Template

-zjhl2

AC 自动机

```
struct ACM
{
    int ch[N][26],fail[N],cnt[N];
    int tot;
    void init()
    {
        tot=0;
        for (int i=0;i<26;i++) ch[tot][i]=0;
        fail[tot]=0; cnt[tot]=0;
    }
    int newnode()
    {
        tot++;
        for (int i=0;i<26;i++) ch[tot][i]=0;
        fail[tot]=0;  cnt[tot]=0;
        return tot;
    }
    int insert(char *s)
    {
        int now=0;
        for (int i=0;s[i];i++)
        {
            int w=s[i]-'a';
            if (!ch[now][w]) ch[now][w]=newnode();
            now=ch[now][w];
        }
        cnt[now]=1;
        return now;
    }
    void build()
    {
        queue<int>Q;
        fail[0]=0;
        for (int i=0;i<26;i++)
        {
            if (!ch[0][i]) ch[0][i]=0;
            else
            {
                int cch=ch[0][i];
                fail[cch]=0;
```

```
                Q.push(cch);
            }
        }
        while(!Q.empty())
        {
            int u=Q.front(); Q.pop();
            for (int i=0;i<26;i++)
            {
                int v=ch[u][i];
                if (v)
                {
                    fail[v]=ch[fail[u]][i];
                    Q.push(v);
                }
                else ch[u][i]=ch[fail[u]][i];
            }
        }
    }
}AC;
```

Manacher

```
int p[N];
char s[N],c[N];
void manacher(int l)
{
    c[0]='('; c[1]='#';
    int len=1;
    for (int i=1;i<=l;i++) c[++len]=s[i],c[++len]='#';
    c[++len]=')';
    int mx=0,id=0;
    for (int i=1;i<len;i++)
    {
        if (i<=mx) p[i]=min(mx-i,p[id*2-i]);
        else p[i]=0;
        while(c[i+p[i]+1]==c[i-p[i]-1]) p[i]++;
        if (i+p[i]>mx) mx=i+p[i],id=i;
    }
}
```

后缀数组

```
//注意数组开多少
struct ST
{
    int f[N][20];
    void make(int *a,int n)
    {
        for (int i=1;i<=n;i++) f[i][0]=a[i];
        for (int j=1;j<20;j++)
            for (int i=1;i<=n+1-(1<<j);i++)
                f[i][j]=min(f[i][j-1],f[i+(1<<j-1)][j-1]);
    }
    int get(int l,int r)
    {
        int k=log(r-l+1)/log(2);
        return min(f[l][k],f[r+1-(1<<k)][k]);
    }
}H;
struct suffixarray
{
    int sa[N],rk[N],h[N];
    int cnt[N],sa2[N],rk2[N];
    int n;
    bool cmp(int i,int j,int len)
    {
        if (rk2[i]==rk2[j]&&rk2[i+len]==rk2[j+len]) return 0;
        return 1;
    }
    void make(int *s,int len)
    {
        s[len+1]=-1;  //按情况修改
        n=len;
        int m=9;   //按情况修改
        for (int i=0;i<=m;i++) cnt[i]=0;
        for (int i=1;i<=n;i++) cnt[s[i]]++;
        for (int i=1;i<=m;i++) cnt[i]+=cnt[i-1];
        for (int i=n;i>=1;i--) sa[cnt[s[i]]--]=i;
        int k=1; rk[sa[1]]=1;
        for (int i=2;i<=n;i++)
        {
            if (s[sa[i]]!=s[sa[i-1]]) k++;
```

```
            rk[sa[i]]=k;
        }
        rk2[n+1]=0;
        for (int j=1;k<n&&j<=n;j*=2)
        {
            int tot=0;
            for (int i=n-j+1;i<=n;i++) sa2[++tot]=i;
            for (int i=1;i<=n;i++)
                if (sa[i]>j) sa2[++tot]=sa[i]-j;
            m=k;
            for (int i=1;i<=m;i++) cnt[i]=0;
            for (int i=1;i<=n;i++) cnt[rk[i]]++;
            for (int i=2;i<=m;i++) cnt[i]+=cnt[i-1];
            for (int i=n;i>=1;i--) sa[ cnt[rk[sa2[i]]]-- ]=sa2[i];
            for (int i=1;i<=n;i++) rk2[i]=rk[i];
            k=1; rk[sa[1]]=1;
            for (int i=2;i<=n;i++)
            {
                if (cmp(sa[i],sa[i-1],j)) k++;
                rk[sa[i]]=k;
            }
        }
        for (int i=1;i<=n;i++)
        {
            if (rk[i]==1)
            {
                h[rk[i]]=0;  //这里很容易错
                continue;
            }
            k=h[rk[i-1]];
            if (k>0) k--;
            int j=sa[rk[i]-1];
            while(s[i+k]==s[j+k]) k++;
            h[rk[i]]=k;
        }
    }
    int lcp(ST &H,int i,int j)
    {
        int l=rk[i],r=rk[j];  //是否有 l==r
        if (l>r) swap(l,r);
        return H.get(l+1,r);
    }
}SA;
```

后缀自动机

```
#include<bits/stdc++.h>
using namespace std;
const int N=600005;
int tot,root,last;
int pa[N],deep[N],ch[N][27],cnt[N];
int newnode(int _deep)
{
    tot++;
    memset(ch[tot],0,sizeof(ch[tot]));
    deep[tot]=_deep;
    cnt[tot]=0;
    return tot;
}
void init()
{
    tot=0;
    root=newnode(0);
    last=root;
}
void insert(int w)
{
    int np=newnode(deep[last]+1);
    int u=last;
    while(u&&!ch[u][w]) ch[u][w]=np,u=pa[u];
    if (!u) pa[np]=root;
    else
    {
        int v=ch[u][w];
        if (deep[u]+1==deep[v]) pa[np]=v;
        else
        {
            int nv=newnode(deep[u]+1);
            memcpy(ch[nv],ch[v],sizeof(ch[v]));
            pa[nv]=pa[v]; pa[v]=pa[np]=nv;
            while(u&&ch[u][w]==v) ch[u][w]=nv,u=pa[u];
        }
    }
    last=np;
}
int sum[N],stk[N];
```

```
void topsort()
{
    for (int i=0;i<=deep[last];i++) sum[i]=0;
    for (int i=1;i<=tot;i++) sum[deep[i]]++;
    for (int i=1;i<=deep[last];i++) sum[i]+=sum[i-1];
    for (int i=1;i<=tot;i++) stk[sum[deep[i]]--]=i;
    for (int i=tot;i>=2;i--) cnt[pa[stk[i]]]+=cnt[stk[i]];
}
```

