mat C(n,B.m);
        C.init();
        for(int i=0;i<n;i++)
        for(int j=0;j<B.m;j++)
        for(int k=0;k<m;k++)
            C.v[i][j]=(C.v[i][j]+v[i][k]*B.v[k][j])%Mod;//Mod
        return C;
    }
    mat operator ^ (int t)//矩阵快速幂 n=m时可用
    {
```

```
        mat ans(n,n),now(n,n);
        ans.init1();
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                now.v[i][j]=v[i][j];
        while(t>0)
        {
            if(t&1) ans=ans*now;
            now=now*now;
            t>>=1;
        }
        return ans;
    }
};
```

1.3 二进制枚举

```
for(int i=0;i<(1<<n);i++)//n个物品取或不取
{

    for(int j=0;j<n;j++)
    {
        if( i & (1<<j) )//取
        {

        }
        else//不取
        {

        }
    }
 }
```

1.4 状态压缩

```
//判断一个数字x二进制下第i位是不是等于1,反之为0
if(((1<<(i-1))&x)>0)

//将一个数字x二进制下第i位更改成1
x=x|(1<<(i-1))

//将一个数字x二进制下第i位更改成0
```

```
x=x^i(1<<(i-1));

//把一个数字二进制下最靠右的第一个1去掉
x=x&(x-1)
```

1.5 bitset

```
/*
C++的 bitset 在 bitset 头文件中，它是一种类似数组的结构，
它的每一个元素只能是 0 或 1，每个元素仅用 1 bit空间。
*/
//————————————构造方法————————————
bitset<4> bitset1;    //无参构造，长度为 4，默认每一位为 0
bitset<8> bitset2(12);    //长度为 8，二进制保存，前面用 0 补充
string s = "100101";
bitset<10> bitset3(s);    //长度为10，前面用 0 补充
char s2[] = "10101";
bitset<13> bitset4(s2);    //长度为13，前面用 0 补充
cout << bitset1 << endl;    //0000
cout << bitset2 << endl;    //00001100
cout << bitset3 << endl;    //0000100101
cout << bitset4 << endl;    //0000000010101
//————————————可用函数————————————
bitset<8> foo ("10011011");

cout << foo.count() << endl;    //5    count函数用来求bitset中1的位数
cout << foo.size() << endl;    //8    size函数用来求bitset的大小

cout << foo.test(0) << endl;    //true    test函数用来查下标处的元素是 0 还是 1
cout << foo.test(2) << endl;    //false

cout << foo.any() << endl;    //true    any函数检查bitset中是否有 1
cout << foo.none() << endl;    //false    none函数检查bitset中是否没有 1
cout << foo.all() << endl;    //false    all函数检查bitset中是全部为 1


bitset<8> foo ("10011011");

cout << foo.flip(2) <<
   endl;    //10011111    flip函数传参数时，用于将参数位取反
cout << foo.flip() << endl;
   //01100000    flip函数不指定参数时，将bitset每一位全部取反

```

```
cout << foo.set() <<
   endl;        //11111111    set函数不指定参数时，将bitset的每一位全部置为 1
cout << foo.set(3,0) <<
   endl;    //11110111    set函数指定两位参数时，将第一参数位的元素置为第二参数的值
cout << foo.set(3) << endl;
   //11111111    set函数只有一个参数时，将参数下标处置为 1

cout << foo.reset(4) <<
   endl;    //11101111    reset函数传一个参数时将参数下标处置为 0
cout << foo.reset() << endl;
   //00000000    reset函数不传参数时将bitset的每一位全部置为 0



bitset<8> foo ("10011011");

string s = foo.to_string();    //将bitset转换成string类型
unsigned long a = foo.to_ulong();    //将bitset转换成unsigned long类型
unsigned long long b = foo.to_ullong();    //将bitset转换成unsigned long
   long类型

cout << s << endl;    //10011011
cout << a << endl;    //155
cout << b << endl;    //155
```

2 数论

2.1 快速幂

```
ll powmod(ll a,ll b,ll mod){
    ll P=1;
    a=a%mod;
    while(b){
        if(b&1) P=(P*(a%mod))%mod;
        a=(a*a)%mod;
        b>>=1;
    }
    return P;
}
```

2.2 快速乘（×）龟速乘（√）

```cpp
ll qmul(ll x, ll y, ll mod) { // 乘法防止溢出，
  //如果不爆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 ret = 0;
    while(y) {
        if(y & 1)
            ret = (ret + x) % mod;
        x = x * 2 % mod;
        y >>= 1;
    }
    return ret;
    */
}
```

2.3 中国剩余定理

2.3.1 模数不互质（扩展中国剩余）

```cpp
typedef __int128 ll;
void exgcd(ll a, ll b, ll &g, ll &x, ll &y)
{
    if (b == 0)
    {
        g = a;
        x = 1;
        y = 0;
        return;
    }
    exgcd(b, a % b, g, y, x);
    y -= (a / b) * x;
}
bool china_flag = false;
ll a1, a2, m1, m2;
ll abs(ll x) { return x > 0 ? x : -x; }
void china()
{
    ll d = a2 - a1;
    ll g, x, y;
    exgcd(m1, m2, g, x, y);
```

```
    if (d % g == 0)
    {
        x = ((x * d / g) % (m2 / g) + (m2 / g)) % (m2 / g);
        a1 = x * m1 + a1;
        m1 = (m1 * m2) / g;
    }
    else
        china_flag = true;
}
int n;
long long as[111];
long long ms[111];//mod m =a;
ll realchina()
{
    a1 = as[0];
    m1 = ms[0];
    for (ll i = 1; i < n; i++)
    {
        a2 = as[i];//输入太多可依次输入
        m2 = ms[i];
        china();
        if (china_flag)
            return -1;
    }
    return a1;
}
```

2.3.2 模数互质

```
void exgcd(ll a,ll b,ll &g,ll &x,ll &y)
{
    if (b == 0) {
        g = a;
        x = 1;
        y = 0;
        return;
    }
    exgcd(b,a%b,g,y,x);
    y-=(a/b)*x;
}
ll n;
ll as[100005];//n
ll ms[100005];
```

```
ll china()
{
    ll ans=0,lcm=1,x,y,g;
    for(int i=0;i<n;i++) lcm*=as[i];
    for(int i=0;i<n;i++)
    {
        ll tp=lcm/as[i];
        exgcd(tp,as[i],g,x,y);
        x=(x%as[i]+as[i])%as[i];
        ans=(ans+tp*x*ms[i])%lcm;
    }
    return (ans%lcm+lcm)%lcm;
}
```

2.4 博弈论

2.4.1 Bash 博弈

```
n%(m+1)==0 ， 先手输
```

2.4.2 威佐夫博弈

```
(a1-a2)*((sqrt(5)+1)/2)==a2，先手输//a1>a2
第k个奇异局势为（（1 + sqrt(5) ) * k / 2，( 3 + sqrt(5) ) * k / 2  ）
```

2.4.3 威佐夫博弈扩展

```
按照上面在棋盘上移动棋子的方式，每次向左下移动可以看作是向左移动dx,向下移动dy，
    要求|dx-dy|<1。
那么如果将这个要求扩展到|dx-dy|<d呢？我们同样可以得出第k个奇异局势(x,y)，
x为前0～k-1个奇异局势中没有出现过的最小自然数，
y=x+d*k。根据Betty定理同样能得出第k个奇异局势为
( (2 - d + sqrt(d ^ 2 + 4))*k/2,  (2 + d + sqrt(d ^ 2 + 4))*k/2 )
```

2.4.4 Nim 博弈

```
算出各个sg值，全体异或=0，先手输
```

2.4.5 非常规

SG值的计算方法：（*重点*）

1.可选步数为1~m的连续整数，直接取模即可，SG(x) = x % (m+1);

2.可选步数为任意步，SG(x) = x;

3.可选步数为一系列不连续的数，用模板计算。

```
//f[]：可以取走的石子个数
//sg[]:0~n的SG函数值
//hash[]:mex{}
int f[N],sg[N],hash[N];
void getSG(int n)
{
    int i,j;
    memset(sg,0,sizeof(sg));
    for(i=1;i<=n;i++)
    {
        memset(hash,0,sizeof(hash));
        for(j=1;f[j]<=i;j++)
            hash[sg[i-f[j]]]=1;
        for(j=0;j<=n;j++)    //求mes{}中未出现的最小的非负整数
        {
            if(hash[j]==0)
            {
                sg[i]=j;
                break;
            }
        }
    }
}
```

2.5 素数

2.5.1 素数检测（米勒罗宾）

```
// 18位素数：154590409516822759
// 19位素数：2305843009213693951（梅森素数）
// 19位素数：4384959246869544497
LL prime[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 ret = 0;
    while(y) {
        if(y & 1)
            ret = (ret + x) % mod;
        x = x * 2 % mod;
        y >>= 1;
    }
    return ret;
    */
}
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 == prime[i]) return 1;
        LL t = s, m = qpow(prime[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;
}
```

2.5.2 素数线性筛

```
bool notp[N];
int prime[N],pnum;//prime里存素数
```

```
void sieve()
{
    memset(notp,0,sizeof(notp));
    notp[0]=notp[1]=1;
    pnum=0;
    for(int i=2;i<N;i++)
    {
        if(!(notp[i])) prime[++pnum]=i;
        for(int j=1;j<=pnum&&prime[j]*i<N;j++)
        {
            notp[prime[j]*i]=1;
            if(i%prime[j]==0) break;
        }
    }
}
```

2.5.3 区间筛

```
//找L~R之间的质数
ll L,R;
int k;
bool notp[N];
int prime[N],pnum;
void sieve()
{
    memset(notp,0,sizeof(notp));
    notp[0]=notp[1]=1;
    pnum=0;
    for(int i=2;i<sqrt(R);i++)
    {
        if(!(notp[i])) prime[++pnum]=i;
        for(int j=1;j<=pnum&&prime[j]*i<sqrt(R);j++)
        {
            notp[prime[j]*i]=1;
            if(i%prime[j]==0) break;
        }
    }
}
bool v[1000010];//v中存L开始的数是否为素数
void interval_sieve()
{
    memset(v,1,sizeof(v));
    if(L==1) v[0]=0;
```

```
    for(int i=1;i<=pnum;i++)
    {
        for(int j=(L-1)/prime[i]+1;j<=R/prime[i];j++)
        {
            if(j>1) v[prime[i]*j-L]=0;
        }//用prime中的素数筛L~R的素数
    }
}
```

2.6 合数

2.6.1 合数分解

```
//素数筛
pair <int ,int> d[N];int dnum;//分解的质数
bool is_prime(ll x)
{//大素数检测，可用米勒罗宾代替
    if(x==1)
        return false;
    if(x==2||x==3)
        return true;
    if(x%6!=1&&x%6!=5)
        return false;
    int s=sqrt(x);
    for(int i=5;i<=s;i+=6)
        if(x%i==0||x%(i+2)==0)
            return false;
    return true;
}
void prime_div(ll n)
{
    dnum=0;
    if(n==0) return;
    int num;
    for(int i=1;i<=pnum;i++)
    {
        if(n%prime[i]==0)
        {
            num=0;
            while(n%prime[i]==0)
            {
                num++;
                n/=prime[i];
            }
```

```
            d[++dnum]=make_pair(prime[i],num);
        }
        if(is_prime(n))
        {
            d[++dnum]=make_pair(n,1);
            break;
        }
    }
}
int main()
{
    ll n;
    sieve();
    while(~scanf("%lld",&n))
    {
        prime_div(n);
        for(int i=1;i<=dnum;i++)
        {
            cout<<d[i].first<<" "<<d[i].second<<endl;
        }
    }
}
```

2.6.2 区间合数分解

```
/*
区间合数分解（d(x^k) ）//有几个因数

若 x 分解成质因子乘积的形式为 x = p1^a1 * p2^a2 * ... * pn^an，那么 d(x) =
   (a1 + 1) * (a2 + 1) * ... * (an + 1) .显然 d(x^k) = (a1 * k + 1) * (a2 * k
   + 1) * ... * (an * k + 1) .
*/

//先来发区间素数筛 2~sqrt(R)
ll num[1000010];L~R
ll d[1000010];//L~R的分解数
int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%lld%lld%d",&L,&R,&k);
```

```
        sieve();
        for(int i=0;i<=R-L;i++) d[i]=1,num[i]=i+L;
        for(int i=1;i<=pnum;i++)
        {
            ll st=(L+prime[i]-1)/prime[i]*prime[i];//st
               :L开始的第一个能整除prime[i]的数
            for(ll j=st;j<=R;j+=prime[i])
            {
                int cnt=0;
                while(num[j-L]%prime[i]==0)
                {
                    num[j-L]/=prime[i];
                    cnt++;
                }
                d[j-L]=d[j-L]*(k*cnt%mod+1)%mod;//cn*k+1
            }
        }
        ll ans=0;
        for(int i=0;i<=R-L;i++)
        {
            if(num[i]!=1) d[i]=d[i]*(k+1)%mod;//未分解完，有大素数
            ans=(ans+d[i])%mod;
        }
        printf("%lld\n",ans);
    }
}
```

2.6.3 阶乘的因数个数

```
阶乘 N! 中包含质因子 x 的个数为 N/x + N/x^2 + N/x^3 +…
```

2.6.4 组合数的质因数个数

```
// C(m,n)=n! / ((n – m)! * m!);
//素数筛
//分别计算分子和分母中不同质因子的个数，相减大于 0 则对答案贡献 1
int ans=0;
    for(int i=1;i<=pnum;i++)
    {
        int tnum=0;
        int ch=prime[i];
        for(;ch<=n||ch<=m||ch<=n-m;ch*=prime[i])
        {
```

```
            if(n>=ch) tnum+=n/ch;
            if(m>=ch) tnum-=m/ch;
            if(n-m>=ch) tnum-=(n-m)/ch;
        }
//      cout<<tnum<<endl;
        if(tnum>0) ans++;
    }
    printf("%d\n",ans);
```

2.6.5 约数之和

```
//N 的约数之和 =(1+p1+p1^2 +……+p1^c1)*(1 +p2 + p2^2
   +……+p2^c2)*…*(1+pn+pn^2 +……+pn^cn)
ll psum(ll p,int c)//psum 可求(1+pn+pn^2 +……+pn^cn)
{
    if(c==0) return 1;
    if(c&1)
    {
        return (1 + powmod(p , (c + 1) / 2) ) % mod * psum( p , (c - 1) / 2)
           % mod;
    }
    else
    {
        return ((1 + powmod(p , c / 2)) % mod * psum(p , c / 2 - 1) % mod +
           powmod(p , c)) % mod;
    }
}
```

2.7 求逆元

2.7.1 模数为质数

```
inva=powmod(a,mod-2,mod);//快速幂
```

2.7.2 模数不为质数

```
void exgcd(ll a,ll b,ll &g,ll &x,ll &y){
    if(!b)
    {
        g=a;
        x=1;
        y=0;
```

