模板

2014

homura

计算几何（二维）

浮点函数

```
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};
struct line{point a,b;};
```


叉乘

```
double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
    return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
```

点乘

```
double dmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
double dmult(double x1,double y1,double x2,double y2,double x0,double y0){
    return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}
```

三点共线

```
int dots_inline(point p1,point p2,point p3){
    return zero(xmult(p1,p2,p3));
}
int dots_inline(double x1,double y1,double x2,double y2,double x3,double y3){
    return zero(xmult(x1,y1,x2,y2,x3,y3));
}
```

点与线段位置关系

```
//判点是否在线段上,包括端点
int dot_online_in(point p,line l){
    return 
zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<
eps;
}
int dot_online_in(point p,point l1,point l2){
    return 
zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
int dot_online_in(double x,double y,double x1,double y1,double x2,double y2){
    return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)*(x2-x)<eps&&(y1-y)*(y2-y)<eps;
}

//判点是否在线段上,不包括端点
int dot_online_ex(point p,line l){
    return 
dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-l.b.x)||
!zero(p.y-l.b.y));
}
int dot_online_ex(point p,point l1,point l2){
    return 
dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-l2.x)|
|!zero(p.y-l2.y));
}
int dot_online_ex(double x,double y,double x1,double y1,double x2,double y2){
    return 
dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!ze
ro(y-y2));
}

//判两点在线段同侧,点在线段上返回0
int same_side(point p1,point p2,line l){
    return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
}
int same_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}

//判两点在线段异侧,点在线段上返回0
int opposite_side(point p1,point p2,line l){
    return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
```

```
}
int opposite_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
```

两直线关系

```
//判两直线平行
int parallel(line u,line v){
    return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b.y));
}
int parallel(point u1,point u2,point v1,point v2){
    return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
}

//判两直线垂直
int perpendicular(line u,line v){
    return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b.y));
}
int perpendicular(point u1,point u2,point v1,point v2){
    return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
}

//判两线段相交,包括端点和部分重合
int intersect_in(line u,line v){
    if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
        return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
    return
dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
    if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
        return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
    return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u2);
}

//判两线段相交,不包括端点和部分重合
int intersect_ex(line u,line v){
    return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
```

```
    return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}

//计算两直线交点,注意事先判断直线是否平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point intersection(line u,line v){
    point ret=u.a;
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
           /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    return ret;
}
point intersection(point u1,point u2,point v1,point v2){
    point ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
           /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    return ret;
}
```


点与线距离关系

```
//点到直线上的最近点
point ptoline(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    return intersection(p,t,l.a,l.b);
}
point ptoline(point p,point l1,point l2){
    point t=p;
    t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
    return intersection(p,t,l1,l2);
}

//点到直线距离
double disptoline(point p,line l){
    return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoline(point p,point l1,point l2){
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
double disptoline(double x,double y,double x1,double y1,double x2,double y2){
```

```
    return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);
}

//点到线段上的最近点
point ptoseg(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
        return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
    return intersection(p,t,l.a,l.b);
}
point ptoseg(point p,point l1,point l2){
    point t=p;
    t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
    if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
        return distance(p,l1)<distance(p,l2)?l1:l2;
    return intersection(p,t,l1,l2);
}

//点到线段距离
double disptoseg(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
        return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
    return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoseg(point p,point l1,point l2){
    point t=p;
    t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
    if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
        return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
```

矢量操作（旋转与缩放）

```
//矢量 V 以 P 为顶点逆时针旋转 angle 并放大 scale 倍
point rotate(point v,point p,double angle,double scale){
    point ret=p;
    v.x-=p.x,v.y-=p.y;
    p.x=scale*cos(angle);
    p.y=scale*sin(angle);
```

```
    ret.x+=v.x*p.x-v.y*p.y;
    ret.y+=v.x*p.y+v.y*p.x;
    return ret;
}
```

光学几何（对称点，平分线，反射线）

```
//p 点关于直线 L 的对称点
ponit symmetricalPointofLine(point p, line L)
{
    point p2;
    double d;
    d = L.a * L.a + L.b * L.b;
    p2.x = (L.b * L.b * p.x - L.a * L.a * p.x -
            2 * L.a * L.b * p.y - 2 * L.a * L.c) / d;
    p2.y = (L.a * L.a * p.y - L.b * L.b * p.y -
            2 * L.a * L.b * p.x - 2 * L.b * L.c) / d;
    return p2;
}

//求两点的平分线
line bisector(point& a, point& b) {
    line ab, ans;  ab.set(a, b);
    double midx = (a.x + b.x)/2.0,    midy = (a.y + b.y)/2.0;
    ans.a = -ab.b, ans.b = -ab.a, ans.c = -ab.b * midx + ab.a * midy;
    return ans;
}

// 已知入射线、镜面，求反射线。
// a1,b1,c1 为镜面直线方程(a1 x + b1 y + c1 = 0 ,下同)系数;
a2,b2,c2 为入射光直线方程系数;
a,b,c 为反射光直线方程系数.
// 光是有方向的，使用时注意：入射光向量:<-b2,a2>；反射光向量:<b,-a>.
// 不要忘记结果中可能会有"negative zeros"

void reflect(double a1,double b1,double c1,
double a2,double b2,double c2,
double &a,double &b,double &c)
{
    double n,m;
    double tpb,tpa;
    tpb=b1*b2+a1*a2;
    tpa=a2*b1-a1*b2;
    m=(tpb*b1+tpa*a1)/(b1*b1+a1*a1);
```

```
    n=(tpa*b1-tpb*a1)/(b1*b1+a1*a1);
    if(fabs(a1*b2-a2*b1)<1e-20)
    {
        a=a2;b=b2;c=c2;
        return;
    }
    double xx,yy; //(xx,yy)是入射线与镜面的交点。
    xx=(b1*c2-b2*c1)/(a1*b2-a2*b1);
    yy=(a2*c1-a1*c2)/(a1*b2-a2*b1);
    a=n;
    b=-m;
    c=m*yy-xx*n;
}
```

面积

```
#include <math.h>
struct point{double x,y;};

//计算cross product (P1-P0)x(P2-P0)
double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
    return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
```

三角形

```
//计算三角形面积,输入三顶点
double area_triangle(point p1,point p2,point p3){
    return fabs(xmult(p1,p2,p3))/2;
}
double area_triangle(double x1,double y1,double x2,double y2,double x3,double
y3){
    return fabs(xmult(x1,y1,x2,y2,x3,y3))/2;
}

//计算三角形面积,输入三边长
double area_triangle(double a,double b,double c){
    double s=(a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

多边形

```
//计算多边形面积,顶点按顺时针或逆时针给出
double area_polygon(int n,point* p){
    double s1=0,s2=0;
    int i;
    for (i=0;i<n;i++)
        s1+=p[(i+1)%n].y*p[i].x,s2+=p[(i+1)%n].y*p[(i+2)%n].x;
    return fabs(s1-s2)/2;
}
```

三角形

```
#include <math.h>
struct point{double x,y;};
struct line{point a,b;};

#include <math.h>
struct point{double x,y;};
struct line{point a,b;};

double distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

point intersection(line u,line v){
    point ret=u.a;
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
            /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    return ret;
}
```

四心一点（外内重垂心与费马点为）

```
//外心
point circumcenter(point a,point b,point c){
    line u,v;
    u.a.x=(a.x+b.x)/2;
    u.a.y=(a.y+b.y)/2;
    u.b.x=u.a.x-a.y+b.y;
```

```
    u.b.y=u.a.y+a.x-b.x;
    v.a.x=(a.x+c.x)/2;
    v.a.y=(a.y+c.y)/2;
    v.b.x=v.a.x-a.y+c.y;
    v.b.y=v.a.y+a.x-c.x;
    return intersection(u,v);
}
//内心
point incenter(point a,point b,point c){
    line u,v;
    double m,n;
    u.a=a;
    m=atan2(b.y-a.y,b.x-a.x);
    n=atan2(c.y-a.y,c.x-a.x);
    u.b.x=u.a.x+cos((m+n)/2);
    u.b.y=u.a.y+sin((m+n)/2);
    v.a=b;
    m=atan2(a.y-b.y,a.x-b.x);
    n=atan2(c.y-b.y,c.x-b.x);
    v.b.x=v.a.x+cos((m+n)/2);
    v.b.y=v.a.y+sin((m+n)/2);
    return intersection(u,v);
}
//垂心
point perpencenter(point a,point b,point c){
    line u,v;
    u.a=c;
    u.b.x=u.a.x-a.y+b.y;
    u.b.y=u.a.y+a.x-b.x;
    v.a=b;
    v.b.x=v.a.x-a.y+c.y;
    v.b.y=v.a.y+a.x-c.x;
    return intersection(u,v);
}

//重心
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点
point barycenter(point a,point b,point c){
    line u,v;
    u.a.x=(a.x+b.x)/2;
    u.a.y=(a.y+b.y)/2;
    u.b=c;
    v.a.x=(a.x+c.x)/2;
```

```
    v.a.y=(a.y+c.y)/2;
    v.b=b;
    return intersection(u,v);
}

//费马点
//到三角形三顶点距离之和最小的点
point fermentpoint(point a,point b,point c){
    point u,v;
    double step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);
    int i,j,k;
    u.x=(a.x+b.x+c.x)/3;
    u.y=(a.y+b.y+c.y)/3;
    while (step>1e-10)
        for (k=0;k<10;step/=2,k++)
            for (i=-1;i<=1;i++)
                for (j=-1;j<=1;j++){
                    v.x=u.x+step*i;
                    v.y=u.y+step*j;
                    if
(distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+distan
ce(v,c))
                        u=v;
                }
    return u;
}
```

曲率半径

```
//求曲率半径三角形内最大可围成面积
#include<iostream>
 #include<cmath>
 using namespace std;
 const double pi=3.14159265358979;
 int main()
 {
    double a,b,c,d,p,s,r,ans,R,x,l; int T=0;
    while(cin>>a>>b>>c>>d&&a+b+c+d)
     {
        T++;
        l=a+b+c;
        p=l/2;
        s=sqrt(p*(p-a)*(p-b)*(p-c));
        R= s /p;
```

```
        if (d >= l)  ans = s;
        else if(2*pi*R>=d) ans=d*d/(4*pi);
        else
        {
            r = (l-d)/((l/R)-(2*pi));
            x = r*r*s/(R*R);
            ans = s - x + pi * r * r;
        }
        printf("Case %d: %.2lf\n",T,ans);
     }
     return 0;
 }
```

多边形

```
#include <stdlib.h>
#include <math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};
double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
```

判定是否为凸多边形

```
//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
int is_convex(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[1]|s[2];
}

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
int is_convex_v2(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
```

```
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[0]&&s[1]|s[2];
}
```

点与多边形关系

```
//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出
int inside_convex(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[1]|s[2];
}

//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回 0
int inside_convex_v2(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[0]&&s[1]|s[2];
}

//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge 表示点在多边形边上时的返回值,offset 为多边形坐标上限
int inside_polygon(point q,int n,point* p,int on_edge=1){
    point q2;
    int i=0,count;
    while (i<n)
        for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
            if
(zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&(p[i].y
-q.y)*(p[(i+1)%n].y-q.y)<eps)
                return on_edge;
            else if (zero(xmult(q,q2,p[i])))
                break;
            else                                                              if
(xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])*xmult
(p[i],q2,p[(i+1)%n])<-eps)
                count++;
    return count&1;
}

inline int opposite_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
```

```
}

inline int dot_online_in(point p,point l1,point l2){
    return
zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回 1
int inside_polygon(point l1,point l2,int n,point* p){
    point t[MAXN],tt;
    int i,j,k=0;
    if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
        return 0;
    for (i=0;i<n;i++)
        if
(opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
            return 0;
        else if (dot_online_in(l1,p[i],p[(i+1)%n]))
            t[k++]=l1;
        else if (dot_online_in(l2,p[i],p[(i+1)%n]))
            t[k++]=l2;
        else if (dot_online_in(p[i],l1,l2))
            t[k++]=p[i];
    for (i=0;i<k;i++)
        for (j=i+1;j<k;j++){
            tt.x=(t[i].x+t[j].x)/2;
            tt.y=(t[i].y+t[j].y)/2;
            if (!inside_polygon(tt,n,p))
                return 0;
        }
    return 1;
}
```

重心

```
point barycenter(int n,point* p){
    point ret,t;
    double t1=0,t2;
    int i;
    ret.x=ret.y=0;
    for (i=1;i<n-1;i++)
        if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
            t=barycenter(p[0],p[i],p[i+1]);
            ret.x+=t.x*t2;
```