扩展 KMP

```
//ext1[i]表示 st[i...len]和 st[1...len]的公共前缀长度
//ext2[i]表示 s[i..len2]和 st[1...len]的公共前缀长度
int ext1[N];
void extend1(char *st,int len)
{
    ext1[1]=len;
    int j=0;
    while(j+2<=len&&st[2+j]==st[j+1]) j++;
    ext1[2]=j;
    int k=2;
    for (int i=3;i<=len;i++)
    {
        int p=k+ext1[k]-1;
        int j=ext1[i-k+1];
        if (i+j-1<p) ext1[i]=j;
        else
        {
            j=max(0,p-i+1);
            while(i+j<=len&&st[i+j]==st[j+1]) j++;
            ext1[i]=j;
            k=i;
        }
    }
}
int ext2[N];
void extend2(char *s,char *st,int len2,int len)
{
    int j=0;
    while(j+1<=len2&&j+1<=len&&s[j+1]==st[j+1]) j++;
    ext2[1]=j;
    int k=1;
    for (int i=2;i<=len2;i++)
```

```
    {
        int p=k+ext2[k]-1;
        int j=ext1[i-k+1];
        if (i+j-1<p) ext2[i]=j;
        else
        {
            j=max(0,p-i+1);
            while(i+j<=len2&&j+1<=len&&s[i+j]==st[j+1]) j++;
            ext2[i]=j;
            k=i;
        }
    }
}
```

最小表示法

```
char s[20005];
int mcp(char *s)
{
    int len=strlen(s);
    for (int i=len;i<2*len;i++) s[i]=s[i-len];
    int i=0,j=1;
    while(i<len&&j<len)
    {
        int p=0;
        while(p<len&&s[i+p]==s[j+p]) p++;
        if (p==len) break;
        if (s[i+p]<s[j+p]) j=j+p+1;
        else i=i+p+1;
        if (i==j) j++;
    }
    return i<j?i:j;
}
int main()
{
    int t; scanf("%d",&t);
    while(t--)
    {
        scanf("%s",s);
        printf("%d\n",mcp(s)+1);
    }
}
```

O(n)逆元

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

扩展欧几里得

```
int egcd(int a,int b,int &x,int &y)
{
    if (b==0){x=1; y=0; return a;}
    int tmp=egcd(b,a%b,y,x);
    y-=a/b*x;
    return tmp;
}
//ax+by=gcd(a,b)的最小解
//x+b/gcd*n 是通解
```

中国剩余定理

```
int p[N],r[N];
int CNA()
{
    int mo=p[1],re=r[1];
    for (int i=2;i<=n;i++)
    {
        //x*mo+re=y*p[i]+r[i];
        int x,y;
        int gcd=egcd(mo,p[i],x,y);
        if ((r[i]-re)%gcd!=0) return -1;
        x*=(r[i]-re)/gcd;
        x=(x%(p[i]/gcd)+(p[i]/gcd))%(p[i]/gcd);  // g
        re=x*mo+re;
        mo=mo/gcd*p[i];
    }
    return re;
}
```

矩阵乘法快速幂

```
#include<cstdio>
const int N=2,mo=10000;
struct mat{
    int c[N][N];
    void init()
    {
        for (int i=0;i<N;i++)
            for (int j=0;j<N;j++) c[i][j]=0;
    }
    mat operator*(mat b)
    {
        mat M; M.init();
        for (int i=0;i<N;i++)
            for (int j=0;j<N;j++)
                for (int k=0;k<N;k++)
                    M.c[i][j]=(1ll*c[i][k]*b.c[k][j]+M.c[i][j])%mo;
        return M;
    }
}B,A[31];
int n,i;
int main()
{
    A[0].init();
    A[0].c[0][1]=A[0].c[1][0]=A[0].c[1][1]=1;
    for (i=1;i<=30;i++) A[i]=A[i-1]*A[i-1];
    while(~scanf("%d",&n)&&n!=-1)
    {
        if (n==0) printf("0\n");
        else
        {
            n--;
            B.init();
            for (i=0;i<N;i++) B.c[i][i]=1;
            for (i=0;i<=30&&n;i++,n>>=1)
                if (n&1) B=B*A[i];
            printf("%d\n",B.c[1][1]);
        }
    }
}
```

高斯消元

```
bool zero(double x)
{
    return x<eps&&x>-eps;
}
void gauss(int n,int m)
{
    for (int i=0;i<=n;i++)
    {
        int k;
        for (k=i;k<=n;k++)
        if (!zero(a[k][i])) break;
        if (k!=i)
            for (int j=0;j<=m;j++) swap(a[i][j],a[k][j]);
        double h=1.0/a[i][i];
        for (int j=i;j<=m;j++) a[i][j]*=h;
        for (k++;k<=n;k++)
        if (!zero(a[k][i]))
        {
            double h=a[k][i]/a[i][i];
            for (int j=i;j<=m;j++) a[k][j]-=a[i][j]*h;
        }
        for (k=0;k<i;k++)
        {
            double h=a[k][i]/a[i][i];
            for (int j=i;j<=m;j++) a[k][j]-=a[i][j]*h;
        }
    }
}
```

自适应辛普森

```
inline double F(double y)
{

}

inline double simpson(double a,double b){
    double c=a+(b-a)/2;
    return (F(a)+4*F(c)+F(b))*(b-a)/6;
```

```
}

inline double asr(double a,double b,double eps,double A){
    double c=a+(b-a)/2;
    double L=simpson(a,c),R=simpson(c,b);
    if(fabs(L+R-A)<=15*eps) return L+R+(L+R-A)/15.0;
    return asr(a,c,eps/2,L)+asr(c,b,eps/2,R);
}

inline double asr(double a,double b,double eps){
    return asr(a,b,eps,simpson(a,b));
}

cout<<asr(1,2,1e-8)<<endl; //1 到 2 关于 F 的积分，误差 1e-8
```

DLX

精确覆盖问题：给定一个由 0-1 组成的矩阵，是否能找到一个行的集合，使得集合中每一列都恰好包含一个 1

```
#include<cstdio>
const int N=100105;
struct DLX
{
    int L[N],R[N],U[N],D[N],row[N],col[N];
    int H[1005];
    int ansd,ans[1005];
    int s[1005];
    int tot,n,m;
    void init(int _n,int _m)
    {
        n=_n;  m=_m;
        tot=m;
        for (int i=0;i<=m;i++) L[i]=i-1,R[i]=i+1,U[i]=i,D[i]=i,s[i]=0;
        L[0]=m;  R[m]=0;
        for (int i=1;i<=n;i++) H[i]=-1;
    }
    void link(int i,int j)
    {
        s[j]++;
        ++tot;
        row[tot]=i;  col[tot]=j;

        U[tot]=U[j];  D[tot]=j;  D[U[j]]=tot;   U[j]=tot;
```

```
    if (H[i]<0)
    {
        H[i]=tot; L[tot]=tot; R[tot]=tot;
    }
    else
    {
        L[tot]=L[H[i]];  R[tot]=H[i]; R[L[H[i]]]=tot; L[H[i]]=tot;
    }
}
void remove(int c)
{
    R[L[c]]=R[c];  L[R[c]]=L[c];
    for (int i=D[c];i!=c;i=D[i])
        for (int j=R[i];j!=i;j=R[j])
        {
            s[col[j]]--;
            D[U[j]]=D[j];
            U[D[j]]=U[j];
        }
}
void resume(int c)
{
    R[L[c]]=c;  L[R[c]]=c;
    for (int i=U[c];i!=c;i=U[i])
        for (int j=L[i];j!=i;j=L[j])
        {
            s[col[j]]++;
            D[U[j]]=j;
            U[D[j]]=j;
        }
}
bool dfs(int d)
{
    if (R[0]==0)
    {
        ansd=d-1;
        return 1;
    }
    int c=R[0];
    for (int i=R[0];i!=0;i=R[i])
        if (s[i]<s[c]) c=i;
    remove(c);
    for (int i=D[c];i!=c;i=D[i])
    {
```

```
            ans[d]=row[i];
            for (int j=R[i];j!=i;j=R[j]) remove(col[j]);
            if (dfs(d+1)) return 1;
            for (int j=L[i];j!=i;j=L[j]) resume(col[j]);
        }
        resume(c);
        return 0;
    }
}g;
int main()
{
    int n,m,k,x;
    while(~scanf("%d%d",&n,&m))
    {
        g.init(n,m);
        for (int i=1;i<=n;i++)
        {
            scanf("%d",&k);
            for (int j=1;j<=k;j++) scanf("%d",&x),g.link(i,x);
        }
        if (g.dfs(1))
        {
            printf("%d",g.ansd);
            for (int i=1;i<=g.ansd;i++) printf(" %d",g.ans[i]);
            printf("\n");
        }
        else printf("NO\n");
    }
}
```

