Computational Geometry

November 3, 2018

Contents

0 Computational Geometry

0.1 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);
}


//□□¨□¶□□□□,¶¥µ🄵□□□□□□□□□□□³□,□□□□□□□□□²□□
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];
}


//□□¨□¶□□□□,¶¥µ🄵□□□□□□□□□□□³□,²»□□□□□□□□□²□□
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];
}


//□□□□□□¶□□□□□□□□□□□□□□,¶¥µ🄵□□□□□□□□□□□□³□
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];
}


//□□□□□□¶□□□□□□,¶¥µ🄵□□□□□□□□□□□□³□,□□□□□□□□□µ»□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];
}


//□□□□□□□□□□□□□□□,¶¥µ🄵□□□□□□□□□□□□³□
//on_edge±□□µ□□□□□□□□□□□µ¸µ»□¸offset□¶□□□□□□±□□□□□
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;
}


//□□□□□□□□□□□□□□□□□□,¶¥µ🄵□□□□□□□□□□□□³□□□□□□□,□· µ»□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;
}


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 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);
}


//¶□□□□□□□□
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;
}
```

0.2　多边形切割

```
//¶□□□□□□□
//¿□□□□□□□□🄵
#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;
}


//½«¶□□□□□□l1,l2□¶¨µ□□□□□□□□side²□□□□,±£□l1,l2,side²»¹²□□
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;
}
```

0.3 浮点函数

```
//¸¡µ🄵°□¯□□¿□
#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;};


//¼□□□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);
}


//¼□□□dot product (P1-P0).(P2-P0)
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);
}


//}µ□□□□
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 distance(double x1,double y1,double x2,double y2){
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}


//□□□□µ🄵□□
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));
}


//□□□□□□□□□□□□□,°□(¶□□
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;
}


//□□□□□□□□□□□□□,²»°□(¶□□
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)||!
    zero(y-y2));
}


//□□}µ□□□□□□²□,µ□□□□□□□□µ»□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;
}


//□□}µ□□□□□□□□□,µ□□□□□□□□µ»□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;
}


```

```
//□□}□□□□□□
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));
}


//□□}□¹□□
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));
}


//□□}□□□□□¸°□(¶□□□¿·□□□□
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);
}


//□□}□□□□□¸²»°□(¶□□□¿·□□□□
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);
}


//¼□□□}□□»µ□,□□□□□□□□□□□□□□□□□□!
//□□□»µ□□□□□□□□□□□□□(□»¹□□□□□□□□□□□□!)
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;
```

```
}


//µ🄵▯▯▯▯▯▯▯▯µ▯
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);
}


//µ🄵▯▯▯▯▯
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);
}


//µ🄵▯▯▯▯▯▯▯▯▯µ▯
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);
}


//µ🄵▯▯▯▯▯▯
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);
```

```
}


//□-V□□P□¶¥µ□□□□□□□□angle²¢□ · Ŵ□scale±¶
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;
}


//pµ□□□□□□LµĶ□□□
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;
}


//□□}µ□□□ · □□□
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;
}


// □□□□□□□µ¾¢□□🄵□□□□□£□
// a1,b1,c1□¾µ□□□□³½□(a1 x + b1 y + c1 = 0 ,□□)□□□;
a2,b2,c2□□□□□□□□³½□□□□;
a,b,c□ · ´□□□□□□³½□□□□.
// ¹□□□□□½□□ģ¬□□□□□□□□□□□□□□-:<-b2,a2>£» · ´□□□□□-:<b,-a>.
// ²»□□□¼□□□□□□□"□□□□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)□□□□□□□□□🄵□□Ľ»µ🄵
    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;
}
```

0.4 面积

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


//¼□□□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);
}


//¼□□□□□½□□□□□□,□□□□□□¶¥µ□
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;
}


//¼□□□□□½□□□□□□,□□□□□□□±¤
double area_triangle(double a,double b,double c){
    double s=(a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}


//¼□□□□□□□□□□,¶¥µ🄵□□□□□□□□□□□³□
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;
}
```

0.5 球

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


//¼□□□□□Ľ□lat±□□□¶□,-90<=w<=90,lng±□□¾¶□
```

```
// · µ»▯}µ▯▯▯▯▯▯▯▯▯¡¶▯▯▯▯L’▯,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));
}


//¼▯▯▯▯▯▯,r▯▯▯🄵
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)));
}


//¼▯▯▯▯▯▯▯▯▯▯,r▯▯▯🄵
inline double sphere_dist(double r,double lng1,double lat1,double lng2,double lat2){
    return r*angle(lng1,lat1,lng2,lat2);
}


//▯▯▯🄵▯▯
//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){   /**** µ🄵▯▯Lµ’▯▯▯▯▯ · ½ **/
    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);
```

```
}


/**¡¡□ý³□(µ🄵□□□ľ□□□□r)·¨□□□  (□□□□□□□□-·¨□□□, }□□□□□□, ¶¼OK)*/
/* □□□□-·□-±□□·¢µ□□□□, ±□□□□□□□□□□□,¹□K±□□□□□□, t□□»µ□***/
/*
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;
}
*/