```
            ret.y+=t.y*t2;
            t1+=t2;
        }
    if (fabs(t1)>eps)
        ret.x/=t1,ret.y/=t1;
    return ret;
}
```

切割（可用于半平面交）

```
#define MAXN 100
#define eps 1e-8
#define zero(x)  (((x)>0?(x):-(x))<eps)
struct point{double x,y;};

double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

int same_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}

point intersection(point u1,point u2,point v1,point v2){
    point ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
            /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    return ret;
}

//将多边形沿 l1,l2 确定的直线切割在 side 侧切割,保证 l1,l2,side 不共线
void polygon_cut(int& n,point* p,point l1,point l2,point side){
    point pp[MAXN];
    int m=0,i;
    for (i=0;i<n;i++){
        if (same_side(p[i],side,l1,l2))
            pp[m++]=p[i];
        if
(!same_side(p[i],p[(i+1)%n],l1,l2)&&!(zero(xmult(p[i],l1,l2))&&zero(xmult(p[(
i+1)%n],l1,l2))))
            pp[m++]=intersection(p[i],p[(i+1)%n],l1,l2);
    }
```

```
    for (n=i=0;i<m;i++)
        if (!i||!zero(pp[i].x-pp[i-1].x)||!zero(pp[i].y-pp[i-1].y))
            p[n++]=pp[i];
    if (zero(p[n-1].x-p[0].x)&&zero(p[n-1].y-p[0].y))
        n--;
    if (n<3)
        n=0;
}
```


半平面交

```
/*
【题意】给出很多个半平面，这里每个半平面由线段组成，都是指向线段方向的左边表示有
(x1 - x) * (y2 - y) - (x2 - x) * (y1 - y) >=0 ( >=0 表示左边，<=0 表示右边)
要你求个半平面的核，就是所有半平面所围成的面积
【算法】O(nlogn)的半平面交算法，最后统计出得到的多边形的点，然后利用叉积公式求出面积就行了
*/
#include<cstdio>
#include<vector>
#include<cmath>
#include<algorithm>
using namespace std;
const double eps=1e-10,big=10000.0;
const int maxn = 20010;
struct point{ double x,y; };
struct polygon { //存放最后半平面交中相邻边的交点，就是一个多边形的所有点
    int n;
    point p[maxn];
};
struct line { //半平面，这里是线段
    point a,b;
};
double at2[maxn];
int ord[maxn],dq[maxn+1],lnum;

int n;
polygon pg;

line ls[maxn]; //半平面集合
inline int sig(double k) { //判是不是等于 0，返回-1，0，1，分别是小于，等于，大于
    return (k < -eps)? -1: k > eps;
}
//叉积>0 代表在左边，<0 代表在右边，=0 代表共线
//e 是否在 o->s 的左边 onleft(sig(multi))>=0
```

```
inline double multi(point o, point s, point e)
{//构造向量，然后返回叉积
     return (s.x-o.x)*(e.y-o.y)-(e.x-o.x)*(s.y-o.y);
}
//直线求交点
point isIntersected(point s1, point e1, point s2, point e2) {
    double dot1,dot2;
    point pp;
    dot1 = multi(s2,e1,s1); dot2 = multi(e1,e2,s1);
    pp.x = (s2.x * dot2 + e2.x * dot1) / (dot2 + dot1);
    pp.y = (s2.y * dot2 + e2.y * dot1) / (dot2 + dot1);
    return pp;
}
//象限排序
inline bool cmp(int u,int v) {
    if(sig(at2[u]-at2[v])==0)
        return sig(multi(ls[v].a,ls[v].b,ls[u].b))>=0;
    return at2[u]<at2[v];
}
//判断半平面的交点在当前半平面外
bool judgein(int x,int y,int z){
    point pnt = isIntersected(ls[x].a, ls[x].b, ls[y].a, ls[y].b); //求交点
   return  sig(multi(ls[z].a,ls[z].b,pnt))<0;  //判断交点位置,如果在右面,返回true,
如果要排除三点共线，改成<=
}
//半平面交
void HalfPlaneIntersection(polygon &pg) { //预处理
    int n = lnum , tmpn , i;
/*  对于atan2(y,x)
    结果为正表示从 X 轴逆时针旋转的角度，结果为负表示从 X 轴顺时针旋转的角度。
         atan2(a, b) 与 atan(a/b)稍有不同，atan2(a,b)的取值范围介于 -pi 到 pi 之间（不
包括 -pi），
        而atan(a/b)的取值范围介于-pi/2 到pi/2 之间（不包括±pi)*/
    for(i = 0 ; i < n ; i ++)
    { //atan2(y,x)求出每条线段对应坐标系的角度
        at2[i] = atan2( ls[i].b.y - ls[i].a.y, ls[i].b.x - ls[i].a.x);
        ord[i] = i;
    }
    sort(ord , ord + n , cmp);
    for (i = 1 , tmpn = 1 ; i < n ; i++) //处理重线的情况
        if( sig(at2[ord[i-1]] - at2[ord[i]]) != 0 ) ord[tmpn++] = ord[i];
    n = tmpn;
    //圈地
    int bot = 1,top = bot + 1; //双端栈，bot 为栈底，top 为栈顶
```

```
    dq[bot] = ord[0]; dq[top] = ord[1]; //先压两根线进栈
    for(i = 2 ; i < n ; i ++)
    {
      //bot < top 表示要保证栈里至少有 2 条线段，如果剩下 1 条，就不继续退栈
      //judgein，判断如果栈中两条线的交点如果在当前插入先的右边，就退栈
      while( bot < top && judgein(dq[top-1] , dq[top] , ord[i]) ) top--;
      //对栈顶要同样的操作
      while( bot < top && judgein(dq[bot+1] , dq[bot] , ord[i]) ) bot++;
      dq[++top] = ord[i];
    }
    //最后还要处理一下栈里面存在的栈顶的线在栈底交点末尾位置，或者栈顶在栈尾两条线的右边
    while( bot < top && judgein(dq[top-1] , dq[top] , dq[bot]) ) top--;
    while( bot < top && judgein(dq[bot+1] , dq[bot] , dq[top]) ) bot++;
    //最后一条线是重合的
    dq[--bot] = dq[top];
    //求多边形
    pg.n = 0;
    for(i = bot + 1;i <= top ; i++) //求相邻两条线的交点
        pg.p[pg.n++]      =      isIntersected(ls[dq[i-1]].a,      ls[dq[i-1]].b,
ls[dq[i]].a,ls[dq[i]].b);
}

inline void add(double a,double b,double c,double d)
{//添加线段
    ls[lnum].a.x = a; ls[lnum].a.y = b;
    ls[lnum].b.x = c; ls[lnum].b.y = d;
    lnum++;
}
int main() {
    int n,i;
    scanf("%d",&n);
    double a,b,c,d;
    for(i = 0 ;i < n ; i ++) {
        //输入代表一条向量(x = (c - a),y = (d - b));
        scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
        add(a,b,c,d);
    }
    //下面是构造一个大矩形边界
    add(0,0,big,0);//down
    add(big,0,big,big);//right
    add(big,big,0,big);//up
    add(0,big,0,0);//left
    HalfPlaneIntersection(pg); //求半平面交
    double area = 0;
```

```
    n = pg.n;
    ///最后多边形的各个点保存在 pg 里面
    for(i = 0 ; i < n ; i ++)
        area += pg.p[i].x * pg.p[(i+1)%n].y - pg.p[(i+1)%n].x * pg.p[i].y;//x1 *
y2 - x2 * y1 用叉积求多边形面积
    area=fabs(area)/2.0; //所有面积应该是三角形面积之和，而叉积求出来的是四边形的面积和，
所以要除 2

    printf("%.1f\n",area);
    return 0;
}
```

旋转卡壳与凸包

```
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn = 50005;
struct point {
    int x,y;
    bool operator<(const point a)const {
        return y < a.y ||(y == a.y && x < a.x);
    }
} PointSet[maxn],ch[maxn];
int multiply(point s,point e,point o) {
    return ((s.x-o.x)*(e.y-o.y)-(s.y-o.y)*(e.x-o.x));
}

int dist2(point a,point b) {
    return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
}

bool cmp(point a,point b) {
    if(multiply(a,b,PointSet[0]) == 0)
        return dist2(a,PointSet[0]) < dist2(PointSet[0],b);
    return multiply(a,b,PointSet[0]) > 0;
}

int rotating_calipers(point *ch,int n) {
    int q=1,ans=0;
    ch[n]=ch[0];
    for(int p=0; p<n; p++) {
        while(multiply(ch[p+1],ch[q+1],ch[p]) > multiply(ch[p+1],ch[q],ch[p]))
            q=(q+1)%n;
```

```
        ans=max(ans,max(dist2(ch[p],ch[q]),dist2(ch[p+1],ch[q+1])));
    }
    return ans;
}
void convex_hull(point *p,point *ch,int n,int &len) {
    sort(p, p+n);
    ch[0]=p[0];
    ch[1]=p[1];
    int top=1;
    for(int i=2; i<n; i++) {
        while(top>0 && multiply(ch[top],p[i],ch[top-1])<=0)
            top--;
        ch[++top]=p[i];
    }
    int tmp=top;
    for(int i=n-2; i>=0; i--) {
        while(top>tmp && multiply(ch[top],p[i],ch[top-1])<=0)
            top--;
        ch[++top]=p[i];
    }
    len=top;
}
int main() {
    int n,len;
    while(scanf("%d",&n)!=EOF) {
        memset(ch,0,sizeof(ch));
        for(int i=0; i<n; i++)
            scanf("%d%d",&PointSet[i].x,&PointSet[i].y);
        convex_hull(PointSet,ch,n,len);
        printf("%d\n",rotating_calipers(ch,len));
    }
    return 0;
}
```


判相似(poj 1931)

```
/*判断两个多边形是否相似(pku1931)
多边形相似充要条件:多边形 A 的任何两点之间长度,与多边形 B 对应的两个点
长度比值都相等注意一下顺时针和逆时针,还有数值的范围*/

#include <math.h>
#include <stdio.h>
```

```
#define eps 1e-6
#define sqr(x) ((x) * (x))
#define same(a, b) (fabs((a) - (b)) < eps)
#define dis2(a, b) (sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)))

struct point
{
    double x, y;
    point operator-(point &b)
    {
        point c;
        c.x = x - b.x;
        c.y = y - b.y;
        return c;
    }
};

double dot(point a, point b)
{
    return a.x * b.x + a.y * b.y;
}

double cross(point a, point b)
{
    return a.x * b.y - a.y * b.x;
}

double get_anlge(point p1, point p2, point p3)
{
    //不需要求反余弦,点击相当，反余弦相等,避免使用反三角函数
    return dot(p1 - p2, p3 - p2) / dis2(p1, p2) / dis2(p2, p3);
}

int get_dir(point p1, point p2, point p3)
{
    //根据三点得到转向
    double t1 = cross(p2 - p1, p3 - p2);
    if(fabs(t1) < eps) return 1;
    if(t1 < 0) return 2;
    else return 3;
}

int slove(double ang1[], double ang2[], double len1[], double len2[], int dir1[],
int dir2[], int n)
```

```
{
    /*由于题目已经告诉了对应点的匹配顺序，所以只需要从
    0，开始匹配，如果没有告诉对应的匹配顺序，就还有枚举
    匹配对应边*/
    //int k;
    /*for(int i = 0;i < n;i++)
    {
        for(int j = 0;j < n;j++)
        {
            //第 i 条边和第 j 条边相对应*/
            int s, t, k;
            s = 0;
            t = 0;
            for(k = 0;k < n;k++)
            {
                if(!same(ang1[s], ang2[t]) || !same(len1[s], len2[t]) || dir1[s] !=
dir2[t]) break;
                s++;
                t++;
                s %= n;
                t %= n;
            }
            if(k == n) return 1;
        /*}
    }*/
    return 0;
}

double polygonArea(point p[], int n)
{
    //已知多边形各顶点的坐标，求其面积
    double area = 0.0;
    for(int i = 1;i <= n;i++)
        area += (p[i - 1].x * p[i % n].y - p[i % n].x * p[i - 1].y);
    return area;
}

int main()
{
    int n, similar;
    point p1[20], p2[20];
    double max1, max2;
    double ang1[20], ang2[20], len1[20], len2[20];
    int dir1[20], dir2[20];
```