```
        return ;
    }
    exgcd(b,a%b,g,y,x);
    y-=(a/b)*x;
}
//a=1( %mod )
//ax+mod *y=1;
//x为逆元
int main()
{
    ll a,mod,x,y,g;
    cin>>a;//a为要求的逆元
    cin>>mod;
    exgcd(a,mod,g,x,y);
    //g=1;
    x=(x%mod+mod)%mod;
    cout<<x;//x为逆元
}
```

2.7.3 线性逆元

```
ll inv[N];
inv[1]=1;
for(int i=2;i<mod;i++)
  inv[i]=(mod-mod/i)*inv[mod%i]%mod;
```

2.7.4 阶乘逆元

```
//先求n! 逆元
for(int i=n-1;i>=1;i--)
{
    inv[i]=inv[i+1]*(i+1)%mod;
}
```

2.8 GCD

2.8.1 一些性质

```
//GCD(Fib(n),Fib(m)) = GCD(Fib(n?m),Fib(m))

//GCD(m^a,n^a) = GCD(m,n)^a

```

```
//gcd(x^a-1,x^b-1)=x^gcd(a,b)-1

//GCD(a^m-b^m,a^n-b^n) = a^GCD(m,n) - b^GCD(m,n) (a,b 互质)

//gcd（b*t+a，b）=gcd（a，b）
//所以和b有关的gcd以phi(b)为循环节循环
```

2.8.2 区间 gcd

```
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}
int v[N], l[N], a[N], len[N];
map<ll, ll> M;
struct point
{
    int l, v;
} p[N][55];

scanf("%d", &n);
memset(v, 0, sizeof(v));
memset(len, 0, sizeof(len));
memset(l, 0, sizeof(l));
M.clear();
for (int i = 1; i <= n; i++)
    scanf("%d", &a[i]);
for (int i = 1, j; i <= n; i++) //枚举右端点
{
    for (v[i] = a[i], j = l[i] = i; j; j = l[j] - 1)
    {
        v[j] = gcd(v[j], a[i]);
        while (l[j] > 1 && gcd(a[i], v[l[j] - 1]) == gcd(a[i], v[j]))
        l[j] = l[l[j] - 1];//和自己边界l[j]-1相同,更新l[j]
        p[i][len[i]].l = l[j];
        p[i][len[i]].v = v[j];
        len[i]++;//边界种类++;
        M[v[j]] += (j - l[j] + 1);//gcd为v[j]的数目++
    }
}
int l, r, ans;
scanf("%d%d", &l, &r);
for (int i = 0; i < len[r]; i++)
```

```
{
    if (l >= p[r][i].l)
    {
        ans = p[r][i].v;
        break;
    }
}
printf("%d %lld\n", ans, M[ans]);
```

2.8.3 gcd+lcm

```
//先求n! 逆元
for(int i=n-1;i>=1;i--)
{
    inv[i]=inv[i+1]*(i+1)%mod;ll gcd(ll a,ll b){
    return b?gcd(b,a%b):a;
}//最大公约       //建议直接__gcd()
ll lcm(ll a,ll b){
    return a*b/gcd(a,b);//可能会爆，快速乘
}//最大公倍
```

2.9 欧拉函数

2.9.1 单个欧拉值 (sqrt(n))

```
ll phi(ll n)
{
    ll ans = n;
    for (ll i = 2; i * i <= n; i++){
        if (n % i == 0){
            ans -= ans / i;          //这一步就是对应欧拉函数的通式
            while (n % i == 0)  //为了保证完全消除我们刚才得到的那个i因子。
                n /= i;             //确保我们下一个得到的i是n的素因子。
        }
    }
    if (n > 1)               //如果还有素因子没有除
        ans -= ans / n;
    return ans;
}
```

2.9.2 线性筛欧拉

```cpp
bool notp[N];
int prime[N], pnum; //prime里存素数
int phi[N];
void sieve()
{
    memset(notp, 0, sizeof(notp));
    phi[1] = notp[0] = notp[1] = 1;
    pnum = 0;
    for (int i = 2; i < N; i++)
    {
        if (!(notp[i]))
            prime[++pnum] = i, phi[i] = i - 1;
        for (int j = 1; j <= pnum && prime[j] * i < N; j++)
        {
            notp[prime[j] * i] = 1;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] ? 1);
        }
    }
}
```

2.9.3 欧拉妙用

```cpp
//对欧拉函数求前缀和可得<=n的质数对

//质数N的欧拉函数是N-1 ， N^k 的欧拉函数是（N-1）*(N)^(k-1);

//欧拉定理 a,n互质时，a^(phi[n])=1（mod n）

//当n为素数时 a^(n-1)=1 mod n 此公式即 费马小定理
//=>a^(n-2) 为在mod数为n下的逆元

//phi[x] 一定是偶数 x>2

//小于或等于n的数中，与n互质的数的总和为：phi[x] * x / 2  (n>1)

//欧拉函数是积性函数□□若m,n互质，phi(mn)=phi(n)*phi(m)，

//当n为奇数的时候，phi(2n)=phi(n)，由上可得
```

```

//因为2必定和所有的奇数都是互质的，所以而phi(2)=1。所以得出这个结果

//若正整数 a,p 互质，则对于正整数 b,
//a^b mod p = a^(b mod phi(p)) mod p

//当正整数 a,p 不一定互质时
//a^b mod p =a^(b mod phi(p)+phi(p)) mod p (phi(p) ≤b)
```

2.9.4 欧拉降幂

```
//当带模求幂过程中指数非常大时可以通过欧拉定理降幂
a^b mod p = a^(b mod phi(p)+phi(p)) mod p  //(phi(p) ≤b)
```

2.10 扩展欧几里得

```
void exgcd(ll a,ll b,ll &g,ll &x,ll &y)//ax+by=gcd(a,b)=g
{
    if (b == 0) {
        g = a;
        x = 1;
        y = 0;
        return;
    }
    exgcd(b,a%b,g,y,x);
    y-=(a/b)*x;
}
//ax + by = c; g;

//c%g==0时有解

//x=x*c/g,y=y*c/g;
//得到特解 :(x,y)

//dx=b/g,dy=-a/g;
//得到通解 :(x+dx*n,y+dy*n);

//得到x最小正整数解 :(x%dx+dx)%dx;
```

2.11 同余不等式

```
// L<=(D × x mod M )<= R
```

```
//L<=Dx-My<=R
//y==0 时直接求解
//y !=0 时有 : Dx-R<= My <= Dx-L ,y 越小则 x 越小
//不等式同模 D
//得 (-R)%D <= (M%D)*y%D <= (-L)%D
// 递归求解
ll find(ll m, ll d, ll l, ll r)
{
    if (!d || l > r)
        return -1;
    if (r / d * d >= l)
        return (l - 1) / d + 1;
    ll x = find(d, m % d, ((-r) % d + d) % d, ((-l) % d + d) % d);
    if (x == -1)
        return -1;
    return (l - 1 + x * m) / d + 1;
}
int main()
{
    int n;
    ll x, y, m, d, l, r;
    scanf("%d", &n);
    while (n--)
    {
        scanf("%lld%lld%lld%lld", &m, &d, &l, &r);
        r = min(m - 1, r);
        printf("%lld\n", find(m, d, l, r));
    }
}
```

2.12 高次同余方程

```
//给定y,z,p，计算满足Y^x = Z(mod P)的最小非负整数
ll bsgs(ll a, ll b, ll p)
{
    map<int, int> H;
    H.clear();
    b %= p;
    int t = (int)sqrt(p) + 1;
    for (int i = 0; i < t; i++)
    {
        int val = (ll)b * powmod(a, i, p) % p;
        H[val] = i;
```

```
    }
    a = powmod(a, t, p);
    if (!a)
        return b ? -1 : 1;
    for (int i = 0; i <= t; i++)
    {
        int val = powmod(a, i, p);
        int j = H.find(val) == H.end() ? -1 : H[val];
        if (j >= 0 && i * t >= j)
            return i * t - j;
    }
    return -1;
}
```

2.13 高斯消元

2.13.1 一般模板（double）

```
#include <bits/stdc++.h>
using namespace std;
#define N 105
double a[N][N];
int n;
void gauss()
{
    for (int i = 1; i <= n; i++) //枚举列（项）
    {
        int maxn = i;
        for (int j = i + 1; j <= n; j++) //选出该列最大系数
        {
            if (fabs(a[j][i]) > fabs(a[max][i]))
            {
                maxn = j;
            }
        }
        for (int j = 1; j <= n + 1; j++) //交换
        {
            swap(a[i][j], a[maxn][j]);
        }
        if (!a[i][i]) //最大值等于0则说明该列都为0，肯定无解
        {
            printf("No Solution\n");
            return;
        }
```

```
        for (int j = 1; j <= n; j++) //每一项都减去一个数（就是小学加减消元）
        {
            if (j != i)
            {
                double temp = a[j][i] / a[i][i];
                for (int k = i + 1; k <= n + 1; k++)
                {
                    a[j][k] -= a[i][k] * temp;
                    //a[j][k]-=a[j][i]*a[i][k]/a[i][i];
                }
            }
        }
        for (int i = 1; i <= n; i++)
        {
            printf("%.2lf\n", a[i][n + 1] / a[i][i]);
        }
    }
    //上述操作结束后，矩阵会变成这样
    /*
    k1*a=e1
    k2*b=e2
    k3*c=e3
    k4*d=e4
    */
    //所以输出的结果要记得除以该项系数，消去常数
}
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n + 1; j++)
        {
            scanf("%lf", &a[i][j]);//tmd卡精度直接java,
        }
    }
    gauss();
    return 0;
}
```

2.13.2 高精度 java 模板

```
import java.util.*;
```

```java
import java.math.*;
import java.io.*;

public class Main {
    public static void main(String[] args) {
        new Task().main();
    }
}

class fraction {
    BigInteger a, b;

    fraction() {
        a = BigInteger.ZERO;
        b = BigInteger.ONE;
    }

    fraction(BigInteger a, BigInteger b) {
        this.a = a;
        this.b = b;
    }

    fraction add(fraction t) {
        BigInteger d, p, q;
        p = a.multiply(t.b);
        p = p.add(t.a.multiply(b));
        q = b.multiply(t.b);
        d = p.gcd(q);
        p = p.divide(d);
        q = q.divide(d);
        return new fraction(p, q);
    }

    fraction substract(fraction t) {
        BigInteger p, q, d;
        p = a.multiply(t.b);
        p = p.subtract(t.a.multiply(b));
        q = b.multiply(t.b);
        d = p.gcd(q);
        p = p.divide(d);
        q = q.divide(d);
        return new fraction(p, q);
    }

```

```
    fraction multiply(fraction t) {
        BigInteger p, q, d;
        p = a.multiply(t.a);
        q = b.multiply(t.b);
        d = p.gcd(q);
        p = p.divide(d);
        q = q.divide(d);
        return new fraction(p, q);
    }

    fraction divide(fraction t) {
        BigInteger p, q, d;
        p = a.multiply(t.b);
        q = b.multiply(t.a);
        d = p.gcd(q);
        p = p.divide(d);
        q = q.divide(d);
        return new fraction(p, q);
    }

    fraction abs() {
        return new fraction(a.abs(), b.abs());
    }

    int compareTo(fraction t) {
        t = this.substract(t);
        return t.a.compareTo(BigInteger.ZERO);
    }

    boolean zero() {
        return a.equals(BigInteger.ZERO);
    }
}

class Task {
    Scanner cin = new Scanner(System.in);
    PrintStream cout = System.out;
    fraction[][] f = new fraction[211][211];
    fraction[] x = new fraction[211];
    fraction tmp;
    BigInteger ONE = BigInteger.ONE;
    BigInteger ZERO = BigInteger.ZERO;
    fraction zero = new fraction(ZERO, ONE);
    fraction r, one = new fraction(ONE, ONE);
```

```
90
91      boolean gauss(int n) {
92          int i, j, k, row;
93          for (i = 1; i <= n; ++i) {
94              row = i;
95              for (j = i + 1; j <= n; ++j) {
96                  if (f[row][i].abs().compareTo(f[j][i].abs()) < 0) {
97                      row = j;
98                  }
99              }
100             if (f[row][i].compareTo(zero) == 0) {
101                 return false;
102             }
103             if (row != i) {
104                 for (k = i; k <= n; ++k) {
105                     tmp = f[row][k];
106                     f[row][k] = f[i][k];
107                     f[i][k] = tmp;
108                 }
109                 tmp = x[row];
110                 x[row] = x[i];
111                 x[i] = tmp;
112             }
113             for (j = i + 1; j <= n; ++j) {
114                 r = f[j][i].divide(f[i][i]);
115                 f[j][i] = zero;
116                 for (k = i + 1; k <= n; ++k) {
117                     f[j][k] = f[j][k].substract(r.multiply(f[i][k]));
118                 }
119                 x[j] = x[j].substract(r.multiply(x[i]));
120             }
121         }
122         for (i = n; i >= 1; --i) {
123             for (j = i - 1; j >= 1; --j) {
124                 r = f[j][i].divide(f[i][i]);
125                 f[j][i] = zero;
126                 x[j] = x[j].substract(r.multiply(x[i]));
127             }
128         }
129         for (i = 1; i <= n; ++i) {
130             x[i] = x[i].divide(f[i][i]);
131         }
132         return true;
133     }
```