重复覆盖问题：给定一个由 0-1 组成的矩阵，是否能找到 k 行，使得集合中每一列都至少包含一个 1

```
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
typedef long long ll;
const int N=65,M=65,V=65*65;
int n,k;
struct DLX
{
    int L[V],R[V],U[V],D[V],row[V],col[V];
    int H[N];
    int s[M];
```

```
int tot,n,m;
bool vis[M];
void init(int _n,int _m)
{
    n=_n;  m=_m;
    tot=m;
    for (int i=0;i<=m;i++) L[i]=i-1,R[i]=i+1,U[i]=i,D[i]=i,s[i]=0;
    L[0]=m;  R[m]=0;
    for (int i=1;i<=n;i++) H[i]=-1;
}
void link(int i,int j)
{
    s[j]++;
    ++tot;
    row[tot]=i;  col[tot]=j;

    U[tot]=U[j];  D[tot]=j;  D[U[j]]=tot;   U[j]=tot;
    if (H[i]<0)
    {
        H[i]=tot; L[tot]=tot; R[tot]=tot;
    }
    else
    {
        L[tot]=L[H[i]];  R[tot]=H[i]; R[L[H[i]]]=tot; L[H[i]]=tot;
    }
}
void remove(int c)
{
    for (int i=D[c];i!=c;i=D[i])
        R[L[i]]=R[i],L[R[i]]=L[i];
}
void resume(int c)
{
    for (int i=U[c];i!=c;i=U[i])
        R[L[i]]=L[R[i]]=i;
}
int h()
{
    int ret=0;
    for (int c=R[0];c!=0;c=R[c])
    {
        if (D[c]==c) return n+1;
        vis[c]=0;
    }
```

```
        for (int c=R[0];c!=0;c=R[c])
            if (!vis[c])
            {
                ret++;
                vis[c]=1;
                for (int i=D[c];i!=c;i=D[i])
                    for (int j=R[i];j!=i;j=R[j])
                        vis[col[j]]=1;
            }
        return ret;
    }
    bool dfs(int d)
    {
        if (d+h()>k) return 0;
        if (R[0]==0) return 1;
        int c=R[0];
        for (int i=R[0];i!=0;i=R[i])
            if (s[i]<s[c]) c=i;
        for (int i=D[c];i!=c;i=D[i])
        {
            remove(i);
            for (int j=R[i];j!=i;j=R[j]) remove(j);
            if (dfs(d+1)) return 1;
            for (int j=L[i];j!=i;j=L[j]) resume(j);
              resume(i);
        }
        return 0;
    }
}g;
struct P
{
    int x,y;
    ll operator-(const P &t)
    {
        return 0ll+abs(x-t.x)+abs(y-t.y);
    }
}a[N];
bool check(ll len)
{
    g.init(n,n);
    for (int i=1;i<=n;i++)
        for (int j=1;j<=n;j++)
            if (a[i]-a[j]<=len)
                g.link(i,j);
```

```
    return g.dfs(0);
}
ll dis[N*N];
int main()
{
    int t; scanf("%d",&t);
    for (int cas=1;cas<=t;cas++)
    {
        scanf("%d%d",&n,&k);
        for (int i=1;i<=n;i++) scanf("%d%d",&a[i].x,&a[i].y);
        int cnt=0;
        for (int i=1;i<n;i++)
            for (int j=i+1;j<=n;j++) dis[++cnt]=a[i]-a[j];
        dis[++cnt]=0;
        sort(dis+1,dis+cnt+1);
        int l=1,r=cnt+1;
        while(l<r)
        {
            int mid=(l+r)/2;
            if (!check(dis[mid])) l=mid+1;else r=mid;
        }
        printf("Case #%d: %lld\n",cas,dis[l]);
    }
}
```

三维偏序

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=200005;
struct P
{
    int x,y,z;
}a[N];
bool cmpx(P a,P b)
{
    return a.x<b.x;
}
bool cmpy(int i,int j)
{
    return a[i].y<a[j].y;
}
```

```
int c[N];
void add(int x,int y)
{
    for (int i=x;i<N;i+=i&-i) c[i]+=y;
}
int get(int x)
{
    int ret=0;
    for (int i=x;i;i-=i&-i) ret+=c[i];
    return ret;
}
int f[N];
int nd1[N],nd2[N];
void solve(int l,int r)
{
    if (l==r) return;
    int mid=(l+r)/2;
    for (int i=l;i<=mid;i++) nd1[i]=i;
    for (int j=mid+1;j<=r;j++) nd2[j]=j;
    sort(nd1+l,nd1+mid+1,cmpy);
    sort(nd2+mid+1,nd2+r+1,cmpy);

    int i=l;
    for (int j=mid+1;j<=r;j++)
    {
        while(i<=mid&&a[nd1[i]].y<a[nd2[j]].y)
            add(a[nd1[i]].z,1),i++;
        f[nd2[j]]+=get(a[nd2[j]].z);
    }

    i=l;
    for (int j=mid+1;j<=r;j++)
    {
        while(i<=mid&&a[nd1[i]].y<a[nd2[j]].y)
            add(a[nd1[i]].z,-1),i++;
    }
    solve(l,mid);
    solve(mid+1,r);
}
int main()
{
    int n; scanf("%d",&n);
    for (int i=1;i<=n;i++) scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].z);
    sort(a+1,a+n+1,cmpx);
```

```
    solve(1,n);
    ll ans=1ll*n*(n-1)/2;
    for (int i=1;i<=n;i++) ans-=f[i];
    printf("%lld\n",ans);
}
```

线段树合并