```
    while(scanf("%d", &n) && n)
    {
        for(int i = 0;i < n;i++)
        {
            scanf("%lf%lf", &p1[i].x, &p1[i].y);
        }
        for(int i = 0;i < n;i++)
        {
            scanf("%lf%lf", &p2[i].x, &p2[i].y);
        }
        ang1[0] = get_anlge(p1[n - 1], p1[0], p1[1]);
        ang2[0] = get_anlge(p2[n - 1], p2[0], p2[1]);
        dir1[0] = get_dir(p1[n - 1], p1[0], p1[1]);
        dir2[0] = get_dir(p2[n - 1], p2[0], p2[1]);
        for(int i = 1;i < n;i++)
        {
            ang1[i] = get_anlge(p1[i - 1], p1[i], p1[(i + 1) % n]);
            ang2[i] = get_anlge(p2[i - 1], p2[i], p2[(i + 1) % n]);
            dir1[i] = get_dir(p1[i - 1], p1[i], p1[(i + 1) % n]);
            dir2[i] = get_dir(p2[i - 1], p2[i], p2[(i + 1) % n]);
        }
        max1 = -1, max2 = -1;
        for(int i = 0;i < n;i++)
        {
            len1[i] = dis2(p1[i], p1[(i + 1) % n]);
            if(len1[i] > max1) max1 = len1[i];
            len2[i] = dis2(p2[i], p2[(i + 1) % n]);
            if(len2[i] > max2) max2 = len2[i];
        }
        for(int i = 0;i < n;i++)
        {
            len1[i] /= max1;
            len2[i] /= max2;
        }
        if(slove(ang1, ang2, len1, len2, dir1, dir2, n)) printf("similar\n");
        else printf("dissimilar\n");
    }
    return 0;
}
```

包含 点集的最小正方形个数

/*
想法都是把枚举数量减少下来，这里用到逼近迭代，就是先 m 分枚举范围，从中找到最小的，然后再在那

个区间 m 分，这样迭代下去。

```
*/
#include <iostream>
#include <math.h>
using namespace std;
const int MAXN = 305;
const double PI = asin(1.0) * 2;
const double EPS = 1e-6;
int n;
double x[MAXN];
double y[MAXN];
double xx[MAXN];
double yy[MAXN];
double check(double a) {
    double minx = 1000000;
    double maxx = -minx;
    double miny = minx;
    double maxy = maxx;
    for(int i = 0; i < n; i++) {
        xx[i] = cos(a) * x[i] + sin(a) * y[i];
        yy[i] = -sin(a) * x[i] + cos(a) * y[i];
        minx = min(minx, xx[i]);
        maxx = max(maxx, xx[i]);
        miny = min(miny, yy[i]);
        maxy = max(maxy, yy[i]);
    }
    double e = (maxx - minx) > (maxy - miny) ? (maxx - minx) : (maxy - miny);
    return e * e;
}
double go() {
    int m = 800;
    int times = 60;
    double begin = 0;
    double end = PI / 3;
    double res = -1;
    double from;
    double a;
    double da;
    while(times--) {
        a = begin;
        da = (end - begin) / m;
        for(int i = 0; i < m; i++)             {
            double t = check(a + da * i);
            if(res == -1 || t < res)                {
```

```
                res = t;
                from = a + da * i;
            }
        }
        begin = from - da;
        end = from + 2 * da;
    }
    return res;
}
int main() {
    int ncase;
    scanf("%d", &ncase);
    while(ncase--) {
        scanf("%d", &n);
        for(int i = 0; i < n; i++)
            scanf("%lf%lf", &x[i], &y[i]);
        printf("%.2lf\n", go());
    }
}
```


圆

```
#include <math.h>
#define eps 1e-8
struct point{double x,y;};

double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

double distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

double disptoline(point p,point l1,point l2){
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}

point intersection(point u1,point u2,point v1,point v2){
    point ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
            /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
```

```
    return ret;
}
```


点与圆关系

```
//计算圆上到点 p 最近点,如 p 与圆心重合,返回 p 本身
point dot_to_circle(point c,double r,point p){
    point u,v;
    if (distance(p,c)<eps)
        return p;
    u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
    u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
    v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
    v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
    return distance(u,p)<distance(v,p)?u:v;
}
```

线与圆关系

```
//判直线和圆相交,包括相切
int intersect_line_circle(point c,double r,point l1,point l2){
    return disptoline(c,l1,l2)<r+eps;
}

//判线段和圆相交,包括端点和相切
int intersect_seg_circle(point c,double r,point l1,point l2){
    double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
    point t=c;
    if (t1<eps||t2<eps)
        return t1>-eps||t2>-eps;
    t.x+=l1.y-l2.y;
    t.y+=l2.x-l1.x;
    return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
}
//计算直线与圆的交点,保证直线与圆有交点
//计算线段与圆的交点可用这个函数后判点是否在线段上
void  intersection_line_circle(point  c,double  r,point  l1,point  l2,point&
p1,point& p2){
    point p=c;
    double t;
    p.x+=l1.y-l2.y;
    p.y+=l2.x-l1.x;
```

```
    p=intersection(p,c,l1,l2);
    t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(l1,l2);
    p1.x=p.x+(l2.x-l1.x)*t;
    p1.y=p.y+(l2.y-l1.y)*t;
    p2.x=p.x-(l2.x-l1.x)*t;
    p2.y=p.y-(l2.y-l1.y)*t;
}
```


圆与圆关系

```
//判圆和圆相交,包括相切
int intersect_circle_circle(point c1,double r1,point c2,double r2){
    return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
}
//计算圆与圆的交点,保证圆与圆有交点,圆心不重合
void intersection_circle_circle(point c1,double r1,point c2,double r2,point&
p1,point& p2){
    point u,v;
    double t;
    t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(c1,c2))/2;
    u.x=c1.x+(c2.x-c1.x)*t;
    u.y=c1.y+(c2.y-c1.y)*t;
    v.x=u.x+c1.y-c2.y;
    v.y=u.y-c1.x+c2.x;
    intersection_line_circle(c1,r1,u,v,p1,p2);
}
```

切点

```
//将向量 p 逆时针旋转 angle 角度
Point Rotate(Point p,double angle) {
    Point res;
    res.x=p.x*cos(angle)-p.y*sin(angle);
    res.y=p.x*sin(angle)+p.y*cos(angle);
    return res;
}
//求圆外一点对圆(o,r)的两个切点 result1 和 result2
void TangentPoint_PC(Point poi,Point o,double r,Point &result1,Point &result2)
{
    double line=sqrt((poi.x-o.x)*(poi.x-o.x)+(poi.y-o.y)*(poi.y-o.y));
    double angle=acos(r/line);
    Point unitvector,lin;
    lin.x=poi.x-o.x;
```

```
    lin.y=poi.y-o.y;
    unitvector.x=lin.x/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
    unitvector.y=lin.y/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
    result1=Rotate(unitvector,-angle);
    result2=Rotate(unitvector,angle);
    result1.x+=o.x;
    result1.y+=o.y;
    result2.x+=o.x;
    result2.y+=o.y;
    return;
}
```

面积并与交

圆面积并（IOI2003）//代码长，慎

```
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;

//（坐标、半径为-10000..10000）
const int chash=12343;
const double zero=1e-8;
int n,nodes;
double pi;
const int nSIZE=1010;
double x[nSIZE],y[nSIZE],r[nSIZE];
double ans;
double inter[nSIZE][3];
int link[10010],next[10010];
double node[10010][3];
int first[chash];
void initial() {
    int i;
    pi=asin(1)*2;
    scanf("%d",&n);
    for (i=1; i<=n; i++)
        scanf("%lf %lf %lf",&x[i],&y[i],&r[i]);
    memset(first,0,sizeof(first));
    memset(next,0,sizeof(next));
}
double sqrdist(double x1,double y1,double x2,double y2) {
    return pow(x1-x2,2)+pow(y1-y2,2);
}
```

```
double dist(double x1,double y1,double x2,double y2) {
    return sqrt(pow(x1-x2,2)+pow(y1-y2,2));
}
void prepare() {
    int i,j;
    bool b[nSIZE];
    for (i=1; i<=n; i++)
        b[i]=true;
    for (i=1; i<=n; i++)
        for (j=i+1; j<=n; j++)
            if (x[i]==x[j] && y[i]==y[j] && r[i]==r[j]) {
                b[i]=false;
                break;
            }
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            if (i!=j && r[i]+zero<r[j]) {
                if (dist(x[i],y[i],x[j],y[j])<=r[j]-r[i]) {
                    b[i]=false;
                    break;
                }
            }
    j=0;
    for (i=1; i<=n; i++)
        if (b[i]) {
            j++;
            x[j]=x[i];
            y[j]=y[i];
            r[j]=r[i];
        }
    n=j;
}
double getangle(double x,double y) {
    if (x<-zero) return atan(y/x)+pi;
    else if (x>zero)
        if (y>0) return atan(y/x);
        else return atan(y/x)+pi*2;
    else if (y>0) return pi/2 ;
    else return pi*3/2;
}
void getcross(int i,int j,double &t1,double &t2) {
    double a,b,c,a1,b1,c1,x1,y1,x2,y2,t,l;
    a=(x[i]-x[j])*2;
    b=(y[i]-y[j])*2;
```

```
    c=pow(r[j],2)-pow(r[i],2)+pow(x[i],2)-pow(x[j],2)+pow(y[i],2)-pow(y[j],2);
    a1=b;
    b1=-a;
    c1=a1*x[i]+b1*y[i];
    t=a*b1-b*a1;
    x1=(c*b1-b*c1)/t;
    y1=(a*c1-c*a1)/t;
    l=sqrt(pow(r[i],2)-pow(x1-x[i],2)-pow(y1-y[i],2));
    t=sqrt(pow(a1,2)+pow(b1,2));
    x2=x1+l*a1/t;
    y2=y1+l*b1/t;
    x1=x1*2-x2;
    y1=y1*2-y2;
    t1=getangle(x1-x[i],y1-y[i]);
    t2=getangle(x2-x[i],y2-y[i]);
    if (t2<t1) t1=t1-pi*2;
    t=(t1+t2)/2;
    if (dist(x[j],y[j],x[i]+r[i]*cos(t),y[i]+r[i]*sin(t))>r[j]) {
        t=t1;
        t1=t2;
        t2=t;
        if (t2<zero) t2=t2+pi*2;
        else t1=t1-pi*2;
    }
}
void sort(int l,int r) {
    int i,j;
    double k1,k2;
    i=l;
    j=r;
    k1=inter[(l+r)>>1][1];
    k2=inter[(l+r)>>1][2];
    while (i<=j) {
        while   ((inter[i][1]+zero<k1)   ||   (fabs(inter[i][1]-k1)<zero)   &&
(inter[i][2]>k2+zero)) i++;
        while   ((inter[j][1]>k1+zero)   ||   (fabs(inter[j][1]-k1)<zero)   &&
(inter[j][2]+zero<k2)) j--;
        if (i<=j) {
            inter[0][1]=inter[i][1];
            inter[i][1]=inter[j][1];
            inter[j][1]=inter[0][1];
            inter[0][2]=inter[i][2];
            inter[i][2]=inter[j][2];
            inter[j][2]=inter[0][2];
```

```
            i++;
            j--;
        }
    }
    if (l<j) sort(l,j);
    if (i<r) sort(i,r);
}
int getwhere(double x,double y) {
    int i,t;
    t=((int)(fabs(x+y+zero)*100000))%chash;
    i=first[t];
    while (i!=0) {
        if ((fabs(node[i][1]-x)<zero)&&(fabs(node[i][2]-y)<zero)) {
            return i;
        }
        i=next[i];
    }
    nodes++;
    node[nodes][1]=x;
    node[nodes][2]=y;
    next[nodes]=first[t];
    first[t]=nodes;
    return nodes;
}
double getchord(double r,double a) {
    return pow(r,2)/2*(a-sin(a));
}
void getnode() {
    int i,j,k,top,t1,t2;
    nodes=0;
    for (i=1; i<=n; i++) {
        top=0;
        for (j=1; j<=n; j++)
            if ((i!=j) && (dist(x[i],y[i],x[j],y[j])+zero<r[i]+r[j])) {
                top++;
                getcross(i,j,inter[top][1],inter[top][2]);
            }
        if (top>0) {
            sort(1,top);
            k=0;
            for (j=1; j<=top; j++)
                if ((k==0)||(inter[j][1]>inter[k][2])) {
                    k++;
                    inter[k][1]=inter[j][1];
```