```
134
135     void main() {
136         int n, i, j;
137         while (cin.hasNext()) {
138             n = cin.nextInt();
139             for (i = 1; i <= n; ++i) {
140                 for (j = 1; j <= n; ++j) {
141                     f[i][j] = new fraction(cin.nextBigInteger(), ONE);
142                 }
143                 x[i] = new fraction(cin.nextBigInteger(), ONE);
144             }
145             boolean ok = gauss(n);
146             if (!ok) {
147                 cout.println("No solution.");
148             } else {
149                 for (i = 1; i <= n; ++i) {
150                     if (x[i].b.compareTo(ZERO) < 0) {
151                         x[i].a = x[i].a.negate();//返回负值
152                         x[i].b = x[i].b.negate();
153                     }
154                 }
155                 for (i = 1; i <= n; ++i) {
156                     cout.print(x[i].a);
157                     if (x[i].b.compareTo(ONE) != 0) {
158                         cout.print("/" + x[i].b);
159                     }
160                     cout.println();
161                 }
162             }
163             cout.println();
164
165         }
166     }
167 }
```

2.13.3 高斯解同余方程

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long LL;
4 const int inf=0x3f3f3f3f;
5 const double pi= acos(-1.0);
6 const double esp=1e-6;
```

```
const int MAXN=310;
int aug[MAXN][MAXN];<span style="background-color: rgb(255, 255, 255);">
//增广矩阵行数为m,分别为0到m-1,列数为n+1,分别为0到n.</span>
int x[MAXN];//解集
int free_num;
int m,n;//m个方程，n个变元
int gcd(int a,int b) {
    int r;
    while(b!=0) {
        r=b;
        b=a%b;
        a=r;
    }
    return a;
}
int lcm(int a,int b) {
    return a/gcd(a,b)*b;
}
/*void Debug(void)
{
    puts("");
    int i,j;
    for(i=0;i<m;i++){
        for(j=0;j<n+1;j++){
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}*/
int trans(char *str) {
    if(strcmp(str,"MON")==0) return 1;
    else if(strcmp(str,"TUE")==0) return 2;
    else if(strcmp(str,"WED")==0) return 3;
    else if(strcmp(str,"THU")==0) return 4;
    else if(strcmp(str,"FRI")==0) return 5;
    else if(strcmp(str,"SAT")==0) return 6;
    else if(strcmp(str,"SUN")==0) return 7;
}
// 高斯消元法解方程组(Gauss-Jordan elimination).(-1表示无解，
//0表示唯一解，大于0表示无穷解，并返回自由变元的个数)
int Gauss() {
    int i,j;
    int row,col,max_r;// 当前这列绝对值最大的行;
```

```
    int LCM;
    int ta,tb;
    int tmp;
    for(row=0,col=0; row<m&&col<n; row++,col++) {
        // 枚举当前处理的行.
        // 找到该col列元素绝对值最大的那行与第row行交换.(为了在除法时减小误差)
        max_r=row;
        for(i=row+1; i<m; i++) {
            if(abs(aug[i][col])>abs(aug[max_r][col]))
                max_r=i;
        }
        if(max_r!=row) {
            // 与第row行交换
            for(j=row; j<n+1; j++)
                swap(aug[row][j],aug[max_r][j]);
        }
        if(aug[row][col]==0) {
            // 说明该col列第row行以下全是0了，则处理当前行的下一列.
            row--;
            continue;
        }
        for(i=row+1; i<m; i++) {
            // 枚举要删去的行.
            if(aug[i][col]!=0) {
                LCM=lcm(abs(aug[i][col]),abs(aug[row][col]));
                ta=LCM/abs(aug[i][col]);
                tb=LCM/abs(aug[row][col]);
                if(aug[i][col]*aug[row][col]<0) tb=-tb;//异号的情况是相加
                for(j=col; j<n+1; j++) {
                    aug[i][j]=(((aug[i][j]*ta-aug[row][j]*tb)%7+7)%7);
                }
            }
        }
    }
    //Debug();
    // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
    for(i=row; i<m; i++) {
        if(aug[i][col]!=0)  return -1;
    }
    // 2. 无穷解的情况: 在n * (n + 1)的增广阵中出现(0, 0, ...,
       0)这样的行，即说明没有形成严格的上三角阵.
    // 且出现的行数即为自由变元的个数.
    if(row<n){
        return n-row;
```

```
    }
     // 3. 唯一解的情况: 在n * (n + 1)的增广阵中形成严格的上三角阵.
    // 计算出Xn-1, Xn-2 ... X0.
    for(i=n-1; i>=0; i--) {
        tmp=aug[i][n];//等式右边的数
        for(j=i+1; j<n; j++) {
            if(aug[i][j]!=0)
               tmp-=aug[i][j]*x[j];//把已知的解带入，减去，只剩下，一个未知的解
            tmp=(tmp%7+7)%7;
        }
        while(tmp%aug[i][i]!=0)//外层每次循环都是为了求
           a[i][i],因为它是每个方程中唯一一个未知的变量（求该方程时）
            tmp+=7;//因为天数不确定，而aug[i][i]必须得为整数才可以，周期为7
        x[i]=(tmp/aug[i][i])%7;
    }
    return 0;
}
int main(void) {
    int nn,mm,i,j,k;
    int num;
    char Start[5],End[5];
    while(~scanf("%d %d",&nn,&mm)) {
        if(nn==0&&mm==0) break;
        n=m=0;
        memset(aug,0,sizeof(aug));
        for(i=0; i<mm; i++) {
            scanf("%d",&k);
            scanf("%s %s",Start,End);
            aug[i][nn]=((trans(End)-trans(Start)+1)%7+7)%7;
            for(j=1; j<=k; j++) {
                scanf("%d",&num);
                num--;
                aug[i][num]++;
                aug[i][num]%=7;//有重复的
            }
        }
        m=mm;
        n=nn;
        free_num = Gauss();
        if(free_num==0) {
            for(i=0; i<n; i++)
                if(x[i]<3)//根据题意，每个零件的加工时间在3-9天.
                    x[i]+=7;
            for(i=0; i<n-1; i++)
```

```
                printf("%d ",x[i]);
            printf("%d\n",x[i]);
        } else if(free_num==-1)
            puts("Inconsistent data.");
        else
            puts("Multiple solutions.");
    }
    return 0;
}
```

3 数学

3.1 蔡勒公式（1592 之后）

```
int WhatDay(int year,int month,int dday){//0是周日，1是周一，6是周六
    if(month<3){year-=1;month+=12;}
    int c=int(year/100),y=year-100*c;
    int w=int(c/4)-2*c+y+int(y/4)+(26*(month+1)/10)+dday-1;
    w=(w%7+7)%7;
    return w;
}
```

4 字符串

4.1 KMP

4.1.1 求 next 方法 1

模式串位置	1	2	3	4	5	6	7
char[]	a	b	a	b	a	b	c
nex[]	-1	0	0	1	2	3	4

模式串位置	0	1	2	3	4	5	6	7	8	9
char[]	a	b	c	a	b	d	a	b	c	d
nex[]	-1	0	0	0	1	2	0	1	2	3

```
const int SIZE=1e5+5;
//x 为要处理的字串， m 为长度 ，nextt 为next数组
void GetNext(string x,int m,int nextt[]){
    int i,j;
    j=nextt[0]=-1;
```

```
    i=0;
    while(i<m){
        while(-1!=j && x[i]!= x[j]){
            j=nextt[j];
        }
        nextt[++i]=++j;
    }
}
```

4.1.2 求 next 方法 2

模式串位置	0	1	2	3	4	5	6
char[]	a	b	a	b	a	b	c
nex[]	-1	0	-1	0	-1	0	4

模式串位置	0	1	2	3	4	5	6	7	8	9
char[]	a	b	c	a	b	d	a	b	c	d
nex[]	-1	0	0	-1	0	2	-1	0	0	3

```
const int SIZE=1e5+5;
void GetNext(string x,int m,int nextt[]){
    int i,j;
    j=nextt[0]=-1;
    i=0;
    while(i<m){
        while(-1!=j && x[i]!=x[j]){
            j=nextt[j];
        }
        if(x[++i]==x[++j]){
            nextt[i]=nextt[j];
        }
        else nextt[i]=j;
    }
}
```

4.2 字符串部分匹配

```
/*
    求a的头和b的屁股最大匹配情况
    a:riemann   b:marjorie res:rie 3
*/
```

```
const int SIZE=100050*2;
char a[SIZE],b[SIZE];
int kmp[SIZE]; //kmp为next数组
int main(){
    while(~scanf("%s%s",a,b)){
        int la=strlen(a);int lb=strlen(b);
        strcat(a,b);
        get_NEX(a);//求next方法1
        int len=la+lb;
        while(kmp[len]>la||kmp[len]>lb){
            len=kmp[len];
        }
        if(kmp[len]==0)
            printf("0\n");
        else{
            for(int i=0;i<kmp[len];i++)
                printf("%c",a[i]);
            printf(" %d\n",kmp[len]);
        }
    }
}
```

4.3 求字符串的前缀是否为周期串

```
/*
    求字符串的前缀是否为周期串，若是， 打印循环节的长度及循环次数；
*/
const int SIZE=1000500;
int n;char s[SIZE];
int nex[SIZE];
int main(){
    while(cin>>n&&n){//n为字符串长度
        cin>>s;
        Get_nex(s);//求next方法1
        for(int i=1;i<=n;i++){
            int tmp=i-nex[i];
            if(i%tmp==0 && i/tmp>1){
                printf("%d %d\n",i,i/tmp);
            }
        }
        printf("\n");
    }
}
```

4.4 字符串由多少相同的字串组成

```
/*
    给出一个字符串 问它最多由多少相同的字串组成
    ababab = 3
*/
const int SIZE=1000500;
int n;char s[SIZE];
int nex[SIZE];
int main(){
    int kase=1;
    while(cin>>s){
        if(s[0]=='.')break;
        Get_nex(s);//求next方法1
        int n=strlen(s);
        int tmp=n-nex[n];
        if(n%tmp==0 && nex[n]!=0)
            printf("%d\n",n/tmp);
        else printf("1\n");
    }
}
```

4.5 双倍回文

```
/*
    记字符串w的倒置为例如(abcd)^R=dcba，（abba)^R=abba。
    如果x能够写成w(w^R)w(w^R)的形式，则称它是一个“双倍回文"。
        换句话说，若要x是双倍回文，它的长度必须是4的倍数，
        而且x、x的前半部分、x的后半部分都要是回。
        例如：abbaabba是一个双倍回文，而abaaba不是， 因为它的长度不是4的倍数。
    x的子串是指在x中连续的一段字符所组成的字符。
        例如bc是abcd的子串，而ac不是。
    x的回文子串，就是指满足回文性质的x的子串。
    x的双倍回文子串，就是指满足双倍回文性质的x的子串。
    你的任务是，对于给定的字符串，计算它的最长双倍回文子串的长度。
*/
#define INF 0x3f3f3f3f
const int SIZE=500050;
int p[SIZE*2],q[SIZE*2];
char A[SIZE*2],str[SIZE*2];
int n;
void manacher(){
    int mx=0,id;
```

```
    for(int i=1;i<=n;i++){
        if(mx>=i)p[i]=min(mx-i,p[2*id-i]);
        else
            p[i]=0;
        for(;str[i+p[i]+1]==str[i-p[i]];p[i]++);
        if(p[i]+i>mx)id=i,mx=p[i]+i;
    }
}
bool cmp(int a,int b){return (a-p[a])<(b-p[b]);}
int main(){
    while(~scanf("%d%s",&n,str)){
        manacher();
        int maxx=-INF;
        for(int i=0;i<n;i++)q[i]=i;
        sort(q,q+n,cmp);
        int pos=0;
        set<int>st;
        for(int i=0;i<n;i++){
            while(pos<=n&&q[pos]-p[q[pos]]<=i){
                st.insert(q[pos]);
                pos++;
            }
            set<int>::iterator it=st.upper_bound(i+p[i]/2);
            if(it!=st.begin()){
                maxx=max(maxx,(*--it-i)*4);
            }
        }
        printf("%d\n",maxx);
    }
}
```

5 计算几何

5.1 叉积

5.1.1 点线式

```
struct point{
    double x,y;
};//点
struct line{
    point a,b;
};//线
```

```
double cross(point a,point b,point c)//叉积
{
    return (a.x - b.x) * (c.y - b.y) - (a.y - b.y) * (c.x-b.x);
}//小于0说明c在ab右侧
```

5.1.2 重载版

```
struct point
{
    double x, y;
    point(double x = 0, double y = 0) : x(x), y(y) {}
    point operator+(const point &t) const
    {
        return point(x + t.x, y + t.y);
    } //a + b
    point operator-(const point &t) const
    {
        return point(x - t.x, y - t.y);
    } //a - b
    double operator*(const point &t) const
    {
        return x * t.x + y * t.y;
    } //a * b
    double operator^(const point &t) const
    {
        return x * t.y - y * t.x;
    } //a X b
    double dis(const point &t)
    {
        return sqrt((x - t.x) * (x - t.x) +
                    (y - t.y) * (y - t.y));
    }
};
```

5.1.3 叉积算多边形面积

```
double cross(point a, point b, point c) //AC X AB
{
    return (c - a) ^ (b - a);
}
double getarea(int n)
{
    if (n < 3)
```

```
        return 0;
    int i;
    double res = 0;
    for (i = 2; i < n; i++)
    {
        res += fabs(cross(xp[s[0]], xp[s[i - 1]], xp[s[i]]) / 2.0);
    }
    return res;
}//小于0说明c在ab右侧
```

5.2 极角排序

5.2.1 叉积计算

```
//叉积计算极角（精度高，时间长）
struct point{
    double x, y;
    point(double x = 0, double y = 0) : x(x), y(y) {}
    point operator-(const point &t) const
    {
        return point(x - t.x, y - t.y);
    } //a - b
    double operator*(const point &t) const
    {
        return x * t.x + y * t.y;
    } //a * b
    double operator^(const point &t) const
    {
        return x * t.y - y * t.x;
    } //a X b
    double dis(const point &t)
    {
        return sqrt((x - t.x) * (x - t.x) + (y - t.y) * (y - t.y));
    }

} p[N], xp[N];
double compare(point a, point b, point c) //计算极角 ab × ac
{
    return (b - a) ^ (c - a);
}
bool cmp(point a, point b)
{
    double f = compare(p[pos], a, b);
    if (f == 0)
```

```
        return a.x - p[pos].x < b.x - p[pos].x;
    else if (f > 0)
        return true;
    else
        return false;
}

/*
//如果取的点不是边角要先按照象限排序
int Quadrant(point a) //象限排序，注意包含四个坐标轴
{
    if (a.x > 0 && a.y >= 0)
        return 1;
    if (a.x <= 0 && a.y > 0)
        return 2;
    if (a.x < 0 && a.y <= 0)
        return 3;
    if (a.x >= 0 && a.y < 0)
        return 4;
}

bool cmp2(point a, point b) //先象限后极角
{
    if (Quadrant(a) == Quadrant(b)) //返回值就是象限
        return cmp(a, b);
    else
        Quadrant(a) < Quadrant(b);
}
*/
```

5.2.2 atan2 函数计算

```
//atan2函数计算（时间短，精度低）
struct point{
    double x, y;
    double angle;
    bool operator<(const point &t)
    {
        return angle < t.angle;
    }
} p[N];
bool cmp(point a, point b)
{
```

```
    if (a.angle == b.angle)
        return a.x < b.x;
    else
    {
        return a.angle < b.angle;
    }
}
for (int i = 1; i <= n; i++)
{
    cin >> p[i].x >> p[i].y;
    p[i].angle = atan2(p[i].y, p[i].x);
}
sort(a + 1, a + 1 + n, cmp);
```

5.3 凸包

5.3.1 一般凸包

```
bool cmp(point a, point b)
{
    double f = (a - xp[0]) ^ (b - xp[0]); //与左下边界点的极角
    if (fabs(f) < eps)
        return a.x - xp[0].x < b.x - xp[0].x; //相等按距离排序
    else if (f > 0)
        return true;
    else
        return false;
}
void graham()
{
    for (int i = 0; i < pnum; i++) //pnum为内部点的数目
    {
        if (xp[i].y < xp[0].y || (xp[i].y == xp[0].y && xp[i].x < xp[0].x))
           //找到左下方点
            swap(xp[0], xp[i]);
    }
    sort(xp + 1, xp + pnum, cmp);
    if (pnum == 1)
        stack[0] = 0, top = 0; //建栈 找凸包
    else if (pnum == 2)
        stack[0] = 0, stack[1] = 1, top = 1;
    else //内树大于三个
    {
        stack[0] = 0, stack[1] = 1;
```

```
        top = 1;
        for (int i = 2; i < pnum; i++)
        {
            while (top > 0 && ((xp[stack[top]] - xp[stack[top - 1]]) ^ (xp[i]
               - xp[stack[top - 1]])) <= 0)
               //栈顶的点在当前点和前一个点的连线左侧
            {
                top--;
            }
            stack[++top] = i;
        }
    }
} ///栈里为从左下开始的组成凸包的点
```