```
int merge(int x,int y,int l,int r)
{
    if (x==0) return y;
    if (y==0) return x;
    if (l==r)
    {
        tree[x].mx+=tree[y].mx;
        return x;
    }
    int mid=(l+r)/2;
    tree[x].ls=merge(tree[x].ls,tree[y].ls,l,mid);
    tree[x].rs=merge(tree[x].rs,tree[y].rs,mid+1,r);
    up(x);
    return x;
}
```

矩形面积并

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=100005;
struct line
{
    int l,r,y,tp;
    bool operator<(const line &b)const
    {
        return y<b.y;
    }
}a[2*N];
int sum[2*N*4],cnt[2*N*4];
int mem[2*N];
void up(int i,int l,int r)
{
```

```
    if (cnt[i]) sum[i]=mem[r]-mem[l];
    else
    {
        if (l+1<r) sum[i]=sum[i*2]+sum[i*2+1];
        else sum[i]=0;
    }
}
void add(int i,int l,int r,int x,int y,int z)
{
    if (x<=l&&r<=y)
    {
        cnt[i]+=z;
        up(i,l,r);
        return;
    }
    int mid=(l+r)/2;
    if (x<mid) add(i*2,l,mid,x,y,z);
    if (y>mid) add(i*2+1,mid,r,x,y,z);
    up(i,l,r);
}
int main()
{
    int n; scanf("%d",&n);
    int tot=0;
    for (int i=1;i<=n;i++)
    {
        int x1,y1,x2,y2; scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
        a[++tot]={min(x1,x2),max(x1,x2)+1,min(y1,y2),1};
        mem[tot]=min(x1,x2);
        a[++tot]={min(x1,x2),max(x1,x2)+1,max(y1,y2)+1,-1};
        mem[tot]=max(x1,x2)+1;
    }
    sort(mem+1,mem+tot+1);
    sort(a+1,a+tot+1);
    ll ans=0;
    for (int i=1;i<=tot;i++)
    {
        int l=lower_bound(mem+1,mem+tot+1,a[i].l)-mem;
        int r=lower_bound(mem+1,mem+tot+1,a[i].r)-mem;
        ans+=1ll*sum[1]*(a[i].y-a[i-1].y);
        add(1,1,tot,l,r,a[i].tp);
    }
    printf("%lld\n",ans);
}//坐标线相交处为点坐标
```

Splay

```
const int N=300005;
int ch[N][2],fa[N],size[N],rev[N];
int root;
char s[10];
bool get(int x)
{
    return ch[fa[x]][1]==x;
}
void update(int x)
{
    size[x]=1;
    if (ch[x][0]) size[x]+=size[ch[x][0]];
    if (ch[x][1]) size[x]+=size[ch[x][1]];
}
void pushdown(int x)
{
    if (rev[x])
    {
        rev[x]=0;
        swap(ch[x][0],ch[x][1]);
        if (ch[x][0]) rev[ch[x][0]]^=1;
        if (ch[x][1]) rev[ch[x][1]]^=1;
    }
}
void rotate(int x)
{
    int old=fa[x],oldf=fa[old];
    int whichx=get(x);
    ch[old][whichx]=ch[x][whichx^1]; fa[ch[old][whichx]]=old;
    ch[x][whichx^1]=old; fa[old]=x;
    fa[x]=oldf;
    if (oldf)
        ch[oldf][ch[oldf][1]==old]=x;
    update(old); update(x);
}
int stk[N],top;
void splay(int x,int goal)
{
    top=0;
    for (int f=x;f!=goal;f=fa[f]) stk[++top]=f;
    for (int i=top;i>=1;i--) pushdown(stk[i]);
```

```
    for (int f;(f=fa[x])!=goal;rotate(x))
      if (fa[f]!=goal)
        rotate((get(x)==get(f))?f:x);
    if (goal==0) root=x;
}
int find(int k)
{
    int now=root;
    while(1)
    {
        pushdown(now);
        int tmp=size[ch[now][0]]+1;
        if (k<tmp) now=ch[now][0];
        if (k==tmp) return now;
        if (k>tmp) now=ch[now][1],k-=tmp;
    }
}
int nxt()
{
    int now=root;
    pushdown(now);
    now=ch[now][1];
    pushdown(now);
    while(ch[now][0]) now=ch[now][0],pushdown(now);
    return now;
}
```

最大团

```
#include<cstdio>
const int N=55;
int a[N][N];
int num[N],n,ans;
bool dfs(int *pre,int tot,int deep)
{
    if (tot==0)
    {
        if (deep>ans) return ans=deep,1;
        else return 0;
    }
    int now[N],tot2;
    for (int i=1;i<=tot;i++)
    {
```

```
        if (deep+tot-i+1<=ans) return 0;
        if (deep+num[pre[i]]<=ans) return 0;
        tot2=0;
        for (int j=i+1;j<=tot;j++)
            if (a[pre[i]][pre[j]]) now[++tot2]=pre[j];
        if (dfs(now,tot2,deep+1)) return 1;
    }
    return 0;
}
int maxclique()
{
    ans=0;
    for (int i=n;i;i--)
    {
        int wait[N],tot=0;
        for (int j=i+1;j<=n;j++)
            if (a[i][j]) wait[++tot]=j;
        dfs(wait,tot,1);
        num[i]=ans;
    }
    return ans;
}
int main()
{
    while(~scanf("%d",&n)&&n)
    {
        for (int i=1;i<=n;i++)
            for (int j=1;j<=n;j++) scanf("%d",&a[i][j]);
        printf("%d\n",maxclique());
    }
}
```

倍增 LCA

```
int lca(int x,int y)
{
    if (deep[x]<deep[y]) swap(x,y);
    if (deep[x]>deep[y])
    {
        for (int j=20;j>=0;j--)
            if (deep[fa[x][j]]>deep[y]) x=fa[x][j];
        x=fa[x][0];
    }
```

```
    if (x==y) return x;
    for (int j=20;j>=0;j--)
        if (fa[x][j]!=fa[y][j]) x=fa[x][j],y=fa[y][j];
    return fa[x][0];
}
int go(int x,int step)
{
    for (int i=20;i>=0;i--)
        if (step>=(1<<i)) x=fa[x][i],step-=(1<<i);
    return x;
}
for (int j=1;j<=20;j++)
    for (i=1;i<=n;i++) fa[i][j]=fa[fa[i][j-1]][j-1];
```

Tarjan 求强连通

```
bool vis[N],is[N];
void tarjan(int u)
{
    T++;
    dfn[u]=T; low[u]=T; stk[++top]=u; is[u]=true; vis[u]=true;
    for (int v:vec[u])
    {
        if (!vis[v])
        {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else
        if (is[v]) low[u]=min(low[u],dfn[v]);
    }
    if (low[u]==dfn[u])
    {
        cnt++;
        while(stk[top]!=u)
        {
            belong[stk[top]]=u;
            is[stk[top]]=0;
            top--;
        }
        belong[u]=u; is[u]=0; top--;
    }
}
```

二分图匹配

```
bool find(int u)
{
    for (int v:vec[u])
    {
        if (!vis[v])
        {
            vis[v]=1;
            if (g[v]==-1||find(g[v]))
            {
                g[v]=u;
                return 1;
            }
        }
    }
    return 0;
}
int HA()
{
    int all=0;
    memset(g,-1,sizeof(g));
    for (int i=1;i<=n;i++)
    {
        memset(vis,0,sizeof(vis));
        if (find(i)) all++;
    }
    return all;
}
/*
最大匹配数：最大匹配的匹配边的数目
最小点覆盖数：选取最少的点，使任意一条边至少有一个端点被选择
最大独立数：选取最多的点，使任意所选两点均不相连
最小路径覆盖数：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以为 0（即单个点）。
定理 1：最大匹配数 = 最小点覆盖数（这是 Konig 定理）
定理 2：最大匹配数 = 最大独立数
定理 3：最小路径覆盖数 = 顶点数 - 最大匹配数
*/
```