```
                    inter[k][2]=inter[j][2];
                } else if (inter[j][2]>inter[k][2]) inter[k][2]=inter[j][2];
            top=k;
            while
            ((top>0)&&(inter[top][2]+zero>inter[1][1]+pi*2)) {
                if (inter[top][1]-pi*2<inter[1][1])
                    inter[1][1]=inter[top][1]-pi*2;
                top--;
            }
            if (top>0) {
                for (j=1; j<=top-1; j++) {
                    ans=ans+getchord(r[i],inter[j+1][1]-inter[j][2]);

t1=getwhere(x[i]+r[i]*cos(inter[j+1][1]),y[i]+r[i]*sin(inter[j+1][1]));

t2=getwhere(x[i]+r[i]*cos(inter[j][2]),y[i]+r[i]*sin(inter[j][2]));
                    link[t1]=t2;
                }
                ans=ans+getchord(r[i],inter[1][1]+pi*2-inter[top][2]);

t1=getwhere(x[i]+r[i]*cos(inter[1][1]),y[i]+r[i]*sin(inter[1][1]));

t2=getwhere(x[i]+r[i]*cos(inter[top][2]),y[i]+r[i]*sin(inter[top][2]));
                link[t1]=t2;
            }
        } else ans=ans+pi*r[i]*r[i];
    }
}
void work() {
    int i,j;
    bool visited[10010];
    ans=0;
    getnode();
    memset(visited,0,sizeof (visited));
    for (i=1; i<=nodes; i++)
        if (!visited[i]) {
            j=i;
            do {
                visited[j]=true;

ans=ans+(node[link[j]][1]*node[j][2]-node[j][1]*node[link[j]][2])/2;
                j=link[j];
            } while (j!=i);
        }
```

```
    printf("%.6lf\n",ans);
}
int main() {

    initial();
    prepare();
    work();
    return 0;
}
```

求圆和多边形的面积交

```
// 求圆和多边形的交
/*圆和简单多边形*/
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;
#define M 30
#define eps 1e-7
const double PI = acos(-1.0);

class pnt_type {
public:
    double x,y;
};
class state_type {
public:
    double angle;
    double CoverArea;
};

pnt_type pnt[M];
pnt_type center;

int n;
double R;

bool read_data() {
    n = 3;
    int i;
```

```
    if (cin >> pnt[1].x >> pnt[1].y) {
        for (i=2; i<=n; i++) cin >> pnt[i].x >> pnt[i].y;
        cin >> center.x >> center.y >> R;
        return true;
    }
    return false;
}
inline double Area2(pnt_type &a,pnt_type &b,pnt_type &c) {
    return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
}
inline double dot(pnt_type &a,pnt_type &b,pnt_type &c) {
    return (b.x - a.x) * (c.x - a.x) + (b.y - a.y) * (c.y - a.y);
}
inline double dist(pnt_type &a,pnt_type &b) {
    return sqrt((b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y));
}

void init() {
    int i;
    double temp,sum;
    for (i=2; i<n; i++) {
        temp = Area2(pnt[1],pnt[i],pnt[i + 1]);
        sum += temp;
    }
    if (sum < 0) reverse(pnt + 1,pnt + n + 1);
    pnt[n + 1] = pnt[1];
}

inline bool inCircle(pnt_type &s) {
    return dist(center,s) <= R;
}

bool SameSide(pnt_type a,pnt_type b) {
    if (dist(a,center) > dist(b,center)) swap(a,b);
    return dot(a,b,center) < eps;
}

double ShadomOnCircle(pnt_type a,pnt_type b) {
    double flag = Area2(center,a,b),res = 0;
    if (fabs(flag) < eps) return 0;

    bool ina = inCircle(a),inb = inCircle(b);
    if (ina && inb) {
        res = fabs(Area2(center,a,b)) / 2;
```

```
    } else if (!ina && !inb) {
        if (SameSide(a,b)) {
            double theta = acos(dot(center,a,b) / dist(center,a) / dist(center,b));
            res = R * R * theta / 2;
        } else {
            double height = fabs(Area2(center,a,b)) / dist(a,b);
            double theta = acos(dot(center,a,b) / dist(center,a) / dist(center,b));
            if (height >= R) {
                res = R * R * theta / 2;
            } else {
                double _theta = 2 * acos(height / R);
                res = R * R * (theta - _theta) / 2 + R * R * sin(_theta) / 2;
            }
        }
    } else {
        if (!ina && inb) swap(a,b);
        double height = fabs(Area2(center,a,b)) / dist(a,b);
        double temp = dot(a,center,b);
        double     theta     =     acos(dot(center,a,b)     /     dist(center,a)     /
dist(center,b)),theta1,theta2;
        if (fabs(temp) < eps) {
            double _theta = acos(height / R);
            res += R * height / 2 * sin(_theta);
            res += R * R / 2 * (theta - _theta);
        } else {
            theta1 = asin(height / R);
            theta2 = asin(height / dist(a,center));
            if (temp > 0) {
                res += dist(center,a) * R / 2 * sin(PI - theta1 - theta2);
                res += R * R / 2 * (theta + theta1 + theta2 - PI);
            } else {
                res += dist(center,a) * R / 2 * sin(theta2 - theta1);
                res += R * R / 2 * (theta - theta2 + theta1);
            }
        }
    }
    if (flag < 0) return -res;
    else return res;
}

double Cover() {
    int i;
    double res = 0;
    for (i=1; i<=n; i++)
```

```
        res += ShadomOnCircle(pnt[i],pnt[i + 1]);
    return res;
}

int main() {
    double ans;
    while (read_data()) {
        init();
        ans = Cover();
        printf("%.2lf\n",ans);
    }
    return 0;
}

/*两圆相交*/
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
const double eps = 1e-8;
const double pi = acos(-1.0);
struct Point {
    double x, y;
    Point operator - (const Point& t) const {
        Point tmp;
        tmp.x = x - t.x;
        tmp.y = y - t.y;
        return tmp;
    }
    Point operator + (const Point& t) const {
        Point tmp;
        tmp.x = x + t.x;
        tmp.y = y + t.y;
        return tmp;
    }
    bool operator == (const Point& t) const {
        return fabs(x-t.x) < eps && fabs(y-t.y) < eps;
    }
} GP;

double cir_area_inst(Point c1, double r1, Point c2, double r2) {            // 两
圆面积交
    double a1, a2, d, ret;
```

```
    d = sqrt((c1.x-c2.x)*(c1.x-c2.x)+(c1.y-c2.y)*(c1.y-c2.y));
    if ( d > r1 + r2 - eps )
        return 0;
    if ( d < r2 - r1 + eps )
        return pi*r1*r1;
    if ( d < r1 - r2 + eps )
        return pi*r2*r2;
    a1 = acos((r1*r1+d*d-r2*r2)/2/r1/d);
    a2 = acos((r2*r2+d*d-r1*r1)/2/r2/d);
    ret = (a1-0.5*sin(2*a1))*r1*r1 + (a2-0.5*sin(2*a2))*r2*r2;
    return ret;
}
int main() {
    Point a,b;
    double r1,r2;
    scanf("%lf%lf%lf%lf%lf%lf",&a.x,&a.y,&r1,&b.x,&b.y,&r2);
    double area = cir_area_inst(a,r1,b,r2);
    printf("%.3lf\n",area);
    return 0;
}
```

球面

```
#include <math.h>
const double pi=acos(-1);
```

圆心角

```
//计算圆心角 lat 表示纬度,-90<=w<=90,lng 表示经度
//返回两点所在大圆劣弧对应圆心角,0<=angle<=pi
double angle(double lng1,double lat1,double lng2,double lat2){
    double dlng=fabs(lng1-lng2)*pi/180;
    while (dlng>=pi+pi)
        dlng-=pi+pi;
    if (dlng>pi)
        dlng=pi+pi-dlng;
    lat1*=pi/180,lat2*=pi/180;
    return acos(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2));
}
```

距离

```
//计算直线距离,r 为球半径
double line_dist(double r,double lng1,double lat1,double lng2,double lat2){
    double dlng=fabs(lng1-lng2)*pi/180;
    while (dlng>=pi+pi)
        dlng-=pi+pi;
    if (dlng>pi)
        dlng=pi+pi-dlng;
    lat1*=pi/180,lat2*=pi/180;
    return r*sqrt(2-2*(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2)));
}

//计算球面距离,r 为球半径
inline double sphere_dist(double r,double lng1,double lat1,double lng2,double
lat2){
    return r*angle(lng1,lat1,lng2,lat2);
}
```

光学几何

```
//球面反射
//SGU110
// http://acm.sgu.ru/problem.php?contest=0&problem=110

#include <cstdio>
#include <cmath>

const int size = 555;
const double eps = 1e-9;

struct point {double x, y, z;} centre = {0, 0, 0};
struct circle {point o; double r;} cir[size];
struct ray {point s, dir;} l;
int n;

int dcmp (double x){return x < -eps ? -1 : x > eps;}
double sqr (double x){return x*x;}
double dot (point a, point b){return a.x * b.x + a.y * b.y + a.z * b.z;}
double dis2 (point a, point b){return sqr(a.x-b.x) + sqr(a.y-b.y) + sqr(a.z-b.z);}
double disToLine2 (point a, ray l){   /**** 点到直线 L 的距离的平方 **/
    point tmp;
    tmp.x =  l.dir.y * (a.z - l.s.z) - l.dir.z * (a.y - l.s.y);
```

```
    tmp.y = -l.dir.x * (a.z - l.s.z) + l.dir.z * (a.x - l.s.x);
    tmp.z =  l.dir.x * (a.y - l.s.y) - l.dir.y * (a.x - l.s.x);
    return dis2 (tmp, centre) / dis2 (l.dir, centre);
}

/**  用解方程(点到圆心的距离为 r)法求交点  (下面有向量法求交点, 两者取其一, 都 OK)*/
/* 是向量分量表示发的系数, 必须在射线上,故 K 必须为正, t 是交点***/
/*
bool find (circle p, ray l, double &k, point &t)
{
    double x = l.s.x - p.o.x, y = l.s.y - p.o.y, z = l.s.z - p.o.z;
    double a = sqr(l.dir.x) + sqr(l.dir.y) + sqr(l.dir.z);
    double b = 2 * (x*l.dir.x + y*l.dir.y + z*l.dir.z);
    double c = x*x + y*y + z*z - p.r*p.r;
    double det = b*b - 4*a*c;
//  printf ("a = %lf, b = %lf, c = %lf", a, b, c);
//  printf ("det = %lf\n", det);
    if (dcmp(det) == -1) return false;
    k = (-b - sqrt(det)) / a / 2;
    if (dcmp(k) != 1) return false;
    t.x = l.s.x + k * l.dir.x;
    t.y = l.s.y + k * l.dir.y;
    t.z = l.s.z + k * l.dir.z;
    return true;
}
*/

/**** 用向量法求交点  ***/
bool find (circle p, ray l, double &k, point &t)
{
    double h2 = disToLine2 (p.o, l);
//  printf ("h2 = %lf\n", h2);
    if (dcmp(p.r*p.r - h2) < 0) return false;
    point tmp;
    tmp.x = p.o.x - l.s.x;
    tmp.y = p.o.y - l.s.y;
    tmp.z = p.o.z - l.s.z;
    if (dcmp(dot(tmp, l.dir)) <= 0) return false;
    k = sqrt(dis2(p.o, l.s) - h2) - sqrt(p.r*p.r - h2);
    double k1 = k / sqrt(dis2(l.dir, centre));
    t.x = l.s.x + k1 * l.dir.x;
    t.y = l.s.y + k1 * l.dir.y;
    t.z = l.s.z + k1 * l.dir.z;
    return true;
```

```
}
/*计算新射线的起点和方向 */
void newRay (ray &l, ray l1, point inter)
{
    double k = - 2 * dot(l.dir, l1.dir);
    l.dir.x += l1.dir.x * k;
    l.dir.y += l1.dir.y * k;
    l.dir.z += l1.dir.z * k;
    l.s = inter;
}
/* 返回的是最先相交的球的编号,均不相交,返回-1 */
int update ()
{
    int sign = -1, i;
    double k = 1e100, tmp;
    point inter, t;
    for (i = 1; i <= n; i++){ //找到最先相交的球
        if (!find (cir[i], l, tmp, t)) continue;
        if (dcmp (tmp - k) < 0) k = tmp, inter = t, sign = i;
    }
    //ray 变向
    if (sign == -1) return sign;
    ray l1;
    l1.s = cir[sign].o;
    l1.dir.x = (inter.x - l1.s.x) / cir[sign].r;
    l1.dir.y = (inter.y - l1.s.y) / cir[sign].r;
    l1.dir.z = (inter.z - l1.s.z) / cir[sign].r;
    newRay (l, l1, inter);
    return sign;
}
int main ()
{
//  freopen ("in", "r", stdin);
    int i;
    scanf ("%d", &n);
    for (i = 1; i <= n; i++) //输入空间的球位置
        scanf ("%lf%lf%lf%lf", &cir[i].o.x, &cir[i].o.y, &cir[i].o.z, &cir[i].r);
    scanf ("%lf%lf%lf%lf%lf%lf", &l.s.x, &l.s.y, &l.s.z, &l.dir.x, &l.dir.y,
&l.dir.z);
    for (i = 0; i <= 10; i++){ //最多输出十次相交的球的编号
        int sign = update ();
        if (sign == -1) break;
        if (i == 0) printf ("%d", sign);
        else if (i < 10) printf (" %d", sign);
```