5.3.2 最大空凸包

```
#define N 105
struct point{
    double x, y;
    point(double x = 0, double y = 0) : x(x), y(y) {}

    point operator-(const point &t) const
    {
        return point(x - t.x, y - t.y);
    } // b - a :   ab
    double operator*(const point &t) const
    {
        return x * t.x + y * t.y;
    } //a * b
    double operator^(const point &t) const
    {
        return x * t.y - y * t.x;
    } //a X b
    double dis(const point &t) const
    {
        return sqrt((x - t.x) * (x - t.x) + (y - t.y) * (y - t.y));
    } // |a-b|
} p[N], xp[N], 0;
int n, pnum;
double dp[N][N], ans;
bool cmp(const point &a, const point &b)
{
    double f = (a - 0) ^ (b - 0); //与左下边界点的极角
```

```
    if (f == 0)
        return O.dis(a) < O.dis(b); //相等按距离排序
    else if (f > 0)
        return true;
    else
        return false;
}

void solve()
{
    memset(dp, 0, sizeof(dp));
    sort(xp + 1, xp + pnum + 1, cmp);
    for (int i = 1; i <= pnum; i++)
    {
        int j = i - 1;
        while (j && !((xp[i] - O) ^ (xp[j] - O)))
            j--;
        bool bz = (j == i - 1); //bz==0 为当前为边，==1为当前为顶点
        while (j)
        {
            int k = j - 1;
            while (k && ((xp[i] - xp[k]) ^ (xp[j] - xp[k])) > 0)
                k--;
            double area = fabs((xp[i] - O) ^ (xp[j] - O)) / 2.0;
            if (k)
                area += dp[j][k];
            if (bz)
                dp[i][j] = area;
            ans = max(ans, area), j = k;
        }
        if (bz)
        {
            for (int j = 1; j < i; j++)
                dp[i][j] = max(dp[i][j], dp[i][j - 1]);
        }
    }
}

int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
```

```
        scanf("%d", &n);
        ans = 0;
        for (int i = 1; i <= n; i++)
        {
            scanf("%lf %lf", &p[i].x, &p[i].y);
        }
        for (int i = 1; i <= n; i++)
        {
            O = p[i], pnum = 0;
            for (int j = 1; j <= n; j++)
            {
                if (p[j].y > p[i].y || (p[j].y == p[i].y && p[j].x > p[i].x))
                {
                    xp[++pnum] = p[j];
                }
            }
            solve();
        }
        printf("%0.1f\n", ans);
    }
}
```

5.4 半平面交

5.4.1 叉积计算

```
#define N 110
#define eps 1e-8
#define inf 0x3f3f3f3f3f3f
struct node
{
    double u, v, w;
} k[N];
struct point{
    double x, y;
    point() {}
    point(double a, double b) { x = a, y = b; }
    point operator-(const point &t) const
    {
        return point(x - t.x, y - t.y);
    }
    double operator*(const point &t) const
    {
        return x * t.x + y * t.y;
```

```
    }
    double operator^(const point &t) const
    {
        return x * t.y - y * t.x;
    }
} p[N];
double dist(point a, point b)
{
    return sqrt((a - b) * (a - b));
}
int n, pnum;
point q[N];
int top = 0;
point cross_p(point x, point y, double a, double b, double c)
//求两个点x,y和该线的交点
{
    point ans;
    double a2 = y.y - x.y;
    double b2 = x.x - y.x;
    double c2 = x.y * y.x - y.y * x.x; //连接x,y的线段的a,b,c
    ans.y = (a * c2 - a2 * c) / (a2 * b - a * b2);
    ans.x = (b * c2 - b2 * c) / (b2 * a - b * a2);
    return ans;
}
void Hpi(double a, double b, double c)
{
    top = 0;
    for (int i = 1; i <= pnum; i++)
    {
        if (a * p[i].x + b * p[i].y + c <= eps) //如果上一个边界点满足，则记录
        {
            q[++top] = p[i];
        }
        else
        {
            if (a * p[i - 1].x + b * p[i - 1].y + c <= eps)
               //该节点不满足，但上一节点满足
            {
                q[++top] = cross_p(p[i - 1], p[i], a, b, c);
                   //加入两个节点的连线与该线的交点
            }
            if (a * p[i + 1].x + b * p[i + 1].y + c <= eps)
               //该节点不满足，但下一节点满足
            {
```

```
                q[++top] = cross_p(p[i], p[i + 1], a, b, c);
            }
        }
    }
    for (int i = 1; i <= top; i++)
        p[i] = q[i]; //将更新好的点集返回p
    p[top + 1] = p[1], p[0] = p[top];
    pnum = top;
}
bool solve(int id)
{
    p[1] = point(0, 0);
    p[2] = point(inf, 0);
    p[3] = point(inf, inf);
    p[4] = point(0, inf);
    p[0] = p[4], p[5] = p[1];
    pnum = 4; //初始边界为第一象限四个边界点，所以起始点为4个
    double a, b, c;
    for (int i = 1; i <= n; i++)
    {
        if (i != id && k[id].u <= k[i].u && k[id].v <= k[i].v && k[id].w <=
           k[i].w)
            return 0; //如果有人3项多比他强，肯定达不到
        if (i == id)
            continue;
        a = (k[i].u - k[id].u) / (k[id].u * k[i].u);
        b = (k[i].v - k[id].v) / (k[id].v * k[i].v);
        c = (k[i].w - k[id].w) / (k[id].w * k[i].w); //得出ax+by=c
        Hpi(a, b, c);
        if (pnum < 3)
            return 0; //如果最后交点数<3，即无法构成平面
    }
    return pnum >= 3;
}

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%lf%lf%lf", &k[i].u, &k[i].v, &k[i].w);
    for (int i = 1; i <= n; i++)
    {
        if (solve(i))
            printf("Yes\n");
```

```
        else
            printf("No\n");
    }
}
```

5.4.2 半平面交存点

```
//可求最大内接圆，最大半平面交面积
struct point{
    double x, y;
    point() {}
    point(double a, double b) { x = a, y = b; }
    point operator-(const point &t) const
    {
        return point(x - t.x, y - t.y);
    }
    double operator*(const point &t) const
    {
        return x * t.x + y * t.y;
    }
    double operator^(const point &t) const
    {
        return x * t.y - y * t.x;
    }
    double dis(const point &t)
    {
        return sqrt((x - t.x) * (x - t.x) + (y - t.y) * (y - t.y));
    }
} p[N], xp[N];
struct line
{
    point s, e;
    double k;
    line() {}
    line(point a, point b) { s = a, e = b, k = atan2(b.y - a.y, b.x - a.x); }
    point operator&(const line &b) const //求两直线交点
    {
        point res = s;
        double t = ((s - b.s) ^ (b.s - b.e)) / ((s - e) ^ (b.s - b.e));
        res.x += (e.x - s.x) * t;
        res.y += (e.y - s.y) * t;
        return res;
    }
```

```
};
double dist(point a, point b)
{
    return sqrt((a - b) * (a - b));
}
int n;
int ans_s; //相交点数
line L[N], q[N];
point ans[N]; //存点
bool cmp(line a, line b)
{
    if (fabs(a.k - b.k) <= eps)
        return ((a.s - b.s) ^ (b.e - b.s)) < 0; //平行取下
    return a.k < b.k;
}
void Hpi() //保证多边形逆时针建边
{
    int tot = 1;
    sort(L, L + n, cmp);
    for (int i = 1; i < n; i++)
    {
        if (fabs(L[i].k - L[i - 1].k) > eps)
        {
            L[tot++] = L[i];
        } //因为平行的时候，左边的线段先判断，所以可以去除无效判定
    }
    int l = 0, r = 1;
    q[0] = L[0];
    q[1] = L[1]; //模拟双端 ，每条线取左删右（顺时针建边取右删左）
    for (int i = 2; i < tot; i++)
    {
        if (fabs((q[r].e - q[r].s) ^ (q[r - 1].e - q[r - 1].s)) <= eps ||
            fabs((q[l].e - q[l].s) ^ (q[l + 1].e - q[l + 1].s)) <= eps)
            return;

                //半平面交不存在
        while (l < r && (((q[r] & q[r - 1]) - L[i].s) ^ (L[i].e - L[i].s)) >
           eps) //上一个交点在下一条线的右侧，不在半平面交上
            r--;
        while (l < r && (((q[l] & q[l + 1]) - L[i].s) ^ (L[i].e - L[i].s)) >
           eps) // 同上
            l++;
        q[++r] = L[i]; //加入新边
    }
```

```
    while (l < r && (((q[r] & q[r - 1]) - L[l].s) ^ (L[l].e - L[l].s)) > eps)
        r--;
    while (l < r && (((q[l] & q[l - 1]) - L[r].s) ^ (L[r].e - L[r].s)) > eps)
        l++;
    if (r <= l + 1)
        return; //如果只有2个或以下的点，构不成平面
    for (int i = l; i < r; i++)
    {
        ans[ans_s++] = q[i] & q[i + 1];
    } //ans里存交点
    ans[ans_s++] = q[l] & q[r];
}
```

6 数据结构

6.1 单调栈 + 单调队列

6.1.1 单调栈

```
int s[N], top=0;
for(int i=1;i<=n;i++)
{
    while(top&&a[s[top]]>a[i]) top--;
    if(s[top]==a[i]) s[top]=i;//去重复操作，视情况而定
    s[++top]=i;
}
```

6.1.2 单调队列

```
int l=0,r=1;
int q[N];
for(int i=1;i<=n;i++)
{
    while(l<=r&&a[q[r]]>a[i]) r--;
    q[++r]=i;
    while(l<=r&&q[l]<i-m+1) l++;//判断条件看情况
    ans[i]=a[q[l]];
}
```

6.2 树状数组

6.2.1 lowbit 之和

```
/*
    HDU5975 有n个集合，第 i 个集合的数是[i-lowbit(i)+1,i-1]+i
    第一种方式是集合[a,b]共有多少个数=lowbit之和？(sol1)
    第二种方式是数字x在几个集合里面？(sol2)
    long long型注意！
*/
#define ll long long
ll lowbit(ll x){
    return x&(-x);
}
ll sum(ll x){           //计算[1,x]中所有数lowbit(k)之和
    ll ans=0ll;
    for(ll p=1ll;p<=x;p<<=1){
        ans+=(x/p-x/(p<<1))*p;
    }
    return ans;
}
ll sol1(ll x,ll y){     //计算[x,y]中所有树的lowbit(k)之和
    return sum(y)-sum(x-1ll);
}
ll sol2(ll x,ll n){
    ll res=0ll;
    while(x<=n){
        res++;
        x+=lowbit(x);
    }
    return res;
}
```

6.2.2 区间加减 + 区间和查询

```
#define LL long long
const int SIZE=10005;
ll T1[SIZE],T2[SIZE],T[SIZE];//T[]用于存结点值，用过一次即仍
int lowbit(int k){
    return k&-k;
}
void update(ll x,ll,N,ll w){//把x位置之后所有数的值+w
    for(ll i=x;i<=N;i+=lowbit(i)){
        T1[i]+=w;T2[i]+=w*(x-1);
    }
}
void range_update(ll l,ll r,ll v){//在[l,r]上加v
```

```
    update(l,v);update(r+1,-v);
}
ll sum(ll x){
    ll ans=0;
    for(ll i=x;i>0;i-=lowbit(i)){
        ans+=x*T1[i]-T2[i];
    }
    return ans;
}
ll range_ask(ll l,ll r){//返回[l,r]的和
    return sum(r)-sum(l-1);
}
void init(int N){
    for(int i=1;i<=N;i++){
        update(i,T[i]-T[i-1]);
    }
}
```

6.2.3 统计前后顺序不同数字对个数（三维偏序问题）

```
#define ll long long
const int SIZE=200000+50;
ll arr[3][SIZE];//数据存放为[1,n]的范围
ll tree[SIZE];
int n;//n为每组数字个数
int lowbit(int k){
    return k&-k;
}
void add(int x,int k){
    while(x<=n){
        tree[x]+=k;
        x+=lowbit(x);
    }
}
ll sum(int x){
    ll ans=0;
    while(x!=0){
        ans+=tree[x];
        x-=lowbit(x);
    }
    return ans;
}
int pos[SIZE];
```

```
ll CountInversions(int x,int y){
    memset(tree,0,sizeof(tree));
    for(int i=1;i<=n;i++){
        pos[arr[x][i]]=i;
    }
    ll ans=0;
    for(int i=n;i;i--){
        ans+=sum(pos[arr[y][i]]);
        add(pos[arr[y][i]],1);
    }
    return ans;
}

//求三组数中有多少对数的前后顺序在三组数中都相同(三维偏序问题)
ll
   invers=(CountInversions(0,1)+CountInversions(1,2)+CountInversions(2,0))/2ll;
ll tot=((ll)n*(ll)(n-1))/2ll;//这里一定要加ll!!不然会爆
printf("%lld\n",tot-invers);
```

6.3 ZKW 线段树

6.3.1 开局

```
for(M=1;M<=n;M<<=1);//获得层数M
for(int i=1;i<=M<<1;i++)//初始化
{
    T[i]=0;
}
for(int i=1,x;i<=n;i++)//建树
{
    scanf("%d",&x);
    modify(i,x);
}
```

6.3.2 单点修改 + 区间查询

```
void modify(int n,int v)//单点修改
{
    for(T[n+=M]+=v,n>>=1;n;n>>=1)
        T[n]=T[n+n]+T[n+n+1];
}
ll query(int l, int r)//区间查询和，可调为最大
{
```

```
    ll ans=0;
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) ans+=T[l^1];
        if(r & 1) ans+=T[r^1];
    }
    return ans;
}
```

6.3.3 单点修改 + 区间查询最大字段和

```
struct Node{
    ll pre,suf,max,sum;
}T[200000],null;
Node merge(Node l, Node r)
{
    Node res;
    res.sum=l.sum+r.sum;//区间和
    res.pre=max(l.pre,l.sum+r.pre);//最大前缀
    res.suf=max(r.suf,l.suf+r.sum);//最大
    res.max=max(l.max,r.max);//最大子段和
    res.max=max(res.max,l.suf+r.pre);
    return res;
}
void modify(int n,int v)//单点修改
{
    for(T[n+=M]={v,v,v,v},n>>=1;n;n>>=1)
        T[n]=merge(T[n+n],T[n+n+1]);
}
ll query(int l, int r)//查询
{
    Node resl(null),resr(null);
//  resl.pre=resl.suf=resl.sum=0;
//  resr.pre=resr.suf=resr.sum=0;
    resl.max=resr.max=-inf;
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) resl=merge(resl,T[l^1]);
        if(r & 1) resr=merge(T[r^1],resr);
    }
    return merge(resl,resr).max;
}
```