二分图边染色

```
const int N=1005;
int clr[N][N];
int ga[N][N],gb[N][N];
int cnt;
void dfs(int u,int p,int c1,int c2)
{
    if (u==0)
    {
        swap(gb[p][c1],gb[p][c2]);
        return;
    }
    int v=ga[u][c2];
    if (v) dfs(gb[v][c1],v,c1,c2);
    swap(ga[u][c1],ga[u][c2]);
    swap(gb[p][c1],gb[p][c2]);
    clr[u][v]=c1;
    clr[u][p]=c2;
}
int mx[N*100],my[N*100];
int main()
{
    int a,b,m; scanf("%d%d%d",&a,&b,&m);
    int ans=0;
    for (int i=1;i<=m;i++)
    {
        int x,y; scanf("%d%d",&x,&y);  mx[i]=x; my[i]=y;
        int c1=1;
        while(ga[x][c1]) c1++;
        int c2=1;
        while(gb[y][c2]) c2++;
        if (c1!=c2) dfs(gb[y][c1],y,c1,c2);
        ga[x][c1]=y; gb[y][c1]=x;
        clr[x][y]=c1;
        ans=max(ans,c1);
        ans=max(ans,c2);
    }
    printf("%d\n",ans);
    for (int i=1;i<=m;i++) printf("%d ",clr[mx[i]][my[i]]);
}
```

树链剖分

```
void dfs1(int u,int d,int p)
{
    fa[u]=p;
    deep[u]=d;
    son[u]=-1;
    size[u]=1;
    for (int now=base[u];now;now=pre[now])
    {
        int v=vec[now];
        if (v==p) continue;
        dfs1(v,d+1,u);
        size[u]+=size[v];
        if (son[u]==-1||size[v]>size[son[u]]) son[u]=v;
    }
}
void dfs2(int u,int p)
{
    top[u]=p;
    id[u]=++T;
    rd[T]=u;
    if (son[u]!=-1) dfs2(son[u],p);
    for (int now=base[u];now;now=pre[now])
    {
        int v=vec[now];
        if (v==fa[u]||v==son[u]) continue;
        dfs2(v,v);
    }
}
int lca(int x,int y)
{
    int f1=top[x],f2=top[y];
    while(f1!=f2)
    {
        if (deep[f1]<deep[f2]) swap(f1,f2),swap(x,y);
        x=fa[f1],f1=top[x];
    }
    if (deep[x]<deep[y]) swap(x,y);
    return y;
}
seg getseg(int x,int LCA,int v)
```

```
{
    seg tmp={-1,-1,-1,-1};
    while(x!=LCA&&deep[top[x]]>deep[LCA])
    {
        tmp=get(1,1,n,id[top[x]],id[x])+tmp;
        add(1,1,n,id[top[x]],id[x],v);
        x=fa[top[x]];
    }
    tmp=get(1,1,n,id[LCA],id[x])+tmp;
    add(1,1,n,id[LCA],id[x],v);
    return tmp;
}
```

树分治

```
int cent,mxsz;
int size[N];
void findcent(int u,int fa,int n)
{
    size[u]=1;
    int mx=0;
    for (P tmp:link[u])
    {
        int v=tmp.v;
        if (vis[v]||v==fa) continue;
        findcent(v,u,n);
        size[u]+=size[v];
        mx=max(mx,size[v]);
    }
    mx=max(mx,n-size[u]);
    if (mx<mxsz) cent=u,mxsz=mx;
}
void solve(int u,int n)
{
    cent=u;  mxsz=n;
    findcent(u,-1,n);
    u=cent;
    vis[u]=1;

    work(u);

    for (P tmp:link[u])
    {
```

```
        int v=tmp.v;
        if (vis[v]) continue;
        solve(v,size[v]);
    }
}
```

动态凸壳

```
#define X first
#define Y second
typedef long long ll;
typedef map<ll,ll>::iterator itr;
struct P
{
    ll x,y;
    P operator-(const P &t)const
    {
        return {x-t.x,y-t.y};
    }
    ll operator^(P t)
    {
        return x*t.y-y*t.x;
    }
};
ll dot(itr it,ll x,ll y)
{
    return it->X*x+it->Y*y;
}
ll cross(itr l,P p2,itr r)
{
    P p1={l->X,l->Y},p3={r->X,r->Y};
    return (p2-p1)^(p3-p2);
}
struct hull
{
    map<ll,ll>s;
    itr mid,l,r,p1,p2;
    void init() { s.clear(); }
    itr pre(itr it)
    {
        if (it==s.end()) return it;
        if (it==s.begin()) return s.end();
        return --it;
```

```
    }
    itr suc(itr it)
    {
        if (it==s.end()) return it;
        return ++it;
    }
    bool inside(P p)
    {
        if (s.empty()) return 0;
        r=s.lower_bound(p.x);
        if (r==s.end()) return 0;
        if (r->X==p.x) return p.y<=r->Y;
        if (r==s.begin()) return 0;
        l=r; l--;
        return cross(l,p,r)>=0;  //下凸壳修改不等号
    }
    void add(P p)
    {
        if (inside(p)) return;
        s[p.x]=p.y;
        mid=s.find(p.x);
        p1=suc(mid); p2=suc(p1);
       while(p1!=s.end()&&p2!=s.end()&&cross(mid,{p1->X,p1->Y},p2)>=0)
            s.erase(p1),p1=p2,p2=suc(p2);   //下凸壳修改不等号
        p1=pre(mid); p2=pre(p1);
       while(p1!=s.end()&&p2!=s.end()&&cross(mid,{p1->X,p1->Y},p2)<=0)
            s.erase(p1),p1=p2,p2=pre(p2);   //下凸壳修改不等号
    }
    ll get(ll x,ll y)
    {
        l=s.begin();
        r=suc(l);
        while(r!=s.end()&&dot(l,x,y)<=dot(r,x,y))
            s.erase(l),l=r,r=suc(r);
        return dot(l,x,y);
    }
}G;//上凸壳
```