```
        else printf (" etc.");
    }
    puts ("");
}
```


网格

```
#define abs(x) ((x)>0?(x):-(x))
struct point{int x,y;};

int gcd(int a,int b){return b?gcd(b,a%b):a;}

//多边形上的网格点个数
int grid_onedge(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));
    return ret;
}

//多边形内的网格点个数
int grid_inside(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
    return (abs(ret)-grid_onedge(n,p))/2+1;
}
```

注意项

1. 注意舍入方式(0.5 的舍入方向);防止输出-0.
2. 整数几何注意 xmult 和 dmult 是否会出界;
   符点几何注意 eps 的使用.
3. 避免使用斜率;注意除数是否会为 0.
4. 公式一定要化简后再代入.
5. 判断同一个 2*PI 域内两角度差应该是
   abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;
   相等应该是
   abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;
6. 需要的话尽量使用 atan2,注意:atan2(0,0)=0,
   atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.
7. cross product = |u|*|v|*sin(a)
   dot product = |u|*|v|*cos(a)

8. (P1-P0)x(P2-P0)结果的意义:
   正: <P0,P1>在<P0,P2>顺时针(0,pi)内
   负: <P0,P1>在<P0,P2>逆时针(0,pi)内
   0 : <P0,P1>,<P0,P2>共线,夹角为 0 或 pi
9. 误差限缺省使用 1e-8!

公式类

三角形

1. 半周长 P=(a+b+c)/2
2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
   =4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
   =Ptan(A/2)tan(B/2)tan(C/2)
7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

四边形

D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角
1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2

(以下对圆的内接四边形)

3. ac+bd=D1D2
4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长

正 n 边形

R 为外接圆半径,r 为内切圆半径
1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

圆

1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

棱柱

1. 体积 V=Ah,A 为底面积,h 为高
2. 侧面积 S=lp,l 为棱长,p 为直截面周长
3. 全面积 T=S+2A

棱锥

1. 体积 V=Ah/3,A 为底面积,h 为高

(以下对正棱锥)

2. 侧面积 S=lp/2,l 为斜高,p 为底面周长
3. 全面积 T=S+A

棱台

1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高

(以下为正棱台)

2. 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
3. 全面积 T=S+A1+A2

圆柱

1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h

圆锥

1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3

圆台

1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

球

1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3

球扇形:
1. 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径
2. 体积 V=2PIr^2h/3

任意四面体

Euler 的任意四面体体积公式（已知边长求体积）

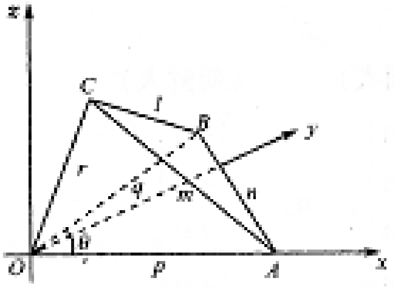


图 2.1 六条棱长已知的四面体

$$36V^2=\begin{vmatrix} p^2 & \frac{p^2+q^2-n^2}{2} & \frac{p^2+r^2-m^2}{2} \\ \frac{p^2+q^2-n^2}{2} & q^2 & \frac{q^2+r^2-l^2}{2} \\ \frac{p^2+r^2-m^2}{2} & \frac{q^2+r^2-l^2}{2} & r^2 \end{vmatrix}.$$

已知 4 点坐标求体积（其中四个点的坐标分别为（x1,y1,z1），（x2,y2,z2），（x3,y3,z3），（x4,y4,z4））

$$V=(1/6)*\begin{vmatrix} 1 & 1 & 1 & 1 \\ x1 & x2 & x3 & x4 \\ y1 & y2 & y3 & y4 \\ z1 & z2 & z3 & z4 \end{vmatrix}.$$

1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

计算几何（三维）

浮点函数

```
#include <cmath>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point3{double x,y,z;};
struct line3{point3 a,b;};
struct plane3{point3 a,b,c;}
```

叉乘

```
point3 xmult(point3 u,point3 v){
    point3 ret;
    ret.x=u.y*v.z-v.y*u.z;
    ret.y=u.z*v.x-u.x*v.z;
    ret.z=u.x*v.y-u.y*v.x;
    return ret;
}
```

点乘

```
//计算 dot product U . V
double dmult(point3 u,point3 v){
    return u.x*v.x+u.y*v.y+u.z*v.z;
}
```

矢量操作

```
//矢量差 U - V
point3 subt(point3 u,point3 v){
    point3 ret;
    ret.x=u.x-v.x;
    ret.y=u.y-v.y;
    ret.z=u.z-v.z;
    return ret;
}
//取平面法向量
point3 pvec(plane3 s){
    return xmult(subt(s.a,s.b),subt(s.b,s.c));
}
point3 pvec(point3 s1,point3 s2,point3 s3){
    return xmult(subt(s1,s2),subt(s2,s3));
}
//旋转
#include <iostream>
#include <cmath>
using namespace std;
const double PI = acos(-1.0);
const double eps = 1e-16;
struct point3 {
    double x,y,z;
    point3(double a,double b,double c) {
        x = a;
        y = b;
        z = c;
    }
    point3() {}
};
struct cube {
    point3 p[8];
} cb,cb1;
inline double zero(double a) {
    return fabs(a)<eps ? 0:a;
}
point3 rotate(double ux,double uy,double uz,double angle,point3 ori) {
    point3 p;
    p.x=(ux*ux+cos(angle)*(1-ux*ux))*ori.x+
        (ux*uy*(1-cos(angle))-uz*sin(angle))*ori.y+
        (uz*ux*(1-cos(angle))+uy*sin(angle))*ori.z;
    p.y=(ux*uy*(1-cos(angle))+uz*sin(angle))*ori.x+
```

```
        (uy*uy+cos(angle)*(1-uy*uy))*ori.y+
        (uy*uz*(1-cos(angle))-ux*sin(angle))*ori.z;
    p.z=(uz*ux*(1-cos(angle))-uy*sin(angle))*ori.x+
        (uy*uz*(1-cos(angle))+ux*sin(angle))*ori.y+
        (uz*uz+cos(angle)*(1-uz*uz))*ori.z;
    return p=point3(zero(p.x),zero(p.y),zero(p.z));
}
void getCube(point3 p0,double a) {
    cb.p[0]=p0;
    cb.p[1]=point3(p0.x,p0.y,p0.z+a);
    cb.p[2]=point3(p0.x,p0.y+a,p0.z);
    cb.p[3]=point3(p0.x+a,p0.y,p0.z);
    cb.p[4]=point3(p0.x+a,p0.y,p0.z+a);
    cb.p[5]=point3(p0.x,p0.y+a,p0.z+a);
    cb.p[6]=point3(p0.x+a,p0.y+a,p0.z);
    cb.p[7]=point3(p0.x+a,p0.y+a,p0.z+a);
}
int main() {
    int ncase;
    point3 us,ue,p0,vec;
    double  angle,a;
    scanf("%d",&ncase);
    while(ncase--) {
        scanf("%lf%lf%lf%lf",&a,&p0.x,&p0.y,&p0.z);
        scanf("%lf%lf%lf%lf%lf%lf",&us.x,&us.y,&us.z,&ue.x,&ue.y,&ue.z);
        scanf("%lf",&angle);
        angle=angle*PI/180;
        getCube(p0,a);
        double
tmp=sqrt((ue.x-us.x)*(ue.x-us.x)+(ue.y-us.y)*(ue.y-us.y)+(ue.z-us.z)*(ue.z-us
.z));
        vec.x=(ue.x-us.x)/tmp;
        vec.y=(ue.y-us.y)/tmp;
        vec.z=(ue.z-us.z)/tmp;
        for(int i=0; i<8; i++)
            cb1.p[i]=rotate(vec.x,vec.y,vec.z,angle,cb.p[i]);

        for(int i=0; i<8; i++)
            PointSet[i].x=cb1.p[i].x, PointSet[i].y=cb1.p[i].y;

        int vcnt;

        GS(PointSet,ch,8,vcnt);
```

```
        printf("%.2lf\n",area(vcnt,ch));
    }
    return 0;
}
```


三点共线与四点共面

```
//判三点共线
int dots_inline(point3 p1,point3 p2,point3 p3){
    return vlen(xmult(subt(p1,p2),subt(p2,p3)))<eps;
}

//判四点共面
int dots_onplane(point3 a,point3 b,point3 c,point3 d){
    return zero(dmult(pvec(a,b,c),subt(d,a)));
}
```

点与线关系

```
//判点是否在线段上,包括端点和共线
int dot_online_in(point3 p,line3 l){
    return
zero(vlen(xmult(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&
        (l.a.y-p.y)*(l.b.y-p.y)<eps&&(l.a.z-p.z)*(l.b.z-p.z)<eps;
}
int dot_online_in(point3 p,point3 l1,point3 l2){
    return
zero(vlen(xmult(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-p.x)<eps&&
        (l1.y-p.y)*(l2.y-p.y)<eps&&(l1.z-p.z)*(l2.z-p.z)<eps;
}

//判点是否在线段上,不包括端点
int dot_online_ex(point3 p,line3 l){
    return
dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y)||!zero(p.z-l.a.z))&&
        (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
}
int dot_online_ex(point3 p,point3 l1,point3 l2){
    return
dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y)||!zero(p.z-l1.z))&&
        (!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
}
```

```
//判两点在线段同侧,点在线段上返回 0,不共面无意义
int same_side(point3 p1,point3 p2,line3 l){
    return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))>ep
s;
}
int same_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))>eps;
}

//判两点在线段异侧,点在线段上返回 0,不共面无意义
int opposite_side(point3 p1,point3 p2,line3 l){
    return
dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))<-e
ps;
}
int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return
dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))<-eps;
}
```

点与面关系

```
//判点是否在空间三角形上,包括边界,三点共线无意义
int dot_inplane_in(point3 p,plane3 s){
    return
zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-vlen(xmult(subt(p,s.a),subt(p,s
.b)))-

    vlen(xmult(subt(p,s.b),subt(p,s.c)))-vlen(xmult(subt(p,s.c),subt(p,s.a))))
;
}
int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
    return
zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-vlen(xmult(subt(p,s1),subt(p,s2)))-

    vlen(xmult(subt(p,s2),subt(p,s3)))-vlen(xmult(subt(p,s3),subt(p,s1))));
}

//判点是否在空间三角形上,不包括边界,三点共线无意义
int dot_inplane_ex(point3 p,plane3 s){
    return dot_inplane_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>eps&&
```

```
    vlen(xmult(subt(p,s.b),subt(p,s.c)))>eps&&vlen(xmult(subt(p,s.c),subt(p,s.
a)))>eps;
}
int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
    return
dot_inplane_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>eps&&

    vlen(xmult(subt(p,s2),subt(p,s3)))>eps&&vlen(xmult(subt(p,s3),subt(p,s1)))
>eps;
}

//判两点在平面同侧,点在平面上返回 0
int same_side(point3 p1,point3 p2,plane3 s){
    return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))>eps;
}
int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))>eps;
}

//判两点在平面异侧,点在平面上返回 0
int opposite_side(point3 p1,point3 p2,plane3 s){
    return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))<-eps;
}
int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return
dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))<-eps;
}
```

线面关系（垂直平行相交交点）

```
//判两直线垂直
int perpendicular(line3 u,line3 v){
    return zero(dmult(subt(u.a,u.b),subt(v.a,v.b)));
}
int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){
    return zero(dmult(subt(u1,u2),subt(v1,v2)));
}

//判两平面垂直
int perpendicular(plane3 u,plane3 v){
    return zero(dmult(pvec(u),pvec(v)));
```

```
}
int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));
}

//判直线与平面平行
int perpendicular(line3 l,plane3 s){
    return vlen(xmult(subt(l.a,l.b),pvec(s)))<eps;
}
int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return vlen(xmult(subt(l1,l2),pvec(s1,s2,s3)))<eps;
}

//判两线段相交,包括端点和部分重合
int intersect_in(line3 u,line3 v){
    if (!dots_onplane(u.a,u.b,v.a,v.b))
        return 0;
    if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
        return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
    return
dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_
in(v.b,u);
}
int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2){
    if (!dots_onplane(u1,u2,v1,v2))
        return 0;
    if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
        return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
    return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||do
t_online_in(v2,u1,u2);
}