6.3.4 区间加减 + 单点查询

```
void add(int l,int r,int v)//区间加减
{
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) T[l^1]+=v;
        if(r & 1) T[r^1]+=v;
    }
}
ll query(int n)//单点查询
{
    ll ans=0;
    for(n+=M;n;n>>=1) ans+=T[n];
    return ans;
}
```

6.3.5 区间加减 + 区间最值查询（lazy 标记）

```
void add(int L,int R,int v)//区间修改
{
    L+=M-1,R+=M+1;
    for(int l=L,r=R;l^r^1;l>>=1,r>>=1)
    {
        if(~l & 1) lazy[l^1]+=v,T[l^1]+=v;
        if(r & 1) lazy[r^1]+=v,T[r^1]+=v;
    }
    for(int l=L>>1,r=R>>1;l;l>>=1,r>>=1)
    {
        T[l]=max(T[l+l],T[l+l+1])+lazy[l];
        T[r]=max(T[r+r],T[r+r+1])+lazy[r];
    }
}
int query(int l, int r)//区间查询
{
    int lmax=-inf,rmax=-inf;
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1)
    {
        if(lazy[l]&&lmax!=-inf) lmax+=lazy[l];
        if(lazy[r]&&rmax!=-inf) rmax+=lazy[r];
        if(~l & 1) lmax=max(lmax,T[l^1]);
        if(r & 1) rmax=max(rmax,T[r^1]);
    }
```

```
    for(;l;l>>=1,r>>=1)
    {
        lmax+=lazy[l],rmax+=lazy[r];
    }
    return max(lmax,rmax);
}
```

6.4 二维线段树

```
#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1
#define INF 0x3f3f3f3f
const int SIZE=1000+50;
int n, mp[SIZE*3][SIZE*3];
void subBuild(int xrt, int l, int r, int rt) {
    mp[xrt][rt]=0;//mp[xrt][rt]=-1;
    if(l != r) {
        int m = l + r >> 1;
        subBuild(xrt, lson);
        subBuild(xrt, rson);
    }
}
void build(int l, int r, int rt) {
    subBuild(rt, 0, n, 1);
    if(l != r) {
        int m = l + r >> 1;
        build(lson);
        build(rson);
    }
}

bool leaf;//是否是叶子结点
void subUpdate(int xrt, int y, int c, int l, int r, int rt) {
    if(l == r){
        if(leaf){
            mp[xrt][rt]+=c;
            // mp[xrt][rt]=c;
        }
        else{
            mp[xrt][rt]=mp[xrt<<1][rt]+mp[xrt<<1|1][rt];
            // mp[xrt][rt]=max(mp[xrt<<1][rt],mp[xrt<<1|1][rt]);
        }
        return ;
```

```
    }
    else {
        int m = l + r >> 1;
        if(y <= m) subUpdate(xrt, y, c, lson);
        else subUpdate(xrt, y, c, rson);
        mp[xrt][rt]=mp[xrt][rt<<1]+mp[xrt][rt<<1|1];
        // mp[xrt][rt]=max(mp[xrt][rt<<1],mp[xrt][rt<<1|1]);
    }
}
//
void update(int x, int y, int c, int l, int r, int rt) {
    if(l==r){
        leaf=1;
        subUpdate(rt, y, c, 0, n, 1);return;
    }
    int m = l + r >> 1;
    if(x <= m) update(x, y, c, lson);
    else update(x, y, c, rson);
    leaf=0;subUpdate(rt, y, c, 0, n, 1);
}
int subQuery(int xrt, int yl, int yr, int l, int r, int rt) {
    if(yl <= l && r <= yr) return mp[xrt][rt];
    else {
        int m = l + r >> 1;
        int res=0;
        // int res=-1;
        if(yl <= m) res += subQuery(xrt, yl, yr, lson);
        // res=max(res,subQuery(xrt, yl, yr, lson));
        if(yr > m) res += subQuery(xrt, yl, yr, rson);
        // res=max(res,subQuery(xrt, yl, yr, rson));
        return res;
    }
}
// l≤xl≤xr≤r,0≤yl≤yr≤n
//
   查询在(限制条件1的下界，限制条件1的上界，限制条件2的下界，限制条件2的上界，)
int query(int xl, int xr, int yl, int yr, int l, int r, int rt)
   {//xl<=xr,yl<=yr
    if(xl <= l && r <= xr) return subQuery(rt, yl, yr, 0, n, 1);
    else {
        int m = l + r >> 1;
        int res = 0;
        // int res=-1;
        if(xl <= m) res += query(xl, xr, yl, yr, lson);
```

```
        // res=max(res,query(xl, xr, yl, yr, lson));
        if(xr > m) res += query(xl, xr, yl, yr, rson);
        // res=max(res,query(xl, xr, yl, yr, rson))
        return res;
    }
}
```

6.5 普通线段树

6.5.1 单点修改 + 区间查询

```
#define lson l, mid, rt << 1
#define rson mid + 1, r, rt << 1 | 1
const int SIZE = 5e5+50;
struct node{
    int l,r;int val;
}T[SIZE*3];
int arr[SIZE];// 用于存从[1,n]中的数组
void build(int l,int r,int rt){
    T[rt].l = l;T[rt].r = r;
    if(l == r){
        T[rt].val = arr[l];
        return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    build(lson); build(rson);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
void update(int tar,int rt,int k){// 单点修改将tar改为k
    if(T[rt].l==T[rt].r){
        T[rt].val += k;return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    if(tar <= mid) update(tar,rt<<1,k);
    else update(tar,rt<<1|1,k);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
int ans = 0;
void Find(int rt,int l,int r){
    if(T[rt].l>=l && T[rt].r<=r){
        ans += T[rt].val;return ;
    }
    if(T[rt<<1].r>=l) Find(rt<<1,l,r);
    if(T[rt<<1|1].l<=r) Find(rt<<1|1,l,r);
```

```
}
```

6.5.2 区间修改 + 区间查询

```
#define ll long long
const int SIZE = 1e5+5;
struct node{
    int l,r;int val;
}T[SIZE*3];
int lazy[SIZE*3];
int arr[SIZE];// arr数组用于记录[1,n]的数据
void build(int l,int r,int rt){
    T[rt].l = l;T[rt].r = r;
    if(l==r){
        T[rt].val = arr[l];return ;
    }
    int mid = (l+r)>>1;
    build(l,mid,rt<<1);
    build(mid+1,r,rt<<1|1);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}
// l,r为指针，L,R是要修改的区间[L,R],为整个区间加上x
void update(int L,int R,int x,int rt){
    if(L<=T[rt].l && R>=T[rt].r){
        T[rt].val += x*(T[rt].r-T[rt].l+1);
        lazy[rt] += x;
        return ;
    }
    int mid = (T[rt].l + T[rt].r)>>1;
    if(lazy[rt]){ // 下推标记更新一波
        T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
        T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
        lazy[rt<<1] += lazy[rt];
        lazy[rt<<1|1] += lazy[rt];
        lazy[rt] = 0;
    }
    if(L <= mid)update(L,R,x,rt<<1);
    if(R >mid)update(L,R,x,rt<<1|1);
    T[rt].val = T[rt<<1].val + T[rt<<1|1].val;
}

ll query(int L,int R,int rt){
    // printf("rt=%d",rt);
```

```
    if(L<=T[rt].l && R>=T[rt].r)return T[rt].val;
    int mid = (T[rt].l + T[rt].r)>>1;
    if(lazy[rt]){
        T[rt<<1].val += lazy[rt]*(mid-T[rt].l+1);
        T[rt<<1|1].val += lazy[rt]*(T[rt].r-mid);
        lazy[rt<<1] += lazy[rt];
        lazy[rt<<1|1] += lazy[rt];
        lazy[rt] = 0;
    }
    ll sum=0;
    if(L <= mid)sum += query(L,R,rt<<1);
    if(R > mid)sum += query(L,R,rt<<1|1);
    return sum;
}
```

6.5.3 区间染色

```
const int SIZE=1e5+50;
const int col_SIZE=1e5+50;
bool col_vis[col_SIZE];
struct node{
    int l,r;int col;
}T[SIZE*3];

void build(int root,int l,int r){
    T[root].l = l;
    T[root].r = r;
    if(l == r)return ;
    int mid=(T[root].l + T[root].r)>>1;
    build(root<<1,l,mid);
    build((root<<1)+1,mid+1,r);
}
void update(int root,int l,int r,int col){
    if(T[root].l>=l && T[root].r<=r){
        T[root].col = col;return ;
    }
    else{
        if(T[root].col > 0){
            T[root<<1].col = T[root].col;
            T[(root<<1)+1].col = T[root].col;
            T[root].col = 0;
        }
        int mid = (T[root].l + T[root].r)>>1;
```

```
        if(l>mid){
            update((root<<1)+1,l,r,col);
        }
        else if(r<=mid){
            update(root<<1,l,r,col);
        }
        else{
            update(root<<1,l,mid,col);
            update((root<<1)+1,mid+1,r,col);
        }
    }
}

void Find(int root,int l,int r){
    if(T[root].l==0 || T[root].r==0)return ;
    if(T[root].col > 0 ){
        col_vis[T[root].col] = true;
    }
    else{
        int mid = (T[ root ].l + T[ root ].r ) >> 1;
        if(l>mid){
            Find((root<<1)+1,l,r);
        }
        else if(r<=mid){
            Find(root<<1,l,r);
        }
        else{
            Find(root<<1,l,mid);
            Find((root<<1)+1,mid+1,r);
        }
    }
}
void init(int N){//N为最大的颜色
    for(int i=0;i<=N;i++){
        col_vis[i]=0;
    }
}

int tot(int l,int r,int N){//统计[l,r]内颜色范围为[1,N]共有几种
    init(N);
    Find(1,l,r);
    int tot=0;
    for(int i=0;i<=N;i++){
        if(col_vis[i])tot++;
```

```
    }
    return tot;
}
```

6.5.4 区间修改 + 区间查询：矩阵

```
/*
    本模板需要配合矩阵快速幂食用qwq
*/
typedef long long ll;
int n;
mat A(2,2),B(2,2);
const int SIZE=1e5+50;
char str[SIZE];
struct node{
    int l,r;
    bool lazy=0;
}T[SIZE*3];
mat ma[SIZE*3][2];//0是正序，1是反序
void push_up(int root){
    ma[root][0]=ma[root<<1][0]*ma[(root<<1)|1][0];
    ma[root][1]=ma[root<<1][1]*ma[(root<<1)|1][1];
    return ;
}
void build(int root,int l,int r){
    T[root].l=l;T[root].r=r;
    if(l==r){
        if(str[l]=='A'){
            ma[root][0].n = A.n;ma[root][0].m = A.m;
            for(int i=0;i<A.n;i++)
            for(int j=0;j<A.m;j++)
                ma[root][0].v[i][j]=A.v[i][j];
            //————————————————————————————
            ma[root][1].n = B.n;ma[root][1].m = B.m;
            for(int i=0;i<B.n;i++)
            for(int j=0;j<B.m;j++)
                ma[root][1].v[i][j]=B.v[i][j];
        }
        else{
            ma[root][0].n = B.n;ma[root][0].m = B.m;
            for(int i=0;i<B.n;i++)
            for(int j=0;j<B.m;j++)
                ma[root][0].v[i][j]=B.v[i][j];
```

```
            //————————————————————————————————
            ma[root][1].n = A.n;ma[root][1].m = A.m;
            for(int i=0;i<A.n;i++)
            for(int j=0;j<A.m;j++)
                ma[root][1].v[i][j]=A.v[i][j];
        }
        return ;
    }
    int mid=(T[root].l+T[root].r)>>1;
    build(root<<1,l,mid);
    build((root<<1)|1,mid+1,r);
    push_up(root);
    return ;
}

void push_down(int root){
    if(T[root].lazy){
        swap(ma[root<<1][0],ma[root<<1][1]);
        swap(ma[(root<<1)|1][0],ma[(root<<1)|1][1]);
        T[root<<1].lazy=!T[root<<1].lazy;
        T[(root<<1)|1].lazy=!T[(root<<1)|1].lazy;
        T[root].lazy=0;
    }
}
void update(int root,int l,int r){
    if(l<=T[root].l && T[root].r<=r){
        T[root].lazy = !T[root].lazy;
        swap(ma[root][0],ma[root][1]);
        return ;
    }
    push_down(root);
    int mid = (T[root].l + T[root].r)>>1;
    if(l <= mid)update(root<<1,l,r);
    if(r > mid)update((root<<1)|1,l,r);
    push_up(root);
}

mat tmp(2,2);// 对最后的答案矩阵进行修改
void find(int root,int l,int r){
    if(l <= T[root].l && T[root].r<=r){
        tmp = tmp*ma[root][0];
        return;
    }
    push_down(root);
```

```
    int mid = (T[root].l + T[root].r)>>1;
    if(l <= mid)find(root<<1,l,r);
    if(mid < r)find((root<<1)|1,l,r);
    return;
}
int main(){
    A.v[0][0]=1;A.v[0][1]=0;A.v[1][0]=1;A.v[1][1]=1;
    B.v[0][0]=1;B.v[0][1]=1;B.v[1][0]=0;B.v[1][1]=1;
    int q;scanf("%d%d",&n,&q);
    scanf("%s",str+1);
    build(1,1,n);
    while(q--){
        int op;scanf("%d",&op);
        if(op==1){
            int L,R;scanf("%d%d",&L,&R);
            update(1,L,R);
        }
        else{
            int L,R;ll AA,BB;scanf("%d%d%lld%lld",&L,&R,&AA,&BB);
            tmp.v[0][0]=1;
            tmp.v[0][1]=0;
            tmp.v[1][0]=0;
            tmp.v[1][1]=1;
            mat qwq(1,2);
            qwq.v[0][0]=AA%mod;qwq.v[0][1]=BB%mod;
            find(1,L,R);
            qwq = qwq*tmp;
            qwq.display();
        }
    }
}
```