Dinic

```
struct graph{
    int S,T;
    int base[mxN],vec[mxM],pre[mxM],tot;
    int c[mxM];
    int d[mxN],q[mxN];
    bool vis[mxN];
    void init()
    {
        memset(base,0,sizeof(base));
        tot=1;
    }
    void link(int x,int y,int z)
    {
        vec[++tot]=y; pre[tot]=base[x]; base[x]=tot; c[tot]=z;
        vec[++tot]=x; pre[tot]=base[y]; base[y]=tot; c[tot]=0;
    }
    bool bfs()
    {
        int head=0,tail=0;
        memset(d,-1,sizeof(d));
        d[S]=0;
        q[++tail]=S;
        while(head<tail)
        {
            head++;
            int u=q[head];
            for (int now=base[u];now;now=pre[now])
            {
                int v=vec[now];
                if (d[v]==-1&&c[now]>0)
                {
                    d[v]=d[u]+1;
                    q[++tail]=v;
                    if (v==T) return 1;
                }
            }
        }
        return 0;
    }
    int dfs(int u,int flow)
    {
```

```
        int r=0;
        if (u==T) return flow;
        for (int now=base[u];now&&r<flow;now=pre[now])
        {
            int v=vec[now];
            if (c[now]>0&&d[v]==d[u]+1)
            {
                int x=min(c[now],flow-r);
                x=dfs(v,x);
                r+=x;
                c[now]-=x;
                c[now^1]+=x;
            }
        }
        if (!r)d[u]=-2;
        return r;
    }
    int dinic()
    {
        int ans=0;
        while(bfs())
            ans+=dfs(S,INF);
        return ans;
    }
}G;
```

上下界最大最小流

```
int in[N];
int S,T,SS,TT;
int inflow;
bool pwork()
{
    G.init();
    memset(in,0,sizeof(in));
    for (int i=1;i<=k;i++)
        G.link(n+rd[i],rc[i],1);
    S=0,T=n+m+3;
    SS=T-2; TT=T-1;
    for (int i=1;i<=n;i++)
    {
        int l=(du[i]-dif+1)/2;
        int r=(du[i]+dif)/2;
```

```
        if (l<0) l=0;
        if (r>du[i]) r=du[i];
        if (l>r) return 0;
        G.link(i,TT,r-l);
        in[i]-=l;
        in[TT]+=l;
    }
    inflow=0;
    for (int i=1;i<T;i++)
    {
        if (in[i]>0) G.link(S,i,in[i]),inflow+=in[i];
        if (in[i]<0) G.link(i,T,-in[i]);
    }
    return 1;
}
//判断可行流
bool check(int dif)
{
    if (!pwork(dif)) return 0;
    G.S=S;  G.T=T;
    G.link(TT,SS,INF);
    int tmp=G.dinic();
    return tmp==inflow;
}
//求最小流
int getmin(int dif)
{
    if (!pwork(dif)) return 0;
    G.S=S;  G.T=T;
    G.dinic();
    G.link(TT,SS,INF);
    return G.dinic();
}
//求最大流
int getmax(int dif)
{
    if (!pwork(dif)) return 0;
    G.S=S;  G.T=T;
    G.link(TT,SS,INF);
    G.dinic();
    G.S=SS; G.T=TT;
    return G.dinic();
}
```

最小费用最大流

```
struct graph{
    int base[mxN],vec[2*mxM],pre[2*mxM],cost[2*mxM],flow[2*mxM],tot;
    int S,T;
    int mincost,maxflow;
    int que[mxN];
    int dis[mxN];
    int pv[mxN],pe[mxN];
    bool vis[N];
    void init()
    {
        tot=1;
        memset(base,0,sizeof(base));
    }
    void link(int x,int y,int f,int z)
    {
        vec[++tot]=y;  pre[tot]=base[x];  base[x]=tot;  cost[tot]=z;
        flow[tot]=f;
        vec[++tot]=x;  pre[tot]=base[y];  base[y]=tot;  cost[tot]=-z;
        flow[tot]=0;
    }
    bool spfa()
    {
        memset(vis,0,sizeof(vis));
        memset(pv,-1,sizeof(pv));
        for (int i=0;i<mxN;i++) dis[i]=INF;
        int head=0,tail=1;
        que[tail]=S;  vis[S]=1;  dis[S]=0;
        while(head!=tail)
        {
            head++;
            head%=mxN;
            int u=que[head];
            vis[u]=0;

            for (int now=base[u];now;now=pre[now])
            if (flow[now]>0)
            {
                int v=vec[now];
                if (dis[u]+cost[now]<dis[v])
                {
```

```
                dis[v]=dis[u]+cost[now];
                pv[v]=u;
                pe[v]=now;
                if (!vis[v])
                {
                    vis[v]=1;
                    tail++;
                    tail%=mxN;
                    que[tail]=v;
                }
            }
        }
    }
    if (dis[T]!=INF) return 1;
    else return 0;
}
void work()
{
    mincost=0;
    maxflow=0;
    while(spfa())
    {
        int mxf=INF;
        for (int v=T;v!=S;v=pv[v])
            mxf=min(mxf,flow[pe[v]]);
        maxflow+=mxf;
        mincost+=mxf*dis[T];
        for (int v=T;v!=S;v=pv[v])
            flow[pe[v]]-=mxf,flow[pe[v]^1]+=mxf;
    }
}
}G;
```