//判两线段相交,不包括端点和部分重合
int intersect_ex(line3 u,line3 v){
    return
dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.
b,u);
}
int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
    return
dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1
,u2);
}
```

```
//判线段与空间三角形相交,包括交于边界和(部分)包含
int intersect_in(line3 l,plane3 s){
    return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
        !same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
        !same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
}

//判线段与空间三角形相交,不包括交于边界和(部分)包含
int intersect_ex(line3 l,plane3 s){
    return opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&

    opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&
        opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
}

//计算两直线交点,注意事先判断直线是否共面和平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point3 intersection(line3 u,line3 v){
    point3 ret=u.a;
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
            /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    ret.z+=(u.b.z-u.a.z)*t;
    return ret;
}
point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
            /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    ret.z+=(u2.z-u1.z)*t;
    return ret;
}

//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
```

```
//线段和空间三角形交点请另外判断
point3 intersection(line3 l,plane3 s){
    point3 ret=pvec(s);
    double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-l.a.z))/
        (ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
    ret.x=l.a.x+(l.b.x-l.a.x)*t;
    ret.y=l.a.y+(l.b.y-l.a.y)*t;
    ret.z=l.a.z+(l.b.z-l.a.z)*t;
    return ret;
}
point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    point3 ret=pvec(s1,s2,s3);
    double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
        (ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
    ret.x=l1.x+(l2.x-l1.x)*t;
    ret.y=l1.y+(l2.y-l1.y)*t;
    ret.z=l1.z+(l2.z-l1.z)*t;
    return ret;
}

//计算两平面交线,注意事先判断是否平行,并保证三点不共线!
line3 intersection(plane3 u,plane3 v){
    line3 ret;
    ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):inte
rsection(v.a,v.b,u.a,u.b,u.c);
    ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):inte
rsection(v.c,v.a,u.a,u.b,u.c);
    return ret;
}
line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    line3 ret;
    ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v
1,v2,u1,u2,u3);
    ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v
3,v1,u1,u2,u3);
    return ret;
}
```

距离

```
//点到直线距离
double ptoline(point3 p,line3 l){
    return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
```

```
}
double ptoline(point3 p,point3 l1,point3 l2){
    return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
}

//点到平面距离
double ptoplane(point3 p,plane3 s){
    return fabs(dmult(pvec(s),subt(p,s.a)))/vlen(pvec(s));
}
double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){
    return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
}

//直线到直线距离
double linetoline(line3 u,line3 v){
    point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));
    return fabs(dmult(subt(u.a,v.a),n))/vlen(n);
}
double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 n=xmult(subt(u1,u2),subt(v1,v2));
    return fabs(dmult(subt(u1,v1),n))/vlen(n);
}
```

三角函数

```
//两直线夹角 cos 值
double angle_cos(line3 u,line3 v){
    return
dmult(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));
}
double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2){
    return dmult(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));
}

//两平面夹角 cos 值
double angle_cos(plane3 u,plane3 v){
    return dmult(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));
}
double angle_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return
dmult(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,v2,v3)
);
}
```

```
//直线平面夹角 sin 值
double angle_sin(line3 l,plane3 s){
    return dmult(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));
}
double angle_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return
dmult(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3));
}
```

三维凸包

面数

```
//凸包求面数
#include<stdio.h>
#include<math.h>
#include<algorithm>
#define eps 1e-7
#define MAXV 305
using namespace std;

struct pt {
    double x, y, z;
    pt() {}
    pt(double _x, double _y, double _z): x(_x), y(_y), z(_z) {}
    pt operator - (const pt p1) {
        return pt(x - p1.x, y - p1.y, z - p1.z);
    }
    pt operator * (pt p) {
        return pt(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);   //叉乘
    }
    double operator ^ (pt p) {
        return x*p.x+y*p.y+z*p.z;   //点乘
    }
} P[MAXV];

struct fac {
    int a, b, c;  //表示凸包一个面上三个点的编号
    bool ok;      //表示该面是否属于最终凸包中的面
} add, F[MAXV*4];

int n, cnt;
int to[MAXV][MAXV];
```

```
void dfs(int, int);
double vlen(pt a) {
    return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
}
double area(pt a, pt b, pt c) {
    return vlen((b-a)*(c-a));
}
double volume(pt a, pt b, pt c, pt d) {
    return (b-a)*(c-a)^(d-a);
}

double ptof(pt &p, fac &f) {  //正：点在面同向
    pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
    return (m * n) ^ t;
}

void deal(int p, int a, int b) {
    int f = to[a][b];
    if (F[f].ok) {
        if (ptof(P[p], F[f]) > eps)
            dfs(p, f);
        else {
            add.a = b, add.b = a, add.c = p, add.ok = 1;
            to[p][b] = to[a][p] = to[b][a] = cnt;
            F[cnt++] = add;
        }
    }
}

void dfs(int p, int cur) {
    F[cur].ok = 0;
    deal(p, F[cur].b, F[cur].a);
    deal(p, F[cur].c, F[cur].b);
    deal(p, F[cur].a, F[cur].c);
}

bool same(int s, int t) {
    pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
    return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b, c, P[F[t].b]))
< eps && fabs(volume(a, b, c, P[F[t].c])) < eps;
}

int solve() {
```

```
if (n < 4)
    return 0;

for (int i = 2; i < n; i++) {
    if (vlen((P[0] - P[1]) * (P[1] - P[i])) > eps) {
        swap(P[2], P[i]);
        break;
    }
}
for (int i = 3; i < n; i++) {
    if (fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) > eps) {
        swap(P[3], P[i]);
        break;
    }
}

cnt = 0;
for (int i = 0; i < 4; i++) {
    add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;
    if (ptof(P[i], add) > 0)
        swap(add.b, add.c);
    to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
    F[cnt++] = add;
}

for (int i = 4; i < n; i++) {
    for (int j = 0; j < cnt; j++) {
        if (F[j].ok && ptof(P[i], F[j]) > eps) {
            dfs(i, j);
            break;
        }
    }
}

int ans = 0;
for (int i = 0; i < cnt; i++)if (F[i].ok) {
        bool nb = 1;
        for (int j = 0; j < i; j++)if (F[j].ok) {
                if (same(i, j)) {
                    nb = 0;
                    break;
                }
            }
        ans += nb;
```

```
        }
    return ans;
}

int main() {
    while (~scanf("%d", &n)) {
        for (int i = 0; i < n; i++)
            scanf("%lf%lf%lf", &P[i].x, &P[i].y, &P[i].z);
        printf("%d\n", solve());
    }
    return 0;
}
```


表面积

```
#include <cmath>
#include <cstdio>

#define sqr(a) ((a) * (a))
#define dis3(a, b) sqrt((double)sqr(a.x - b.x) + sqr(a.y - b.y) + sqr(a.z - b.z))

struct point {
    int x, y, z;
    void Input() {
        scanf("%d%d%d", &x, &y, &z);
    }
    point operator-(point &b) {
        point c;
        c.x = x - b.x;
        c.y = y - b.y;
        c.z = z - b.z;
        return c;
    }
    point cross(point &b) {
        point c;
        c.x = y * b.z - z * b.y;
        c.y = -(x * b.z - b.x * z);
        c.z = x * b.y - y * b.x;
        return c;
    }
    int dot(point &b) {
        return x * b.x + y * b.y + z * b.z;
    }
```

```
};

int check(point p[], int i, int j, int k, int m) {
    point a, b, c, d;
    a = p[j] - p[i];
    b = p[k] - p[i];
    c = p[m] - p[i];
    d = a.cross(b);
    return d.dot(c);
}

double Area(point p[], int i, int j, int k) {
    double a, b, c, s;
    a = dis3(p[i], p[j]);
    b = dis3(p[j], p[k]);
    c = dis3(p[k], p[i]);
    s = (a + b  + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}

int cross(point a, point b, point c) {
    point e, f, g;
    e = b - a;
    f = c - a;
    g = e.cross(f);
    if(sqr(g.x) + sqr(g.y) + sqr(g.z) == 0) return 1;
    return 0;
}

int main() {
    int n, t1, t2, t;
    point p[30];
    double area;
    while(scanf("%d", &n) && n) {
        for(int i = 0; i < n; i++)
            p[i].Input();
        area = 0;
        for(int i = 0; i < n; i++) {
            for(int j = i + 1; j < n; j++) {
                for(int k = j + 1; k < n; k++) {
                    if(cross(p[i], p[j], p[k])) continue;
                    t1 = 0, t2 = 0;
                    for(int m = 0; m < n; m++) {
                        t = check(p, i, j, k, m);
```

```
                    if(t > 0) t1++;
                    else if(t < 0) t2++;
                }
                if(t1 == 0 || t2 == 0) area += Area(p, i, j, k);
            }
        }
    }
    printf("%d\n",int(area + 0.5));
  }
  return 0;
}
```

数论

快速 gcd

```
ll gcd( ll a, ll b) {
    while( b^=a^=b^=a%=b);
    return a;
  }
```

扩展 gcd

```
/* 功能：扩展gcd，求x,y满足ax+by = gcd(a, b)的一组解，同时返回gcd*/
  int exgcd( int a, int b, int &x, int &y){
    if(!b){
      x=1, y=0;
      return a;
    }
    int ret = exgcd(b, a%b, x, y);
    int t=x, x=y, y=t-a/b*y;
    return ret;
  }
```

逆元

```
int InverseMod(int a, int n) {
    int x, y;
    exgcd(a, m, x, y);
    return ( m+x%m) % m;
```

```
    }
```

求欧拉函数

```
ll calPhi( ll n ){
        int ret= n;
        for( int i=2; i*i < n; i++) {
            if( n&i == 0) {
                ret = ret*(i-1)/i;
                while(n%i==0)n/=i;
            }
        }
        if( n!=1) ret=ret*(n-1)*n;
        return ret;
    }
```

欧拉表

```
void SegPhi(int phi[], int uplimit){
        for(int i=0; i<=uplimit; i++)phi[i]=i;
        for(int i=2; i<MaxPrime; i++) {
            if(phi[i] == i) {
                for(int j=i; j<MaxPrime; j+=i) {
                    phi[j]=phi[j]*(i-1)/i;
                }
            }
        }
    }
```

米勒-罗宾素数测试

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<cstdlib>
using namespace std;
typedef long long ll;
ll mulmod(ll a, ll b, ll c){
    a%=c;b%=c;
    ll ans=0;
    while(b){
        if(b&1){
```

```
            ans+=a;
            ans%=c;
        }
        a<<=1;
        if(a>=c)
            a%=c;
        b>>=1;
    }
    return ans;
}
ll gcd(ll a, ll b){
    if(a==0)return 1;
    if(a<0)return gcd(-a, b);
    while(b){
        ll t=a%b;
        a=b;
        b=t;
    }
    return a;
}
ll exp_mod(ll a, ll u, ll n){
    ll ret=1;a=a%n;
    while(u){
        if(u&1)
            ret=mulmod(ret, a, n);
        a=mulmod(a, a, n);
        u>>=1;
    }
    return ret;
}
bool Miller_Rabbin(ll n, ll times){
    if(n==2)return true;
    if(n==1||!(n&1))return false;
    ll a, u=n-1, x, y;
    int t=0;
    while((u&1)==0){
        t++;
        u>>=1;
    }
    srand(100);
    for(int i=0;i<times;i++){
        a=(rand()%(n-1))+1;
        x=exp_mod(a, u, n);
        for(int j=0;j<t;j++){
```

```
            y=mulmod(x, x, n);
            if(x!=1&&x!=n-1&&y==1)
                return false;
            x=y;
        }
        if(y!=1)
            return false;
    }
    return true;
}
ll Pollard_Rho(ll n, ll c){
    ll i=1, k=2, x, y, d;
    x=(rand())%(n-1)+1;
    y=x;
    while(1){
        i++;
        x=(mulmod(x, x, n)+c)%n;
        d=gcd(y-x, n);
        if(d>1&&d<n)
            return d;
        if(y==x)
            return n;
        if(i==k){
            y=x;
            k<<=1;
        }
    }
}
ll minfactor;
void Find_Factor(ll n, ll c){
    //if(!(n&1))return ;
    if(n==1)return ;
    ll p=n;ll k=c;
    minfactor=min(minfactor, p);
    if(Miller_Rabbin(n, 10)){
        minfactor=min(minfactor, p);
        return ;
    }
    while(p>=n)
        p=Pollard_Rho(p, c--);
    Find_Factor(p, k);
    Find_Factor(n/p, k);
}
int main(){
```

```
    int ncase;
    cin>>ncase;
    ll n;
    while(ncase--){
        scanf("%lld", &n);
        if(Miller_Rabbin(n, 6))
            cout<<"Prime"<<endl;
        else{
            minfactor=(((ll)1)<<56);
            Find_Factor(n, 200);
            cout<<minfactor<<endl;
        }
    }
    return 0;
}
```


pell 方程

```
/*  功能：求解佩尔方程的最小整数解(x^2-n*y^2 = 1)
        注意：如果 n 为完全平方数，则返回一组平凡解（x=0, y=1)，否则返回最小正解 */
template<class T>inline int Pell(T n,T& x,T& y){
        T aa=(T)sqrt((double)n),a=aa;
        x=1;
        y=0;
        if(aa*aa==n)return 0;
        T p1=1,p2=0,q1=0,q2=1,g=0,h=1;
        while(true){
            g=-g+a*h;
            h=(n-g*g)/h;
            x=a*p1+p2;
            y=a*q1+q2;
            if(x*x-n*y*y==1)
                return 1;
            p2=p1;
            q2=q1;
            p1=x;
            q1=y;
            a=(g+aa)/h;
        }
        return 1;
    }
```