6.6 普通平衡树 Treap

```
#define inf 0x3f3f3f3f
const int maxn = 1e6+5;
int ch[maxn][2];
int val[maxn],dat[maxn];
int sz[maxn],cnt[maxn];
int tot,root;
int New(int v){// 辅助函数
    val[++tot] = v;
    dat[tot] = rand();
```

```
    sz[tot] = 1;
    cnt[tot] = 1;
    return tot;
}
void pushup(int id){// 辅助函数
    sz[id] = sz[ch[id][0]] + sz[ch[id][1]] + cnt[id];
}
void build(){// 辅助函数
    root = New(-INF),ch[root][1] = New(INF);
    pushup(root);
}
void Rotate(int &id,int d){// 辅助函数
    int temp = ch[id][d ^ 1];
    ch[id][d ^ 1] = ch[temp][d];
    ch[temp][d] = id;
    id = temp;
    pushup(ch[id][d]),pushup(id);
}
void insert(int &id,int v){// 插入一个数值为v
    if(!id){
        id = New(v);return ;
    }
    if(v == val[id])cnt[id]++;
    else{
        int d = v < val[id] ? 0 : 1;
        insert(ch[id][d],v);
        if(dat[id] < dat[ch[id][d]])Rotate(id,d ^ 1);
    }
    pushup(id);
    }
void Remove(int &id,int v){// 删除一个指为v数(若有多个相同的数，因只删除一个)
    if(!id)return ;
    if(v == val[id]){
        if(cnt[id] > 1){cnt[id]--,pushup(id);return ;}
        if(ch[id][0] || ch[id][1]){
            if(!ch[id][1] || dat[ch[id][0]] > dat[ch[id][1]]){
                Rotate(id,1),Remove(ch[id][1],v);
                }
            else Rotate(id,0),Remove(ch[id][0],v);
            pushup(id);
        }
        else id = 0;
        return ;
    }
```

```
    v < val[id] ? Remove(ch[id][0],v) : Remove(ch[id][1],v);
    pushup(id);
}
int get_rank(int id,int v){//
   查询v数的排名(排名定义为比当前数小的数的个数+1。
   若有多个相同的数，因输出最小的排名)
    if(!id)return 0;
    if(v == val[id])return sz[ch[id][0]] + 1;
    else if(v < val[id])return get_rank(ch[id][0],v);
    else return sz[ch[id][0]] + cnt[id] + get_rank(ch[id][1],v);
}
int get_val(int id,int rank){// 查询排名为rank的数
    if(!id)return INF;
    if(rank <= sz[ch[id][0]])return get_val(ch[id][0],rank);
        else if(rank <= sz[ch[id][0]] + cnt[id])return val[id];
    else return get_val(ch[id][1],rank - sz[ch[id][0]] - cnt[id]);
}
int get_pre(int v){// 求v的前驱(前驱定义为小于v，且最大的数)
    int id = root,pre;
    while(id){
        if(val[id] < v)pre = val[id],id = ch[id][1];
        else id = ch[id][0];
    }
    return pre;
}
int get_next(int v){// 求v的后继(后继定义为大于v，且最小的数)
    int id = root,next;
    while(id){
        if(val[id] > v)next = val[id],id = ch[id][0];
        else id = ch[id][1];
    }
    return next;
}
int main(){
    build();int x;
    int shk;scanf("%d",&shk);
    while(shk--){
        int op;scanf("%d",&op);
        switch(op){
            case 1:scanf("%d",&x);insert(root,x);break;
            case 2:scanf("%d",&x);Remove(root,x);break;
            case 3:scanf("%d",&x);printf("%d\n",get_rank(root,x)-1);break;
            case 4:scanf("%d",&x);printf("%d\n",get_val(root,x+1));break;
            case 5:scanf("%d",&x);printf("%d\n",get_pre(x));break;
```

```
            case 6:scanf("%d",&x);printf("%d\n",get_next(x));break;
        }
    }
}
```

6.7 树链剖分

```
int sum[MAXM*4],add[MAXM*4];
int a[MAXM],n;
int idx;
int first[MAXM];
struct edge{
    int v,next;
}e[MAXM*2];
int f[MAXM],son[MAXM],size[MAXM],dfn[MAXM],dep[MAXM],top[MAXM],seq[MAXM];
int cnt;
int mod;
/*
f是节点的父亲,son是重儿子,size是以该节点为根的子树大小,
   dfn是给节点重新编上的序号.
seq是与dfn相反的数组,表示标到的这个号表示的原节点, top是目前节点所在链的顶端.
dep是结点的深度
*/

//————————————————————加边和预处理————————————————————————
void eadd(int a,int b)
{
    e[idx].v = b;
    e[idx].next = first[a];
    first[a] = idx++;
}
void init()
{
    memset(first,-1,sizeof(first));
    idx = 1;
    cnt = 0;
}
void dfs1(int u,int fa,int depth)
{
    f[u] = fa; size[u] = 1; dep[u] = depth;
    int maxson = -1;
    for(int i = first[u];i != -1;i = e[i].next){
        int v = e[i].v;
```

```
        if(v == fa) continue;
        dfs1(v,u,depth+1);
        size[u] += size[v];
        if(size[v]>maxson) son[u] = v,maxson = size[v];
    }
}
void dfs2(int u,int t)
{
    top[u] = t;
    dfn[u] = ++cnt;
    seq[cnt] = a[u];
    if(!son[u]) return;
    dfs2(son[u],t);
    for(int i = first[u];i != -1;i = e[i].next){
        int v = e[i].v;
        if(v != son[u]&&v != f[u]) dfs2(v,v);
    }
}


//————————————————————————线段树————————————————————————————
void pushup(int rt)
{
    sum[rt] = (sum[rt<<1]+sum[rt<<1|1])%mod;
}
void build(int l,int r,int rt)
{
    if(l == r){
        sum[rt] = seq[l]%mod;
        return;
    }
    int m = (l+r)>>1;
    build(l,m,rt<<1);
    build(m+1,r,rt<<1|1);
    pushup(rt);
}
void pushdown(int rt,int ln,int rn)
{
    if(add[rt]){
        add[rt<<1] = (add[rt<<1] + add[rt])%mod;
        add[rt<<1|1] = (add[rt<<1|1] + add[rt])%mod;
        sum[rt<<1] = (sum[rt<<1] + add[rt]*ln%mod)%mod;
        sum[rt<<1|1] = (sum[rt<<1|1] + add[rt]*rn%mod)%mod;
        add[rt] = 0;
```

```
    }
}
void update(int L,int R,int C,int l,int r,int rt)
{
    if(L <= l&&r <= R){
        sum[rt] = (sum[rt] + C*(r-l+1)%mod)%mod;
        add[rt] = (add[rt] + C)%mod;
        return;
    }
    int m = (l+r)>>1;
    pushdown(rt,m-l+1,r-m);
    if(L <= m) update(L,R,C,l,m,rt<<1);
    if(R >  m) update(L,R,C,m+1,r,rt<<1|1);
    pushup(rt);
}

ll query(int L,int R,int l,int r,int rt)
{
    if(L <= l&&r <= R){
        return sum[rt];
    }
    int m = (l+r)>>1;
    pushdown(rt,m-l+1,r-m);
    ll ans = 0;
    if(L <= m) ans = (ans + query(L,R,l,m,rt<<1))%mod;
    if(R >  m) ans = (ans + query(L,R,m+1,r,rt<<1|1))%mod;
    return ans;
}


//————————————————————————树上加、树上求和————————————————————————
void tadd(int x,int y,int k)
{
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        update(dfn[top[x]],dfn[x],k,1,n,1);
        x = f[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    update(dfn[x],dfn[y],k,1,n,1);
}
ll tsum(int x,int y)
{
    ll ans = 0;
```

```
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        ans = (ans + query(dfn[top[x]],dfn[x],1,n,1))%mod;
        x = f[top[x]];
    }
    if(dep[x]>dep[y]) swap(x,y);
    ans = (ans + query(dfn[x],dfn[y],1,n,1))%mod;
    return ans;
}


int main()
{
    int m;
    int num;//根节点序号
    scanf("%d%d%d%d",&n,&m,&num,&mod);
    init();
    for(int i = 1;i <= n;i++){
        scanf("%d",&a[i]);
    }
    for(int i = 1;i <= n-1;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        eadd(u,v);
        eadd(v,u);
    }
    dfs1(num,0,1);
    dfs2(num,num);
    build(1,n,1);
    for(int i = 1;i <= m;i++){
        int op;
        scanf("%d",&op);
        if(op == 1){//将树从x到y节点最短路径上所有节点的值都加上z
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            tadd(x,y,z%mod);
        }
        else if(op == 2){//求树从x到y节点最短路径上所有节点的值之和
            int x,y;
            scanf("%d%d",&x,&y);
            printf("%lld\n",tsum(x,y));
        }
        else if(op == 3){//将以x为根节点的子树内所有节点值都加上z
            int x,z;
```

```
            scanf("%d%d",&x,&z);
            update(dfn[x],dfn[x]+size[x]-1,z%mod,1,n,1);
        }
        else if(op == 4){//求以x为根节点的子树内所有节点值之和
            int x;
            scanf("%d",&x);
            printf("%lld\n",query(dfn[x],dfn[x]+size[x]-1,1,n,1)%mod);
        }
    }
    return 0;
}
```

7 图论

7.1 前向星

```
struct edge{
    int v,next,w;
}e[MAXM*2];
void add(int a,int b,int c)
{
    e[idx].v = b;e[idx].w = c;
    e[idx].next = first[a];
    first[a] = idx++;
}
void init()
{
    memset(first,-1,sizeof(first));
    idx = 1;
}
```

7.2 最短路

7.2.1 Dijkstra+ 堆优化

```
struct node{
    int id,cost;
    node(int a,int b):id(a),cost(b){}
    bool operator < (const node &t) const
    {
        return t.cost < cost;
    }
```

```
};
void dijkstra(int x)
{
    priority_queue<node> q;
    for(int i = 1;i <= n;i++){
        vis[i] = 0;
        dist[i] = INF;
    }
    dist[x] = 0;
    q.push(node(x,0));
    while(!q.empty()){
        node cur = q.top();
        q.pop();
        if(vis[cur.id]) continue;
        vis[cur.id] = 1;
        for(int i = first[cur.id];i!=-1;i=e[i].next){
            if(dist[e[i].v]>dist[cur.id]+e[i].w){
                dist[e[i].v] = dist[cur.id] + e[i].w;
                q.push(node(e[i].v,dist[e[i].v]));
            }
        }
    }
}
```

7.2.2 SPFA（判环）

```
bool inq[MAXN];//是否在队列中
int cnt[MAXN];//入队列次数
int dist[MAXN];
bool spfa(int start)
{
    queue<int> q;
    for(int i = 1;i <= n;i++){
        dist[i] = INF;
        inq[i] = 0;
        cnt[i] = 0;
    }
    dist[start] = 0;
    cnt[start] = 1;
    q.push(start);
    while(!q.empty()){
        int cur = q.front();
        q.pop();
```

```
        inq[cur] = 0;
        for(int i = first[cur];i != -1;i = e[i].next){
            int v = e[i].v;
            int w = e[i].w;
            if(dist[v] > dist[cur] + w){
                dist[v] = dist[cur] + w;
                if(!inq[v]){
                    inq[v] = 1;
                    q.push(v);
                    if(++cnt[v] > n) return 0;
                    //若入队列次数大于n，说明存在环
                }
            }
        }
    }
    return 1;
}
```

7.2.3 Floyd

```
void floyd()
{
    for(int k = 1;k <= n;k++){
        for(int i = 1;i <= n;i++){
            for(int j = 1;j <= n;j++){
                if(cost[i][j] > cost[i][k] + cost[k][j]){
                    cost[i][j] = cost[i][k] + cost[k][j];
                    path[j] = k;//path[]记录最短路径
                }
            }
        }
    }
}
```

7.3 第 K 短路

```
int dist[1010];
int first[1010];//正向图
int rfirst[1010];//反向图
int vis[1010];
int times[1010];//点的访问次数
int idx,ridx;
```

```
struct node{
    int p,g,h;//p表示点的编号，g为点到终点的距离（估价），
       h为点到起点的距离（实际）
    bool operator < (const node &t)const
    {
        return t.g+t.h<g+h;
    }
};
struct qnode{
    int id;
    int cost;
    qnode(int a,int b):id(a),cost(b){}
    bool operator < (const qnode &t) const
    {
        return t.cost < cost;
    }
};
struct edge{
    int v,next,w;
}e[100100],re[100100];
void add(int a,int b,int c)
{
    e[idx].v = b;e[idx].w = c;
    e[idx].next = first[a];
    first[a]=idx++;
}
void radd(int a,int b,int c)
{
    re[ridx].v=b;re[ridx].w=c;
    re[ridx].next = rfirst[a];
    rfirst[a]=ridx++;
}
void dijkstra(int x)
{
    priority_queue<qnode> q;
    for(int i = 1;i <= n;i++){
        vis[i]=0;
        dist[i]=INF;
    }
    dist[x]=0;
    q.push(qnode(x,0));
    while(!q.empty()){
        qnode cur = q.top();
        q.pop();
```

```
        if(vis[cur.id]) continue;
        vis[cur.id] = 1;
        for(int i = rfirst[cur.id];i!=-1;i=re[i].next){
            if(dist[re[i].v]>dist[cur.id]+re[i].w){
                dist[re[i].v] = dist[cur.id]+re[i].w;
                q.push(qnode(re[i].v,dist[re[i].v]));
            }
        }
    }
}

int A_star(int start,int end,int k)
{
    memset(times,0,sizeof(times));
    priority_queue<node> q;
    node t1;
    t1.g = t1.h = 0;
    t1.p = start;
    q.push(t1);
    while(!q.empty()){
        node t = q.top();
        q.pop();
        times[t.p]++;
        if(times[t.p]==k&&t.p==end) return t.h+t.g;
        if(times[t.p]>k) continue;
        for(int i = first[t.p];i!=-1;i=e[i].next){
            node tmp;
            tmp.p = e[i].v;
            tmp.g = dist[e[i].v];
            tmp.h = e[i].w + t.h;
            q.push(tmp);
        }
    }
    return -1;
}
void init()
{
    memset(first,-1,sizeof(first));
    memset(rfirst,-1,sizeof(rfirst));
    idx = 1;
    ridx = 1;
}
int main()
{
```

```
    scanf("%d%d",&n,&m);
    init();
    for(int i = 1;i <= m;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        add(u,v,w);
        radd(v,u,w);
    }
    scanf("%d%d%d",&start,&end,&k);
    if(start==end) k++;//若题目要求必须走动时加上
    dijkstra(end);
    int ans = A_star(start,end,k);
    printf("%d\n",ans);
    return 0;
}
```

7.4 最小环

7.4.1 Floyd

```
int pos[MAXN][MAXN];//pos[i][j]:i到j的最短路的路径
vector<int> path;//最小环路径
int ans;//最小环
//ans == INF  说明无环
void getpath(int x,int y)
{
    if(!pos[x][y]) return;
    getpath(x,pos[x][y]);
    path.push_back(pos[x][y]);
    getpath(pos[x][y],y);
}
void floyd()
{
    ans = INF;
    memcpy(dist,g,sizeof(dist));
    for(int k = 1;k <= n;k++){
        for(int i = 1;i < k;i++){
            for(int j = i+1;j < k;j++){
                if(dist[i][j]+g[j][k]+g[k][i]<ans){
                    ans = dist[i][j]+g[j][k]+g[k][i];
                    ans.clear();
                    ans.push_back(i);
                    getpath(i,j);
                    ans.push_back(j);
```

```
                    ans.push_back(k);
                }
            }
        }
        for(int i = 1;i <= n;i++){
            for(int j = 1;j <= n;j++){
                if(dist[i][k]+dist[k][j]<dist[i][j]){
                    dist[i][j] = dist[i][k]+dist[k][j];
                    pos[i][j] = k;
                }
            }
        }
    }
}
```