面向对象计算几何

```
#include<bits/stdc++.h>
using namespace std;
typedef long double lod; //long long 或者 long double
typedef long long ll;
typedef long double ld;
const ld eps=1e-8;
const ld pi=acos(-1.0);
```

```
int sgn(ld x)
{
    if (x<-eps) return -1;
    if (x>eps) return 1;
    return 0;
}
struct P; //点，向量
struct LINE; //线段，射线，直线;
struct CIRCLE;
struct TRIANGLE;
struct POLYGON;

void kr(ld &x)
{
    double t; scanf("%lf",&t);
    x=t;
}
void kr(ll &x)
{
    scanf("%lld",&x);
}
struct P
{
    lod x,y;
    void read()
    {
        kr(x); kr(y);
    }
    P operator+(const P &t)const
    {
        return {x+t.x,y+t.y};
    }
    P operator-(const P &t)const
    {
        return {x-t.x,y-t.y};
    }
    P operator*(ld t)const
    {
        return {x*t,y*t};
    }
    P operator/(ld t)const
    {
        return {x/t,y/t};
    }
```

```
    lod operator*(const P &t)const
    {
        return x*t.y-y*t.x;
    } //叉积
    lod operator%(const P &t)const
    {
        return x*t.x+y*t.y;
    } //点积
    bool operator<(const P &t)const
    {
        return sgn(x-t.x)<0||sgn(x-t.x)==0&&sgn(y-t.y)<0;
    }
    bool operator==(const P &t)const
    {
        return sgn(x-t.x)==0&&sgn(y-t.y)==0;
    }
    ld ang()const
    {
        return atan2(y,x);
    }
    ld length()const
    {
        return sqrt(x*x+y*y);
    }
    P rotate(const P &t,ld sita)const
    {
        return {(x-t.x)*cos(sita)-(y-t.y)*sin(sita)+t.x,
                (x-t.x)*sin(sita)+(y-t.y)*cos(sita)+t.y};
    } //逆时针转 sita
    ld btang(const P &t)const
    {
        return acos( (*this%t)/length()/t.length() );
    } //向量夹角
    P midvec(const P &t)const
    {
        return (*this)/length()+t/t.length();
    } //角平分向量
};

struct LINE
{
    P p1,p2;
    void read()
    {
```

```
    p1.read(); p2.read();
}
LINE midLINE()
{
    P midp=(p1+p2)/2;
    P v=p2-p1;
    v=v.rotate({0,0},pi/2);
    return {midp,midp+v};
} //中垂线
bool have1(const P &p)const
{
    return sgn( (p-p1)*(p-p2) )==0&&sgn( (p-p1)%(p-p2) )<=0;
} //线段上有点
bool have2(const P &p)const
{
    return sgn( (p-p1)*(p-p2) )==0&&sgn( (p-p1)%(p2-p1) )>=0;
} //射线上有点
bool have3(const P &p)const
{
    return sgn( (p-p1)*(p-p2) )==0;
} //直线上有点
lod areawith(const P &p)const
{
    return abs( (p1-p)*(p2-p)/2 );
} //线段和点围成面积
P vecfrom(const P &p)const
{
    P v=(p2-p1);
    v=v.rotate({0,0},pi/2);
    ld s1=(p1-p)*(p2-p);
    ld s2=v*(p2-p1);
    v=v*(s1/s2);
    return v;
}//点到直线垂足的向量
P footfrom(const P &p)const
{
    P v=vecfrom(p);
    return p+v;
} //点到直线垂足
ld dis1from(const P &p)const
{
    P foot=footfrom(p);
    if (have1(foot)) return (foot-p).length();
    return min( (p1-p).length(),(p2-p).length());
```

```
    }//点到线段距离
    ld dis2from(const P &p)const
    {
        P foot=footfrom(p);
        if (have2(foot)) return (foot-p).length();
        return (p1-p).length();
    }//点到射线距离
    ld dis3from(const P &p)const
    {
        return vecfrom(p).length();
    }//点到直线距离
    P symP(const P &p)const
    {
        P v=vecfrom(p);
        return p+v*2;
    } //点关于直线的对称点



    //1 线段 2 射线 3 直线
    bool isct11(const LINE &L)const
    {
        P a1=p1,a2=p2;
        P b1=L.p1,b2=L.p2;
        if (sgn( max(a1.x,a2.x)-min(b1.x,b2.x) )<0||
            sgn( max(b1.x,b2.x)-min(a1.x,a2.x) )<0||
            sgn( max(a1.y,a2.y)-min(b1.y,b2.y) )<0||
            sgn( max(b1.y,b2.y)-min(a1.y,a2.y) )<0)
                return 0;
        lod tmp1=(a2-a1)*(b1-a1);
        lod tmp2=(a2-a1)*(b2-a1);
        if (sgn(tmp1)<0&&sgn(tmp2)<0||sgn(tmp1)>0&&sgn(tmp2)>0) return
0;
        tmp1=(b2-b1)*(a1-b1);
        tmp2=(b2-b1)*(a2-b1);
        if (sgn(tmp1)<0&&sgn(tmp2)<0||sgn(tmp1)>0&&sgn(tmp2)>0) return
0;
        return 1;
    }
    bool isct21(const LINE &L)const
    {
        P v=p2-p1;
        P a=p1;
        P b1=L.p1,b2=L.p2;
```

```
        lod tmp1=v*(b1-a);
        lod tmp2=v*(b2-a);
        if (sgn(tmp1)<0&&sgn(tmp2)<0||sgn(tmp1)>0&&sgn(tmp2)>0) return
0;
        if (tmp1>tmp2) swap(b1,b2);
        if (sgn( (b1-a)*(b2-a) )>0) return 1;
        if (sgn( (b1-a)*(b2-a) )<0) return 0;
        //最后排除共线但不相交的情况
        return L.have1(a)||have2(b1)||have2(b2);
    }
    bool isct31(const LINE &L)const
    {
        P v=p2-p1;
        P a=p1;
        lod tmp1=v*(L.p1-a);
        lod tmp2=v*(L.p2-a);
        if (sgn(tmp1)<0&&sgn(tmp2)<0||sgn(tmp1)>0&&sgn(tmp2)>0) return
0;
        return 1;
    }
    bool isct22(const LINE &L)const
    {
        if (have2(L.p1)||L.have2(p1)) return 1;
        P v=vecfrom(L.p1);
        if (sgn( v%(L.p2-L.p1) )<=0) return 0;
        v=L.vecfrom(p1);
        if (sgn( v%(p2-p1) )<=0) return 0;
        return 1;
    }
    bool isct32(const LINE &L)const
    {
        if (have3(L.p1)) return 1;
        P v=vecfrom(L.p1);
        if (sgn( v%(L.p2-L.p1) )<=0) return 0;
        return 1;
    }
    bool isct33(const LINE &L)const
    {
        if (have3(L.p1)) return 1;
        return sgn( (p2-p1)*(L.p2-L.p1) )!=0;
    }

    //前提是不重合且有交点,p1 沿 p2-p1 方向到达 L 上的长度，负数表示反向
    //直线交多边形需要用到
```

```
    ld dis33(const LINE &L)const
    {
        return (L.p1-p1)*(L.p2-p1) / ( (p2-p1)*(L.p2-L.p1) )
               * (p2-p1).length();
    }
    P isctPoint(const LINE &L)const
    {
        ld len=dis33(L);
        P v=p2-p1;
        return p1+v*(len/v.length());
    }//直线交点坐标
};

struct CIRCLE
{
    P cent;
    lod r;
    void read()
    {
        cent.read(); kr(r);
    }
    ld area()const
    {
        return pi*r*r;
    }
    bool have(const P &p)const
    {
        return sgn( (p-cent).length()-r ) <=0;
    }//点在圆内
    P LeftcutPoint(const P &p)const
    {
        P v=p-cent;
        ld sita=acos(r/v.length());
        v=v.rotate({0,0},sita);
        v=v/v.length()*r;
        return cent+v;
    }//左切点
    P RightcutPoint(const P &p)const
    {
        P v=p-cent;
        ld sita=acos(r/v.length());
        v=v.rotate({0,0},-sita);
        v=v/v.length()*r;
        return cent+v;
```

```