Fibonacci 第 n 项模 mod(n>=2)

```
int Fibonacci( int n, int k) {
        if( n<=1) return 1;
        n--;
        Matrix x;x.init();x.mod=k, x.size=2;
        x.matrix[0][0]=x.matrix[0][1]=x.matrix[1][0]=1;
        x=x^n;
        return (x.matrix[0][0]+x.matrix[0][1])%k;
    }
```

Fibonacci 前 n 项和% mod

```
/* 功能：求 fibonacci 数列前 n 项和 mod k 注意：f[0]=f[1]=1;n>=2  */
    int FibonacciSum(int n, int k){
        return Mod( Fibonacci(n+2, k)-1, k);
    }
```

求 a^x=b（mod c）的最小 x（O(0.5*c^0.5log c)）

```
int BabyStep(int A,int B,int C){
        map<int,int> Hash;
        ll buf=1%C,D=buf,K;
        int i,d=0,tmp;
        for(i=0;i<=100;buf=buf*A%C,++i)if(buf==B)return i;
        while((tmp=gcd(A,C))!=1){
            if(B%tmp)return -1;
            ++d;
            C/=tmp;
            B/=tmp;
            D=D*A/tmp%C;
        }
        Hash.clear();
        int M=(int)ceil(sqrt((double)C));

    for(buf=1%C,i=0;i<=M;buf=buf*A%C,++i)if(Hash.find((int)buf)==Hash.end())Ha
sh[(int)buf]=i;
        for(i=0,K=PowMod((ll)A,M,C);i<=M;D=D*K%C,++i){
            tmp=Inval((int)D,B,C);
            if(tmp>0&&Hash.find(tmp)!=Hash.end())return i*M+Hash[tmp]+d;
```

```
        }
        return -1;
    }
```

线性同余方程组


```
/*      功能：解线性同余方程组

        参数：ai*x=bi(mod mi)，会返回一个数对(b, m)，即 x=b(mod m)；

                无解时会返回(0, -1) */
struct X{
    ll b, m;
    X(ll bb=0, ll mm=0):b(bb), m(mm){}
};
ll gcd(ll a, ll b){
    if(b==0) return a;
    return gcd(b, a%b);
}
ll exgcd( ll a, ll b, ll &x, ll &y) {
    if(!b) {
        x=1, y=0;
        return a;
    }
    ll ret = exgcd(b, a%b, x, y);
    ll t=x;
    x=y, y=t-a/b*y;
    return ret;
}
ll InverseMod(ll a, ll n) {
    if( n<= 0)return -1;
    ll x, y;
    ll comDiv = exgcd( a, n, x, y);
    return ((x%n)+n)%n;
}
ll Mod(ll x, ll m){
    return (x%m+m)%m;
}
X Liner_Cong_Equations(int n){
    ll x = 0, m = 1;
    for(int i=0; i<n; i++){
        ll ai, bi, mi;
        ai = 1;
```

```
        scanf("%lld%lld", &mi, &bi);
        if(x == -1) continue;
        ll a = ai*m, b = Mod(bi-ai*x, mi), d = gcd(a, mi);
        if(b%d) {x = -1;continue;}
        a /= d, b /= d;
        ll cur_m = mi / d;//为了不改变原来数组的内容，对于 mi 的处理，为使用临时变量
        ll t = b*InverseMod(a, cur_m) % cur_m;
        x += m*t;
        m *= cur_m;
        x = Mod(x, m);
    }
    if(x==-1) return X(-1, -1);
    return X(x, m);
}
```

求解 n!=xp^e

```
/*      功能：求解 n!=xp^e 的值，n,p 已知，求 x%p 和 e

        注意：需要先预处理 n! mod p 的表，保存在 fact[]里   */

    int ModFact(int n, int p, int &e, int fact[]) {
        e = 0;
        if( n==0 ) return 1;
        int res = ModFact( n/p, p, e, fact);
        e += n/p;
        if( (n/p)&1 )return res*(p-fact[n%p])%p;
        return res*fact[n%p]%p;
    }
```

大组合数取模

```
int ModComb(int p, int q, int mod, int fact[]) {
        int x=q, y=p-q;
        return fact[p]*PowMod(fact[x]*fact[y], mod-2, mod)%mod;
    }
```

FFT

```
/* 快速计算大数乘法，以 A*B plus 为例*/

#include <cstdio>
```

```
#include <cstring>
#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

const double PI = acos(-1.0);
//复数结构体
struct complex
{
    double r,i;
    complex(double _r = 0.0,double _i = 0.0)
    {
        r = _r; i = _i;
    }
    complex operator +(const complex &b)
    {
        return complex(r+b.r,i+b.i);
    }
    complex operator -(const complex &b)
    {
        return complex(r-b.r,i-b.i);
    }
    complex operator *(const complex &b)
    {
        return complex(r*b.r-i*b.i,r*b.i+i*b.r);
    }
};
/*
 * 进行 FFT 和 IFFT 前的反转变换。
 * 位置 i 和（i 二进制反转后位置）互换
 * len 必须去 2 的幂
 */
void change(complex y[],int len)
{
    int i,j,k;
    for(i = 1, j = len/2;i < len-1; i++)
    {
        if(i < j)swap(y[i],y[j]);
        //交换互为小标反转的元素，i<j 保证交换一次
        //i 做正常的+1，j 左反转类型的+1,始终保持 i 和 j 是反转的
        k = len/2;
        while( j >= k)
        {
```

```
            j -= k;
            k /= 2;
        }
        if(j < k) j += k;
    }
}
/*
 * 做 FFT
 * len 必须为 2^k 形式，
 * on==1 时是 DFT，on==-1 时是 IDFT
 */
void fft(complex y[],int len,int on)
{
    change(y,len);
    for(int h = 2; h <= len; h <<= 1)
    {
        complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h));
        for(int j = 0;j < len;j+=h)
        {
            complex w(1,0);
            for(int k = j;k < j+h/2;k++)
            {
                complex u = y[k];
                complex t = w*y[k+h/2];
                y[k] = u+t;
                y[k+h/2] = u-t;
                w = w*wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0;i < len;i++)
            y[i].r /= len;
}
const int MAXN = 200010;
complex x1[MAXN],x2[MAXN];
char str1[MAXN/2],str2[MAXN/2];
int sum[MAXN];
int main()
{
    while(scanf("%s%s",str1,str2)==2)
    {
        int len1 = strlen(str1);
        int len2 = strlen(str2);
```

```
        int len = 1;
        while(len < len1*2 || len < len2*2)len<<=1;
        for(int i = 0;i < len1;i++)
            x1[i] = complex(str1[len1-1-i]-'0',0);
        for(int i = len1;i < len;i++)
            x1[i] = complex(0,0);
        for(int i = 0;i < len2;i++)
            x2[i] = complex(str2[len2-1-i]-'0',0);
        for(int i = len2;i < len;i++)
            x2[i] = complex(0,0);
        //求 DFT
        fft(x1,len,1);
        fft(x2,len,1);
        for(int i = 0;i < len;i++)
            x1[i] = x1[i]*x2[i];
        fft(x1,len,-1);
        for(int i = 0;i < len;i++)
            sum[i] = (int)(x1[i].r+0.5);
        for(int i = 0;i < len;i++)
        {
            sum[i+1]+=sum[i]/10;
            sum[i]%=10;
        }
        len = len1+len2-1;
        while(sum[len] <= 0 && len > 0)len--;
        for(int i = len;i >= 0;i--)
            printf("%c",sum[i]+'0');
        printf("\n");
    }
    return 0;
}
```

公式类

自然数 k 次方和

$$S_k = \frac{1}{k+1}[(1+n)^{k+1} - (n+1) - (C_{k+1}^2 S_{k-1} + C_{k+1}^3 \cdot S_{k-2} + C_{k+1}^4 S_{k-3} + \cdots + C_{k+1}^{k-2} S_3 + C_{k+1}^{k-1} S_2 + C_{k+1}^k S_1)].$$

$$S_1 = 1+2+3+\cdots+n = \frac{1}{2}n(n+1);$$

$$S_2 = 1^2+2^2+3^2+\cdots+n^2 = \frac{1}{6}(n+1)(2n+1)n;$$

$$S_3 = 1^3+2^3+3^3+\cdots+n^3 = \frac{1}{4}n^2(n+1)^2;$$

$$S_4 = 1^4+2^4+3^4+\cdots+n^4 = \frac{1}{30}n(n+1)(6n^3+9n^2+n-1);$$

$$S_5 = 1^5+2^5+3^5+\cdots+n^5 = \frac{1}{12}n^2(n+1)(2n^3+4n^2+n-1);$$

$$S_6 = 1^6+2^6+3^6+\cdots+n^6 = \frac{1}{42}n(n+1)(6n^5+15n^4+6n^3-6n^2-n+1);$$

$$S_7 = 1^7+2^7+3^7+\cdots+n^7 = \frac{1}{24}n^2(n+1)(3n^5+9n^4+5n^3-5n^2-2n+2);$$

$$S_8 = 1^8+2^8+3^8+\cdots+n^8 = \frac{1}{90}n(n+1)(10n^7+35n^6+25n^5-25n^4-17n^3+17n^2+3n-3);$$

$$S_9 = 1^9+2^9+3^9+\cdots+n^9 = \frac{1}{20}n^2(n+1)(2n^7+8n^6+7n^5-7n^4-7n^3+7n^2+3n-3);$$

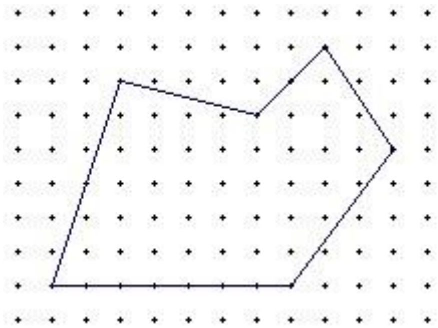
$$S_{10} = 1^{10}+2^{10}+3^{10}+\cdots+n^{10} = \frac{1}{66}n(n+1)(6n^9+27n^8+28n^7-28n^6-38n^5+38n^4+28n^3-28n^2-5n+5).$$

划分问题

1、 n 个点最多把直线分成 C(n,0)+C(n,1)份；
2、n 条直线最多把平面分成 C(n,0)+C(n,1)+C(n,2)份；
3、n 个平面最多把空间分成 C(n,0)+C(n,1)+C(n,2)+C(n,3)=(n3+5n+6)/6 份；
4、n 个空间最多把“时空”分成 C(n,0)+C(n,1)+C(n,2)+C(n,3)+C(n,4)份.

皮克定理

一个多边形的顶点如果全是格点，这多边形就叫做格点多边形。



给定顶点坐标均是整点（或正方形格点）的简单多边形，皮克定理说明了其面积 S 和内部格点数目 a、边上格点数目 b 的关系：

S=a+ b/2 - 1。

(其中 a 表示多边形内部的点数,b 表示多边形边界上的点数,S 表示多边形的面积)

卡特兰数

原理：

令 h(1)＝1,h(0)=1，catalan

数满足递归式：

h(n)= h(0)*h(n-1)+h(1)*h(n-2) + ... + h(n-1)h(0) (其中 n>=2)

另类递归式：

h(n) = h(n-1)*(4*n-2)/(n+1);

该递推关系的解为：

h(n)=C(2n,n)/(n+1) (n=1,2,3,...)