7.4.2 Dijkstra+ 剪枝

```
//边的计数idx从2开始，idx = 2;
ll dijkstra(int x,int y,int k)
{
    priority_queue<node> q;
    for(int i = 1;i <= cnt;i++){
        vis[i] = 0;
        dist[i] = INF;
    }
    dist[x] = 0;
    q.push(node(x,0));
    while(!q.empty()){
        node cur = q.top();
        q.pop();
        if(cur.cost>ans-e[k].w) break;  //剪枝
        if(vis[cur.id]) continue;
        vis[cur.id] = 1;
        for(int i = first[cur.id];i!=-1;i=e[i].next){
            if(i == k||i == (k^1)) continue;
            if(dist[e[i].v]>dist[cur.id]+e[i].w){
                dist[e[i].v] = dist[cur.id]+e[i].w;
                q.push(node(e[i].v,dist[e[i].v]));
            }
        }
    }
    return dist[y];
}
```

```
for(int i = 2;i <= m*2+1;i+=2){
    ans = min(ans,dijkstra(e[i].u,e[i].v,i)+e[i].w);
}
```

7.5 网络流

7.5.1 二分图匹配

```
int dfs(int u)
{
    for(int i = first[u];i!=-1;i = e[i].next){
        int v = e[i].to;
        if(!vis[v]){
            vis[v] = 1;
            if(linker[v]==-1||dfs(linker[v])){
                linker[v] = u;
                return 1;
            }
        }
    }
    return 0;
}
int hungary()
{
    int res =  0;
    memset(linker,-1,sizeof(linker));
    for(int  u = 1; u <= n;u++){
        memset(vis,0,sizeof(vis));
        if(dfs(u)){
            res++;
        }
    }
    return res;
}
```

7.5.2 最大流

Dinic

```
//1.用BFS建立分层图
//2.用DFS的方法寻找一条由源点到汇点的路径,获得这条路径的流量x.
//根据这条路径修改整个图,将所经之处正向边流量减少x,反向边流量增加x
//重复步骤2,直到DFS找不到新的路径时,重复步骤1
//时间复杂度O(n^2*m)
```

```
int bfs()
{
    memset(dis,-1,sizeof(dis));
    queue<node> q;
    q.push(node(1,0));
    dis[1] = 0;
    while(!q.empty()){
        node cur = q.front();
        q.pop();
        for(int i = first[cur.id];i!=-1;i = e[i].next){
            if(e[i].w == 0) continue;
            if(dis[e[i].v] == -1){
                dis[e[i].v] = cur.cost+1;
                q.push(node(e[i].v,dis[e[i].v]));
            }
        }
    }
    if(dis[n] == -1) return 0;
    return 1;
}
int dfs(int x,int low)
{
    if(x == n) return low;
    for(int i = first[x];i!=-1;i = e[i].next){
        int a = 0;
        if(e[i].w > 0&&dis[e[i].v] == dis[x]+1&&(a =
           dfs(e[i].v,min(low,e[i].w)))){
            e[i].w -= a;
            add(e[i].v,x,a);
            return a;
        }
    }
    return 0;
}
int main()
{
    init();
    scanf("%d%d",&n,&m);
    for(int i = 1;i <= m;i++){
      int u,v,w;
      scanf("%d%d%d",&u,&v,&w);
      add(u,v,w);
    }
    ll ans = 0;
```

```
    int sum;
    while(bfs()){
      while(sum = dfs(1,INF)){
        ans += sum;
      }
    }
    printf("Max flow: %lld\n",ans);
    return 0;
}
```

7.6 点分治

```
/*
    HYSBZ - 2152 计算两个点之间所有边上数的和加起来恰好是3的倍数的个数的概率
*/
const int SIZE=20050;
struct edge{//邻接表存数据
    int to,nex,w;
}e[SIZE*2];
int n,cnt,ans,rt,sum;
int head[SIZE],son[SIZE],f[SIZE],d[SIZE],t[5];
bool vis[SIZE];
void add(int x,int y,int w){//邻接表插入
    e[++cnt].to=y;e[cnt].nex=head[x];head[x]=cnt;e[cnt].w=w;
    e[++cnt].to=x;e[cnt].nex=head[y];head[y]=cnt;e[cnt].w=w;
}
void Get_Root(int x,int fa){
    son[x]=1;f[x]=0;
    for(int i=head[x];i;i=e[i].nex){
        if(!vis[e[i].to] && e[i].to != fa){
            Get_Root(e[i].to,x);
            son[x]+=son[e[i].to];
            f[x]=max(f[x],son[e[i].to]);
        }
    }
    f[x]=max(f[x],sum-son[x]);
    if(f[x]<f[rt])rt=x;
}
void Get_Deep(int x,int fa){
    t[d[x]]++;
    for(int i=head[x];i;i=e[i].nex)
        if(!vis[e[i].to] && e[i].to !=fa){
            d[e[i].to]=(d[x]+e[i].w)%3;
```

```
            Get_Deep(e[i].to,x);
        }
}
int cal(int x,int now){
    t[0]=t[1]=t[2]=0;
    d[x]=now;Get_Deep(x,0);
    return t[1]*t[2]*2+t[0]*t[0];
}
void find(int x){
    ans+=cal(x,0);
    vis[x]=1;
    for(int i=head[x];i;i=e[i].nex){
        if(!vis[e[i].to]){
            ans-=cal(e[i].to,e[i].w);
            rt=0;sum=son[e[i].to];
            Get_Root(e[i].to,0);
            find(rt);
        }
    }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<n;i++){
        int u,v,w;scanf("%d%d%d",&u,&v,&w);w%=3;
        add(u,v,w);
    }
    f[0]=sum=n;
    Get_Root(1,0);
    find(rt);
    int t=__gcd(ans,n*n);
    printf("%d/%d",ans/t,n*n/t);
}
```

8 莫队

8.1 区间查询，统计两个相同概率

```
/*
    HYSBZ - 2038 小Z的袜子
    统计[L,R]区间内选两只袜子，颜色相同的概率。区间查询，统计两个相同概率。
    N,M ≤ 50000，1 ≤ L < R ≤ N，Ci ≤ N。
*/
#define ll long long
```

```
const ll SIZE=50050;
struct node{//查询
    ll L,R;
    ll id,res1,res2;
}q[SIZE];

ll c[SIZE];//n只袜子的颜色
ll n,m,unit;//n只瓦子，m次查询
ll num[SIZE];

ll com(ll nn,ll mm){
    if(mm>nn/2)mm=nn-mm;
    ll aa=1,bb=1;
    for(ll i=1;i<=mm;i++){
        aa*=nn+1-i;
        bb*=i;
        if(aa%bb==0){
            aa/=bb;
            bb=1;
        }
    }
    ll tmp=aa/bb;
    return tmp;
}

//朴素排序
bool cmp(node a,node b){
    if(a.L/unit != b.L/unit)return a.L/unit < b.L/unit;
    else return a.R < b.R;
}

//对答案进行排序
bool cmp_id(node a,node b){
    return a.id<b.id;
}

int main(){
    cin>>n>>m;
    unit=sqrt(n);//分块
    for(ll i=1;i<=n;i++)cin>>c[i];
    for(ll i=1;i<=m;i++){
        q[i].id=i;
        cin>>q[i].L>>q[i].R;
    }
```

```
    sort(q+1,q+1+m,cmp);
    ll L=1,R=0;
    ll sum=0;
    for(ll i=1;i<=m;i++){
        while(R<q[i].R){
            R++;
            if(num[c[R]]>1)sum-=com(num[c[R]],2);
            num[c[R]]++;
            if(num[c[R]]>1)sum+=com(num[c[R]],2);
        }
        while(R>q[i].R){
            if(num[c[R]]>1)sum-=com(num[c[R]],2);
            num[c[R]]--;
            if(num[c[R]]>1)sum+=com(num[c[R]],2);
            R--;
        }
        while(L<q[i].L){
            if(num[c[L]]>1)sum-=com(num[c[L]],2);
            num[c[L]]--;
            if(num[c[L]]>1)sum+=com(num[c[L]],2);
            L++;
        }
        while(L>q[i].L){
            L--;
            if(num[c[L]]>1)sum-=com(num[c[L]],2);
            num[c[L]]++;
            if(num[c[L]]>1)sum+=com(num[c[L]],2);
        }
        ll under=com(q[i].R-q[i].L+1,2);
        ll gcdd=__gcd(sum,under);
        q[i].res1=sum/gcdd;q[i].res2=under/gcdd;
    }
}
```

8.2 时间戳 + 统计有多少个不同的数

```
const int SIZE=10500;
struct node{
    int l,r,time,id;
}q[SIZE];
struct Update{
    int x,y;
}upd[SIZE];
```

```
int l,r;
const int COLSIZE=1000500;
int col[COLSIZE];int unit;int bl[SIZE];int vis[COLSIZE];int ans=0;int
   res[SIZE];
void pop(int x){
    ans -= !--vis[x];
}
void push(int x){
    ans += !vis[x]++;
}
bool cmp(node a,node b){// 奇偶排序
    return (bl[a.l]^bl[b.l]) ? bl[a.l] < bl[b.l] : ((bl[a.r]^bl[b.r]) ?
       bl[a.r] < bl[b.r] : a.time < b.time);
}
void modify(int x){
    if(l <= upd[x].x && upd[x].x <= r){
        pop(col[upd[x].x]);
        push(upd[x].y);
    }
    swap(upd[x].y,col[upd[x].x]);
}
int main(){
    int n,m;
    scanf("%d%d",&n,&m);
    int tmpb=pow(double(n),double(2.0/3.0));
    for(int i=1;i<=n;i++){
        scanf("%d",&col[i]);
        bl[i]=i/tmpb;
    }
    int time=0;char ch;int x,y;
    int mm=1,updm=1;
    for(int i=1;i<=m;i++){
        cin>>ch>>x>>y;
        if(ch=='Q'){// Q查询
            q[mm].l=x;q[mm].r=y;q[mm].time=time;q[mm].id=mm;mm++;
        }
        else if(ch=='R'){// R更换画笔
            upd[updm].x=x,upd[updm].y=y;time++;updm++;
        }
    }
    sort(q+1,q+mm,cmp);
    l=1,r=0;int tp=0;
    for(int i=1;i<mm;i++){
        while(l < q[i].l)pop(col[l++]);
```

```
        while(l > q[i].l)push(col[--l]);
        while(r < q[i].r)push(col[++r]);
        while(r > q[i].r)pop(col[r--]);
        while(tp < q[i].time)modify(++tp);
        while(tp > q[i].time)modify(tp--);
        res[q[i].id]=ans;
    }
}
```

8.3 树状数组维护区间两数之差

```
/*
    HDU – 6534 树状数组维护区间两数之差
    i<j，|ai-aj|≤ K，n  (1≤n≤27000), m (1≤m≤27000) and K (1≤K≤109)
*/
//树状数组部分————————————————
int lowbit(int k){return k&-k;}
void add(int x,int k){
    while(x<=n){
        tree[x]+=k;
        x+=lowbit(x);
    }
}
int Get_sum(int x){
    int ans=0;
    while(x!=0){
        ans+=tree[x];
        x-=lowbit(x);
    }
    return ans;
}
//————————————————————————————
/*
// 统计 a[i]^a[i+1]^...a[j]=k,非树状数组
// 1 ≤ n, m ≤ 100 000， 0 ≤ k ≤ 1 000 000,(0 ≤ ai ≤ 1 000 000)
ll tmp=0;
void add(ll x){
    tmp += cnt[sum[x]^k];
    cnt[sum[x]]++;
}
void del(ll x){
    cnt[sum[x]]--;
    tmp -= cnt[sum[x]^k];
```

```
}
*/
int sum=0;
void push(int x){
    sum+=Get_sum(rr[x])-Get_sum(ll[x]-1);
    add(a[x],1);
}
void pop(int x){
    add(a[x],-1);
    sum-=Get_sum(rr[x])-Get_sum(ll[x]-1);
}

int main(){
    sort(q+1,q+1+m,cmp);
    int L=1,R=0;
    for(int i=1;i<=m;i++){
        while(L > q[i].l)push(--L);
        while(R < q[i].r)push(++R);
        while(L < q[i].l)pop(L++);
        while(R > q[i].r)pop(R--);
        display[q[i].id]=sum;//用于输出
    }
}
```

8.4 统计有多少个不同的数

```
/*
    HYSBZ - 1878 统计[L,R]区间内有多少个不同的数
*/
#define ll long long
const ll SIZE=1005000;
struct node{
    ll L,R;
    ll id,res;
}q[SIZE];
ll c[SIZE];ll n,m;ll num[SIZE];ll sum=0;ll dis[SIZE];ll pos[SIZE];
bool cmp(const node& a,const node& b){//奇偶排序
    return (pos[a.L]^pos[b.L])?pos[a.L]<pos[b.L]:((pos[a.L]
       &1)?a.R<b.R:a.R>b.R);
}
ll ans=0;
int main(){
    scanf("%lld",&n);//n个数
```

```
    ll tmpb=sqrt(n);
    sort(q+1,q+1+m,cmp);
    ll L=1,R=0;
    for(ll i=1;i<=m;i++){
        while(L<q[i].L)ans-=!--num[c[L++]];
        while(L>q[i].L)ans+=!num[c[--L]]++;
        while(R<q[i].R)ans+=!num[c[++R]]++;
        while(R>q[i].R)ans-=!--num[c[R--]];
        dis[q[i].id]=ans;//用于输出
    }

}
```