}//右切点
bool isct3(const LINE &L)const
{
    return sgn(L.dis3from(cent)-r)<=0;
}//圆与直线相交
P vecto(const LINE &L)const
{
    P v=L.p2-L.p1;
    v=v.rotate({0,0},pi/2);
    if (sgn(v%(L.p1-cent))<0) v=v.rotate({0,0},pi);
    return v/v.length()*r;
}//从圆心垂直射向直线的向量，长度为 r
P LeftisctPoint(const LINE &L)const
{
    P v=vecto(L);
    ld d=L.dis3from(cent);
    ld sita=acos(d/r);
    return cent+v.rotate({0,0},sita);
}//左交点
P RightisctPoint(const LINE &L)const
{
    P v=vecto(L);
    ld d=L.dis3from(cent);
    ld sita=acos(d/r);
    return cent+v.rotate({0,0},-sita);
}//右交点
bool separate(const CIRCLE &C)const
{
    ld d=(cent-C.cent).length();
    return sgn(r+C.r-d)<=0;
}//相离
bool contain(const CIRCLE &C)const
{
    if (sgn(r-C.r)<0) return 0;
    ld d=(cent-C.cent).length();
    return sgn( d+C.r-r)<=0;
}//包含
ld isctarea(const CIRCLE &C)const
{
    if (separate(C)) return 0;
    if (contain(C)) return C.area();
    if (C.contain(*this)) return area();
    ld d=(cent-C.cent).length();
    ld ang1=acos( (r*r+d*d-C.r*C.r)/2/r/d );
```

```
        ld ang2=acos( (C.r*C.r+d*d-r*r)/2/C.r/d);
        return ang1*r*r+ang2*C.r*C.r
              -r*r*sin(2*ang1)/2-C.r*C.r*sin(2*ang2)/2;
    }//圆相交面积，分两个三角形减，否则被 codeforces 卡精度
};


struct TRIANGLE
{
    P a[3];
    void read()
    {
        for (int i=0;i<3;i++) a[i].read();
    }
    ld area()const
    {
        ld ret=0;
        for (int i=0;i<3;i++)
            ret+=a[i]*a[(i+1)];
        return abs(ret);
    }
    P center1()const
    {
        return (a[0]+a[1]+a[2])/3;
    }//重心
    P center2()const
    {
        LINE L1={a[0],a[1]};
        LINE L2={a[1],a[2]};
        return L1.midLINE().isctPoint(L2.midLINE());
    }//外心
    P center3()const
    {
        P v0=(a[1]-a[0]).midvec(a[2]-a[0]);
        P v1=(a[0]-a[1]).midvec(a[2]-a[1]);
        LINE L1={a[0],a[0]+v0};
        LINE L2={a[1],a[1]+v1};
        return L1.isctPoint(L2);
    }//内心
    bool have(const P &p)const
    {
        lod tmp1=0;
        for (int i=0;i<3;i++)
            tmp1+=a[i]*a[(i+1)%3];
```

```
    tmp1=abs(tmp1);
    lod tmp2=0;
    for (int i=0;i<3;i++)
        tmp2+=abs( (a[i]-p)*(a[(i+1)%2]-p) );
    return sgn(tmp1-tmp2)==0;
}//点在三角形内
bool isctLINE1(const LINE &L)const
{
    for (int i=0;i<3;i++)
    {
        LINE R={a[i],a[(i+1)%3]};
        if (L.isct11(R)) return 1;
    }
    return 0;
}//与线段相交
bool isctLINE2(const LINE &L)const
{
    for (int i=0;i<3;i++)
    {
        LINE R={a[i],a[(i+1)%3]};
        if (L.isct21(R)) return 1;
    }
    return 0;
}//与射线相交
bool isctLINE3(const LINE &L)const
{
    for (int i=0;i<3;i++)
    {
        LINE R={a[i],a[(i+1)%3]};
        if (L.isct31(R)) return 1;
    }
    return 0;
}//与直线相交
P FirstisctPoint(const LINE &L)const;//与有向直线第一个交点
P SecondisctPoint(const LINE &L)const;//与有向直线第二个交点

bool isct(const TRIANGLE &TRI)const
{
    for (int i=0;i<3;i++)
    {
        LINE L={a[i],a[(i+1)%3]};
        for (int j=0;j<3;j++)
        {
            LINE R={TRI.a[j],TRI.a[(j+1)%3]};
```

```
            if (L.isct11(R)) return 1;
         }
      }
      return 0;
   }//三角形相交
   bool contain(const TRIANGLE &TRI)const
   {
      for (int j=0;j<3;j++)
         if (!have(TRI.a[j])) return 0;
      return 1;
   }//当前三角形包含 TRI
   bool separate(const TRIANGLE &TRI)const
   {
      return !isct(TRI)&&!contain(TRI)&&!TRI.contain(*this);
   }//互相分离
   ld isctarea(const TRIANGLE &TRI)const;
};


const int POLNUM=1005;
struct PL
{
   ld len;
   int v;
}stk[POLNUM];
int top;
bool cmplen(const PL &a,const PL &b)
{
   return a.len<b.len;
}
P cent;
bool cmpang(const P &p1,const P &p2)
{
   int tmp=sgn( (p1-cent).ang() - (p2-cent).ang() );
   if (tmp!=0) return tmp<0;
   return (p1-cent).length() < (p2-cent).length();
}
struct POLYGON
{
   int n;
   P a[POLNUM];
   void read(int k)
   {
      for (int i=1;i<=k;i++) a[i].read();
```

```
    n=k;
}
void ChangetoConvex()
{
    for (int i=2;i<=n;i++)
        if (a[i].x<a[1].x||a[i].x==a[1].x&&a[i].y<a[1].y)
            swap(a[1],a[i]);
    cent=a[1];
    sort(a+2,a+n+1,cmpang);
    int top=2;
    for (int i=3;i<=n;i++)
    {
        while(top>=2&&
            sgn((a[top]-a[top-1])*(a[i]-a[top]))<=0)
                top--;
        a[++top]=a[i];
    }
    n=top;
}//变凸包!(逆时针)
ld Clength()const
{
    ld ret=0;
    for (int i=2;i<=n;i++) ret+=(a[i]-a[i-1]).length();
    if (n>2) ret+=(a[1]-a[n]).length(); //防止 n==2 重复计算
    return ret;
}//周长
bool have(const P p)
{
    int k,d1,d2,wn=0;
    a[0]=a[n];
    for (int i=1;i<=n;i++)
    {
        LINE L={a[i-1],a[i]};
        if (L.have1(p)) return 1;
        k=sgn( (a[i]-a[i-1])*(p-a[i-1]) );
        d1=sgn( a[i-1].y-p.y );
        d2=sgn( a[i].y-p.y );
        if (k>0&&d1<=0&&d2>0) wn++;
        if (k<0&&d2<=0&&d1>0) wn--;
    }
    return wn!=0;
}//点在多边形内
ld cutlength(const LINE &L)
{
```

```
        a[0]=a[n]; top=0;
        for (int i=1;i<=n;i++)
        {
            LINE R={a[i-1],a[i]};
            lod s1=sgn( (L.p2-L.p1)*(R.p1-L.p1) );
            lod s2=sgn( (L.p2-L.p1)*(R.p2-L.p1) );
            if (s1<0&&s2<0||s1==0&&s2==0||s1>0&&s2>0) continue;
            if (s1<s2) stk[++top]={L.dis33(R),(s1!=0&&s2!=0?2:1)};
            else stk[++top]={L.dis33(R),(s1!=0&&s2!=0?-2:-1)};
        }
        sort(stk+1,stk+top+1,cmplen);
        int cnt=0;
        ld ret=0;
        for (int i=1;i<=top;i++)
        {
            if (cnt) ret+=stk[i].len-stk[i-1].len;
            cnt+=stk[i].v;
        }
        return ret;
    }//直线和多边形的交线总长，两个多边形一顺一逆可求只被覆盖一次的长度，多个同方向可求直线与多个多边形的交线总长
    bool isct(const POLYGON &POL)const
    {
        for (int i=2;i<=n;i++)
            for (int j=2;j<=POL.n;j++)
            {
                LINE L1={a[i-1],a[i]};
                LINE L2={POL.a[j-1],POL.a[j]};
                if (L1.isct11(L2)) return 1;
            }
        return 0;
    }//多边形相交，当前是两链相交判断
};
////////////////////////////////////////
```

注意事项：

1.string.size()返回的是 unsigned int ，加减的时候如果有负数会出问题