卡特兰数的应用

（实质上都是递归等式的应用）

错排公式

当 n 个编号元素放在 n 个编号位置,元素编号与位置编号各不对应的方法数用 M(n)表示,那么 M(n-1)就表示 n-1 个编号元素放在 n-1 个编号位置,各不对应的方法数,其它类推.

第一步,把第 n 个元素放在一个位置,比如位置 k,一共有 n-1 种方法;

第二步,放编号为 k 的元素,这时有两种情况.1,把它放到位置 n,那么,对于剩下的 n-2 个元素,就有 M(n-2)种方法;2,不把它放到位置 n,这时,对于这 n-1 个元素,有 M(n-1)种方法;

综上得到递推公式：

M(n)=(n-1)[M(n-2)+M(n-1)]特殊地，M(1)=0,M(2)=1

通项公式：

M(n)=n![(-1)^2/2!+…+(-1)^(n-1)/(n-1)!+(-1)^n/n!]

优美的式子：

Dn=[n!/e+0.5],[x]为取整函数.

公式证明较简单. 观察一般书上的公式，可以发现 e-1 的前项与之相同，然后作比较可得/Dn-n!e-1/<1/(n+1)<0.5,于是就得到这个简单而优美的公式（此仅供参考）

约瑟夫环

令 f 表示 i 个人玩游戏报 m 退出最后胜利者的编号，最后的结果自然是 f[n].
递推公式：

```
f[1]=0;
f=(f[i-1]+m)%i;  (i>1)
```

有了这个公式，我们要做的就是从 1-n 顺序算出 f 的数值，最后结果是 f[n]。因为实际生活中编号总是从 1 开始，我们输出 f[n]+1 由于是逐级递推，不需要保存每个 f，程序也是异常简单：

乘法与因式分解

$$1.3 \quad a^n-b^n=\begin{cases}(a-b)(a^{n-1}+a^{n-2}b+a^{n-3}b^2+\cdots+ab^{n-2}+b^{n-1}) & (n\text{为正整数})\\(a+b)(a^{n-1}+a^{n-2}b-a^{n-3}b^2+\cdots+ab^{n-2}-b^{n-1}) & (n\text{为偶数})\end{cases}$$

$$a^b+b^n=(a+b)(a^{n-1}-a^{n-2}b+a^{n-3}b^2-\cdots-ab^{n-2}+b^{n-1}) \qquad (n\text{为奇数})$$

图论

缩点构造新图

```
//uva11323
/*
给出各个关系，然后对原图缩点，形成一个新图，每个点都有权值，
缩点的权值是原强连通分量里点的个数，其它点的权值是 1.
先用强连通分量缩点，每个点的权值都是连通分量的点的个数，
然后选一些点使权值最大，这些点之间必须是至少有一方能到达
另一方
比如 1-2,1-3.就只能选 1 和 2 或者 1 和 3。就不能选 1,2,3.因为 2 没
法到 3,3 也没法到 2
*/
#include<cstdio>
#include<cstring>
#include<stack>
#include<vector>
#include<algorithm>
#define MAXN 1010
#define MAXM 50010
using namespace std;
int head[MAXN], low[MAXN], dfn[MAXN], dag[MAXN];
bool instack[MAXN];
```

```
int newhead[MAXN];
int valdot[MAXN],finalval[MAXN];
//valdot[]用来记录每个点的权值
//finalval[]用来记录所有可能的权值
stack<int>stacks;
int cnt, t, dcnt, ncnt;
//cnt 在 addedges()要用
//t 在 tarjan()里用来记录是第几次遍历到该处
//dcnt 即 dagcnt, dag[]数组用来标记每个点属于哪个连通分量, 其值可代表缩点
//ncnt 即 newdagcnt, 在构建新图里的 adddag()中使用

int n;
struct node {
    int v, next;
} newdag[MAXN];
//用来记录新生成的图
struct nodes {
    int v,next;
} edges[MAXM];
//原始边
void addedges(int u,int v) {
    edges[cnt].v=v;
    edges[cnt].next=head[u];
    head[u]=cnt++;
}
void clearstack() {
    while(!stacks.empty())
        stacks.pop();
}//清空栈
void init() {
    memset(finalval,0,sizeof(finalval));
    memset(newhead,-1,sizeof(newhead));
    memset(valdot,0,sizeof(valdot));
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    memset(dag,0,sizeof(dag));
    memset(head,-1,sizeof(head));
    memset(instack, false, sizeof(instack));
    clearstack();
    cnt=0;
    t=1;
    dcnt=0;
    ncnt=0;
}
```

```
void tarjan(int u) {
    dfn[u]=low[u]=t++;
    stacks.push(u);
    instack[u]=true;
    for(int i=head[u]; i!=-1; i=edges[i].next) {
        int v=edges[i].v;
        if(!dfn[v]) {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        } else if(instack[v]) {
            low[u]=min(low[u], dfn[v]);
        }
    }
    int tmpcnt=0;
    if(dfn[u]==low[u]) { //给每个点进行标记，更新 dag[]，标记的值为其所在的缩点的标号
        int tmp=-1;
        while(tmp!=u) {
            tmp=stacks.top();
            stacks.pop();
            dag[tmp]=dcnt;
            tmpcnt++;//记录这个强连通分量里原始点的数目
            instack[tmp]=false;
        }
        valdot[dcnt]=tmpcnt;
        dcnt++;
    }
}
bool isexist(int u,int v) { //判断新图里是否存在 u->v 这条边
    for(int i=newhead[u]; i!=-1; i=newdag[i].next)
        if(newdag[i].v==v)
            return true;
    return false;
}
void adddag(int u, int v) {
    newdag[ncnt].v=v;
    newdag[ncnt].next=newhead[u];
    newhead[u]=ncnt++;
}
void build_dag() { //构建新图
    for(int i=1; i<=n; i++) {
        for(int j=head[i]; j!=-1; j=edges[j].next) {
            int v=edges[j].v;
            if(dag[i]!=dag[v]) {
                if(isexist(dag[i],dag[v]))
```

```
                continue;
            else
                adddag(dag[i],dag[v]);
        }
    }
  }
}
```


2SAT

```
#include<cstdio>
#include<cstring>
#include<vector>
#include<stack>
#include<cmath>
#include<algorithm>
#define mem(x,y) memset(x,y,sizeof(x))
#define MAXN 1020
using namespace std;
int dag[MAXN], low[MAXN], dfn[MAXN];
int n, ndag,cnt;
bool instack[MAXN];
vector<int>dot[MAXN];
stack<int>stacks;
void addedges(int planea, int astate, int planeb, int bstate) {
    planea=2*planea+astate;
    planeb=2*planeb+bstate;
    dot[planea].push_back(planeb);
}
void buildmap(int n) {

}
bool ok() {
    bool yes = false;
    for(int i = 0; i < n; i++) {
        for(int j =i+1; j < n; j++) {
            if(b[i][j]) return true;
        }
    }
    return false;
}
void tarjan(int u) {
    dfn[u]=low[u]=cnt++;
```

```
    stacks.push(u);
    instack[u]=true;
    int len=dot[u].size();
    for(int i=0; i<len; i++) {
        int v=dot[u][i];
        if(!dfn[v]) {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        } else if(instack[v]) {
            low[u]=min(low[u],dfn[v]);
        }
    }
    if(dfn[u]==low[u]) {
        int tmp=-1;
        while(tmp!=u) {
            tmp=stacks.top();
            stacks.pop();
            dag[tmp]=ndag;
            instack[tmp]=false;
        }
        ndag++;
    }

}
void builddag() {
    for(int i=0; i<2*n; i++)
        if(!dfn[i])tarjan(i);
}
void init() {
    mem(dag, 0);
    mem(dfn, 0);
    mem(low, 0);
    mem(instack, false);
    for(int i=0; i<=2*n; i++)
        dot[i].clear();
    while(!stacks.empty())
        stacks.pop();
    cnt=1;
    ndag=0;
}
bool haveresult() {
    init();
    buildmap(n);
    builddag();
```

```
    for(int i=0; i<2*n; i+=2) {
        if(dag[i]==dag[i^1])
            return false;
    }
}
return true;
}
void readdat() {

}
int main() {
    while(~scanf("%d",&n)) {
        init();
        readdat();
        if(haveresult()) puts("YES");
        else puts("NO");
    }
}
```

瓶颈树

```
/*n<=50000, m<=100000 的无向图，对于 Q<=50000 个询问，每次求 q->p 的瓶颈路*/
#include<cstdio>
#include<queue>
#include<climits>
#include<algorithm>
#include<cstring>
#define MAXN  50010
#define MAXL 200010
using namespace std;
int ndots;
struct node {
    int v,val;
    int next;
} mst[MAXL];
struct edge {
    int from,to;
    int val;
} edges[MAXL];
int msthead[MAXN],level[MAXN],vis[MAXN],pre[MAXN],dis[MAXL];
int f[MAXN];
int mstn,lines;
void addmst(int u,int v,int val) {
```

```
    mst[mstn].v=v;
    mst[mstn].val=val;
    mst[mstn].next=msthead[u];
    msthead[u]=mstn++;
    mst[mstn].v=u;
    mst[mstn].val=val;
    mst[mstn].next=msthead[v];
    msthead[v]=mstn++;
}
int findit(int x) {
    int y=x;
    while(x!=f[x])
        x=f[x];
    while(y!=x) {
        int tmp=f[y];
        f[y]=x;
        y=tmp;
    }
    return x;
}
void merge(int x,int y) {
    int fx=findit(x);
    int fy=findit(y);
    if(fx!=fy)
        f[fx]=fy;
}
int cmp(const struct edge &a,const struct edge &b) {
    if(a.val!=b.val)return a.val<b.val;
    if(a.from!=b.from)return a.from<b.from;
    return a.to<b.to;
}
void kruskal() {
    int k=0;
    int x,y;
    mstn=0;
    memset(msthead,-1,sizeof(msthead));
    for(int i=1; i<=ndots; i++)
        f[i]=i;
    sort(edges,edges+lines,cmp);
    for(int i=0; i<lines; i++) {
        if(k==ndots-1)break;
        x=findit(edges[i].from);
        y=findit(edges[i].to);
        if(x!=y) {
```

```
            merge(x,y);
            //dis[edges[i].to]=edges[i].val;
            addmst(edges[i].from,edges[i].to,edges[i].val);
            k++;
        }
    }
}

void makelevel() {
    memset(vis,0,sizeof(vis));
    memset(level,-1,sizeof(level));
    queue<int>qb;
    qb.push(1);
    level[1]=0;
    while(!qb.empty()) {
        int tmp=qb.front();
        qb.pop();
        for(int i=msthead[tmp]; i!=-1; i=mst[i].next) {
            int iv=mst[i].v;
            if(vis[iv])continue;
            if(level[iv]<0) {
                level[iv]=level[tmp]+1;
                pre[iv]=tmp;
                qb.push(iv);
                dis[iv]=mst[i].val;
                vis[iv]=1;
            }
        }
    }
}
int solve(int x,int y) {
    int val1=-1,val2=-1;
    if(level[x]==-1||level[y]==-1)return 0;
    if(level[x]>level[y]) {
        while(level[x]!=level[y]) {
            if(val1<dis[x])val1=dis[x];
            x=pre[x];
        }
    } else if(level[x]<level[y]) {
        while(level[x]!=level[y]) {
            if(val2<dis[y])val2=dis[y];
            y=pre[y];
        }
    }
```

```
    while(x!=y) {
        if(val1<dis[x])val1=dis[x];
        x=pre[x];
        if(val2<dis[y])val2=dis[y];
        y=pre[y];
    }
    return val1>val2?val1:val2;
}
int main() {
    scanf("%d%d",&ndots,&lines);
    for(int i=1; i<=ndots; i++)
        f[i]=i;
    for(int i=0; i<lines; i++) {
        scanf("%d%d%d",&edges[i].from,&edges[i].to,&edges[i].val);
    }
    kruskal();
    makelevel();
    int questions;
    scanf("%d",&questions);
    int starts,ends;
    while(questions--) {
        scanf("%d%d",&starts,&ends);
        printf("%d\n",solve(starts,ends));
    }
    while(~scanf("%d%d",&ndots,&lines)) {
        printf("\n");
        for(int i=1; i<=ndots; i++)
            f[i]=i;
        for(int i=0; i<lines; i++) {
            scanf("%d%d%d",&edges[i].from,&edges[i].to,&edges[i].val);
        }
        kruskal();
        makelevel();
        int questions;
        scanf("%d",&questions);
        int starts,ends;
        while(questions--) {
            scanf("%d%d",&starts,&ends);
            printf("%d\n",solve(starts,ends));
        }
    }
}
```