8.5 回滚莫队

```
/*
    给定一个长为n的序列{a1,a2,a3..}, 询问区间a1*cnt(a1) + a2*cnt(a2) + ...
       的最大值, 即某个值乘上出现次数
*/
#define inf 0x3f3f3f3f3f3f
#define N 100005
#define M 4000005
typedef long long ll;
using namespace std;
struct point{
    ll l,r,id;
    ll res;
}q[N];
ll vis[M];
ll a[N],pos[N];
ll s[N],as[N];
bool cmp(const point &a,const point &b)
{
    return (pos[a.l]^pos[b.l]) ? pos[a.l] < pos[b.l] : a.r < b.r;
}
bool cmp2(const point &a,const point &b)
{
    return a.id<b.id;
}
ll rpos[N];
ll tempv[N];
int main()
{
```

```
    ll n,m;
    scanf("%lld%lld",&n,&m);
    ll b=sqrt(n);
    ll bnum=ceil((double) n/b);
    for(int i=1;i<=bnum;i++)
    {
        rpos[i]=b*i;
        for(int j=b*(i-1)+1;j<=rpos[i];j++)
        {
            pos[j]=i;
        }
    }
    rpos[bnum]=n;
    for(int i=1;i<=n;i++)
    {
        scanf("%lld",&a[i]);
        s[i]=a[i];
    }
    sort(s+1,s+1+n);
    ll tot=unique(s+1,s+1+n)-s-1;
    for(int i=1;i<=n;i++)
    {
        as[i]=lower_bound(s+1,s+1+tot,a[i])-s;
    }
    for(int i=1;i<=m;i++)
    {
        scanf("%lld%lld",&q[i].l,&q[i].r);
        q[i].id=i;
    }
    sort(q+1,q+1+m,cmp);
    int i=1;
    for(ll k=0;k<=bnum;k++)
    {
        ll l=rpos[k]+1,r=rpos[k];
        ll ans=0;
        memset(vis,0,sizeof(vis));
        for(;pos[q[i].l]==k;i++)
        {
            if(pos[q[i].l]==pos[q[i].r])
            {
                ll res=0;
                for(int j=q[i].l;j<=q[i].r;j++)
                {
                    tempv[as[j]]=0;
```

```
                }
                for(int j=q[i].l;j<=q[i].r;j++)
                {
                    res=max(res,(++tempv[as[j]])*a[j]);
                }
                q[i].res=res;
                continue;
            }
            while(r<q[i].r)
            {
                r++;
                ans=max(ans,(++vis[as[r]])*a[r]);
            }
            ll temp=ans;
            while(l>q[i].l)
            {
                l--;
                ans=max(ans,(++vis[as[l]])*a[l]);
            }
            q[i].res=ans;
            while(l<rpos[k]+1)
            {
                --vis[as[l++]];
            }
            ans=temp;
        }
    }
    sort(q+1,q+1+m,cmp2);
    for(int i=1;i<=m;i++)
    {
        printf("%lld\n",q[i].res);
    }
}
```

9 Java

9.1 开头

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.text.DecimalFormat;
public class Main {
```

```
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        DecimalFormat df=new DecimalFormat("0.00");//小数保留2位
      //System.out.println(df.format(a));
        while(cin.hasNext()){
          BigDecimal a= cin.nextBigDecimal();
        }//多组输入
    }
}
```

9.2 加减乘除等

```
/*
加      a+b: a=a.add(b);
减      a-b: a=a.subtract(b);
乘      a*b: a=a.multiply(b);
除      a/b: a=a.divide(b);
求余        a%b: a=a.mod(b);
绝对值      abs a=a.abs();
转换        a=b: b=BigInteger.valueOf(a);
比较        a.compareTo(b)  =-1,a<b;
                              =0,a=b;
                                =1,a>b;
a.equals(b)   a==b返回ture
a.mod(b)    //求余数即a%b
a.gcd(b)     //求最大公约数
a.max(b)    //求最大值
a.min(b)     //求最小值
a.pow(b)    //求a^b的大数
*/
public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        BigInteger a;
        BigInteger b;
        BigInteger t1=new BigInteger(1);
        BigInteger t2=new BigInteger("12313");
        int t=5;
        BigInteger t3=BigInteger.valueOf(t);//t3=5;
        BigDecimal d=new BigDecimal(5.3435345);
        BigInteger dd=d.toBigInteger();//小数型转整形无四舍五入//d=5;
        BigInteger []s = new BigInteger[4040];//初始化一个4040的大数数组
    }
```

```
}
```

9.3 Java 大数二分

```
//这是将sqrt（5）精确到100位的二分
BigDecimal d=new BigDecimal(5);
BigDecimal eps=new BigDecimal("0.00000 00000 00000 00000 00000 00000 00000
   00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
   00000 01");//（100个0）
BigDecimal l=new BigDecimal("2.236067977499789696");
BigDecimal r=new BigDecimal("2.236067977499789697");
BigDecimal mid;
BigDecimal t1=new BigDecimal(1);
BigDecimal t2=new BigDecimal(2);
while(l.multiply(l).subtract(d).abs().compareTo(eps)==1)
{
    mid=l.add(r).divide(t2);
    if(mid.multiply(mid).compareTo(d)<0)
    {
        l=mid;
    }
    else r=mid;
}
```

9.4 判大素数

```
BigInteger x;
x=cin.nextBigInteger();
if(x.isProbablePrime(1))
    System.out.println("Yes");
else
    System.out.println("No");
```

10 Python

10.1 计算表达式

```
#多组输入，读取到文件末尾EOF结束
while True:
    try:
        line = input()
```

```
    except EOFError:
        break
    z = eval(line) #将line转化为表达式类型并运算
    print(z)
```

10.2 正则表达式

```
>>> import re
>>> match = re.match(r"Hello(\s*)(.*)World!", "Hello                Python
   World!")
>>> match.groups()
('\t\t  ', 'Python ')

'''
模式 描述
^ 匹配字符串的开头
$ 匹配字符串的末尾。
. 匹配任意字符，除了换行符，当re.DOTALL标记被指定时，
   则可以匹配包括换行符的任意字符。
[...] 用来表示一组字符,单独列出：[amk] 匹配'a'，'m'或'k'
[^...] 不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。
re* 匹配0个或多个的表达式。
re+ 匹配1个或多个的表达式。
re? 匹配0个或1个由前面的正则表达式定义的片段，非贪婪方式
re{ n} 精确匹配 n 个前面表达式。例如， o{2} 不能匹配 "Bob"
   中的"o"，但是能匹配 "food" 中的两个 o。
re{ n,} 匹配 n 个前面表达式。例如， o{2,} 不能匹配"Bob"中的"o"，
但能匹配 "fooooood"中的所有 o。"o{1,}" 等价于 "o+"。"o{0,}"
则等价于 "o*"。
re{ n, m} 匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式a| b 匹配a或b
(re) 匹配括号内的表达式，也表示一个组
(?imx) 正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。
(?-imx) 正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。
(?: re) 类似 (...), 但是不表示一个组
(?imx: re) 在括号中使用i, m, 或 x 可选标志
(?-imx: re) 在括号中不使用i, m, 或 x 可选标志
(?#...) 注释.
(?= re) 前向肯定界定符。如果所含正则表达式，以 ...
   表示，在当前位置成功匹配时成功， 否则失败。但一旦所含表达式已经尝试
匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。
(?! re) 前向否定界定符。与肯定界定符相反；
   当所含表达式不能在字符串当前位置匹配时成功
(?> re) 匹配的独立模式，省去回溯。
```

```
\w 匹配字母数字及下划线
\W 匹配非字母数字及下划线
\s 匹配任意空白字符，等价于 [\t\n\r\f].
\S 匹配任意非空字符
\d 匹配任意数字，等价于 [0-9].
\D 匹配任意非数字
\A 匹配字符串开始
\Z 匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。
\z 匹配字符串结束
\G 匹配最后匹配完成的位置。
\b 匹配一个单词边界，也就是指单词和空格间的位置。例如，
'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B 匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配
"never" 中的 'er'。
\n, \t, 等. 匹配一个换行符。匹配一个制表符。等
\1...\9 匹配第n个分组的内容。
\10 匹配第n个分组的内容，如果它经匹配。否则指的是八进制字符码的表达式。


用户名  /^[a-z0-9_-]{3,16}$/
密码    /^[a-z0-9_-]{6,18}$/
十六进制值  /^#?([a-f0-9]{6}|[a-f0-9]{3})$/
电子邮箱    /^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
/^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/
URL /^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
IP 地址 /((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5] | [01]?\d\d?)/
/^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5] |
   2[0-4][0-9]|[01]?[0-9][0-9]?)$/
HTML 标签   /^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$/
删除代码\\注释  (?<!http:|\S)//.*$
Unicode编码中的汉字范围 /^[\u2E80-\u9FFF]+$/
'''
```

11 其它

11.1 快读

```
inline int read(){//如果是long long，这行和下面一行int改ll
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-')
            f=-1;
```

```
        ch=getchar();
    }
    while(ch>='0'&&ch<='9'){
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}//打死我都不用！！！！
```

11.2 int128

```
inline __int128 read()
{
    __int128 x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')
            f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=x*10+ch-'0';
        ch=getchar();
    }
    return x*f;
}

inline void write(__int128 x)
{
    if(x<0)
    {
        putchar('-');
        x=-x;
    }
    if(x>9)
        write(x/10);
    putchar(x%10+'0');
}
```

11.3 对拍

```cpp
//————————————data.cpp————————————
int main()
{
    freopen("in","w",stdout);
    srand(time(0));
    int n,m,q;
    n = rand()%100000;
    m = rand()%100000;
    q = rand()%100000;
    printf("%d %d %d\n",n,m,q);
    for(int i = 1;i <= q;i++){
        int a = rand()%n+1;
        int b = rand()%n+1;
        int c = rand()%2;
        printf("%d %d %d\n",a,b,c);
    }
    return 0;
}

//————————————1.cpp&&2.cpp————————————
int main()
{
    freopen("in","r",stdin);
    freopen("1.out","w",stdout);
    //freopen("2.out","w",stdout);
    .....
}

//————————————duipai.cpp————————————
int main()//Windows
{
    int cases = 0;
    do{
        if(cases) printf("#%d AC\n",cases);
        cases++;
        system("data.exe > data.txt");
        system("1.exe < data.txt > 1.txt");
        system("2.exe < data.txt > 2.txt");
    }while(!system("fc 1.txt 2.txt"));
    printf("#%d WA",cases);
    return 0;
}
int main()//Linux
{
```

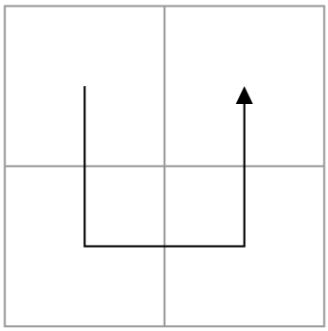
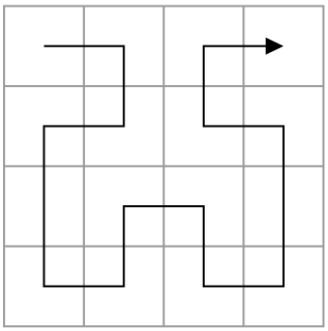
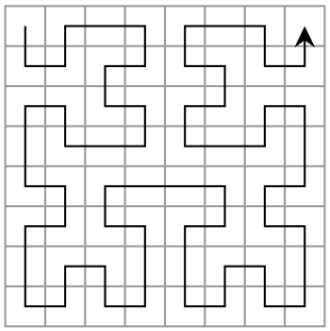
```
    int i;
    for (i=1;i<=1000;i++)
        {
            system("./data");
            system("./1");
            system("./2");
            printf("%d : ",i);
            if (system("diff 1.out 2.out"))
                {
                    printf("WA\n");
                    return 0;
                }
            else printf("AC\n");
        }
    return 0;
}
```

11.4 华容道

```
//判断是否有解
int map[16],ans=0;
for(int i=0;i<16;i++){
    scanf("%d",&map[i]);
    if(!map[i])
        ans+=6-i%4-i/4;
    for(int j=0;j<i;j++)
        if(map[j]>map[i])
            ans++;
}
if(ans&1)
    printf("Yes\n");
else
    printf("No\n");
```

11.5 希尔伯特曲线

```
#define ll long long
int two[50];//2的次方
ll f(int n, int x, int y) {//返回第几位
    if (n == 0) return 1;
    int m = two[n-1];//1 << (n - 1);//2的n-1次方
    if (x <= m && y <= m) {
```

```
        return f(n - 1, y, x);
    }
    if (x > m && y <= m) {
        return 3LL * m * m + f(n - 1, m-y+ 1, m * 2 - x + 1); // 3LL表示ll
            类型的3
    }
    if (x <= m && y > m) {
        return 1LL * m * m + f(n - 1, x, y - m);
    }
    if (x > m && y > m) {
        return 2LL * m * m + f(n - 1, x - m, y - m);
    }
}
const int SIZE=1e6+50;
struct node{                                    //用于存点
    int x,y;
    ll no;
}p[SIZE];
int main() {
    int n;int k;
    scanf("%d%d",&n,&k);
    two[0]=1;                                   //tow[1]=2;
    for(int i=1;i<=32;i++){
        two[i]=2*two[i-1];
    }
    for(int i=1;i<=n;i++){
        scanf("%d%d",&p[i].y,&p[i].x);          //注意y,x的读入顺序!
        p[i].no=f(k,p[i].x,p[i].y);             //用于存点的编号
    }
}
```

11.6 非整数希尔伯特曲线

```
const int SIZE=2e5+5;
```

```
struct node{
    string id;string name;
    bool operator < (const node &b) const {
        return id<b.id;
    }
}a[SIZE];
string f(double x,double y,double s,int dp){
    double mid = s/2;
    string ans;
    if(dp >= 1){
        if(x <= mid && y<= mid)ans = "1" + f(y,x,mid,dp-1);
        else if(x <= mid && y>mid)ans = "2" + f(x,y-mid,mid,dp-1);
        else if(x >mid && y>mid)ans="3" + f(x-mid,y-mid,mid,dp-1);
        else ans = "4" + f(mid-y,s-x,mid,dp-1);
    }
    return ans;
}
int main(){
    int n,k; //n个数，边长为k
    while(~scanf("%d %d",&n,&k)){
        int x,y;
        for(int i=1;i<=n;i++){
            string strtmp;
            cin>>x>>y>>strtmp;
            a[i].name=strtmp;
            a[i].id=f(x,y,k,30);
        }
    }
}
```

11.7 约瑟夫环

11.7.1 一般方法

```
/* *  n个  人(编号 1...n),先去掉第m个数,然后从m+1个开始报1, *
  报到k的退出,剩下的  人继续从1开始报数.求胜利利者的编号. */
int main(int argc, const char *argv[])
{
    int n, k, m;
    while (cin >> n >> k >> m, n || k || m)
    {
        int i, d, s = 0;
        for (i = 2; i <= n; i++)
        {
```

```
            s = (s + k) % i;
        }
        k = k % n;
        if (k == 0)
        {
            k = n;
        }
        d = (s + 1) + (m - k);
        if (d >= 1 && d <= n)
        {
            cout << d << '\n';
        }
        else if (d < 1)
        {
            cout << n + d << '\n';
        }
        else if (d > n)
        {
            cout << d % n << '\n';
        }
    }
    return 0;
}
```

11.7.2 函数图像解

```
/* * n 个  人数到 k 出列列，后剩下的人编号 */
unsigned long long n, k;
int main()
{
    cin >> n >> k;
    long long y = k % 2;
    long long x = 2, t = 0;
    long long z1 = y, z2 = x;
    while (x <= n)
    {
        z1 = y;
        z2 = x;
        t = (x - y) / (k - 1);
        if (t == 0)
        {
            t++;
        }
```

```
        y = y + t * k - ((y + t * k) / (x + t)) * (x + t);
        x += t;
    }
    cout << (z1 + (n - z2) * k) % n + 1 << endl;
    return 0;
}
```