/**** □□□□-·¨□□□  ***/
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;
}
/*¼□□□□□□□□□□□□□□½□□ */
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;
}
/* ·µ»□□□□□□□□□µ□□□ı□□,¾□²»□□,·µ»□-1 */
int update ()
{
    int sign = -1, i;
    double k = 1e100, tmp;
    point inter, t;
    for (i = 1; i <= n; i++){ //□□½□□□□□µ□□□
        if (!find (cir[i], l, tmp, t)) continue;
        if (dcmp (tmp - k) < 0) k = tmp, inter = t, sign = i;
    }
    //ray ±□□□
    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++) //□□□□□□□□□□□□□
        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++){ //□□□□□□□´□□µ□□□ι□□
        int sign = update ();
        if (sign == -1) break;
        if (i == 0) printf ("%d", sign);
        else if (i < 10) printf (" %d", sign);
        else printf (" etc.");
    }
    puts ("");
}
```

0.6 三角形

```
#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;
}


//□□□□
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);
}


//□□□□
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);
}


//´¹□□
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);
}


//□□□□
//µ½□□½□□□□□¶¥µ□□□□□□ · ½°□□□□µ□□
//□□½□□□□½□□□±□□□□»□□□□□□
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);
}


// · □□□□
```

```
//µ½□□½□□□□□¶¥µ□□□□º□□□□µ□□
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)+distance(v,c))
                        u=v;
                }
    return u;
}


//□□□□□□🄵 □□½□□□□□□□□□□³□□□□
#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;
 }
```

0.7 三维几何

```
//□□□¼¸°□¯□□¿□
#include <math.h>
#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;};
```

```


//¼□□□cross product U x V
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;
}


//¼□□□dot product U . V
double dmult(point3 u,point3 v){
    return u.x*v.x+u.y*v.y+u.z*v.z;
}


//□-²□ 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;
}


//□□□□□□-
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));
}


//}µ□□□□,µ¥²□□□□□□-´□□
double distance(point3 p1,point3 p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z));
}


//□□-´□□
double vlen(point3 p){
    return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}


//□□□□µ□□□
int dots_inline(point3 p1,point3 p2,point3 p3){
    return vlen(xmult(subt(p1,p2),subt(p2,p3)))<eps;
}


//□□□□□□□
int dots_onplane(point3 a,point3 b,point3 c,point3 d){
    return zero(dmult(pvec(a,b,c),subt(d,a)));
```

```
}


//▯▯▯▯▯▯▯▯▯▯▯▯▯,°▯(¶▯▯▯²▯▯
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;
}


//▯▯▯▯▯▯▯▯▯▯▯▯▯,²»°▯(¶▯▯
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));
}


//▯▯▯▯▯▯▯▯▯▯▯▯½▯▯▯▯▯,°▯(±▯▯,▯▯µ🄵▯▯▯▯▯▯▯▯
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))));
}


//▯▯▯▯▯▯▯▯▯▯▯▯½▯▯▯▯▯,²»°▯(±▯▯,▯▯µ🄵▯▯▯▯▯▯▯▯
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;
}


//▯▯}µ▯▯▯▯▯▯²▯,µ▯▯▯▯▯▯▯▯µ»▯0,²»¹²▯▯▯▯▯▯▯▯
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)))>eps;
}
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;
}


//□□}μ□□□□□□□□□,μ□□□□□□□□μ»□0,²»¹²□□□□□□□□
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)))<-
    eps;
}
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;
}


//□□}μ□□□□□□□□,μ□□□□□□□□μ»□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;
}


//□□}μ□□□□□□□□□,μ□□□□□□□□μ»□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;
}


//□□}□□□□□
int parallel(line3 u,line3 v){
    return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<eps;
}
int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
    return vlen(xmult(subt(u1,u2),subt(v1,v2)))<eps;
}


//□□}□□□□□□
int parallel(plane3 u,plane3 v){
    return vlen(xmult(pvec(u),pvec(v)))<eps;
}
int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;
}


//□□□□□□□□□□□□
int parallel(line3 l,plane3 s){
    return zero(dmult(subt(l.a,l.b),pvec(s)));
}
int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));
}


//□□}□¹□
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)));
}


//□□}□□洹□
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)));
}


//□□□□□□□□□□□□
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;
}


//□□}□□□□□,°□(¶□□□¿ · □□□□
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)||
    dot_online_in(v2,u1,u2);
}


//□□}□□□□□,²»°□(¶□□□¿ · □□□□
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);
}


//□□□□□□□□□□□½□□□□,°□(½»□□)□□□²¿ · □)°□°¬
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);
}


//□□□□□□□□□□□½□□□□,²»°□(½»□□)□□□²¿·□)°□°¬
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);
}


//¼□□□}□□»µ□,□□□□□□□□□□□□□□□□□□□!
//□□□»µ□□□□□□□□□□□□(□»¹□□□□□□□□□□□!)
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;
}


//¼□□□□□□□□🄵µ□,□□□□□□□□□□□□□□,²¢±£□□□µ🄵¹²□□!
//□□□□□□□½□□□»µ□□□□□□□□
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;
}


//¼□□□}□□🄵□□,□□□□□□□□□□□□□□□,²¢±£□□µ🄵¹²□□!
                                    □
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):intersection(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):intersection(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(v1,v2,u1,u2,u3);
    ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v3,v1,u1,u2,u3);
    return ret;
}


//µ🄵□□□□□
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);
}


//µ🄵□□□□□□
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));
}


//□½□□□□□□
   □
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);
}


//}□□□□cos
         □
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));
```

```
}


//}□□□□□cos□
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
    ));
}


//□□□□□□□sin□
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));
}
```

0.8 凸包

```
//□□□□
#define maxn 100005


struct point
{double x,y;}p[maxn],s[maxn];
bool operator < (point a,point b)
{return a.x<b.x || a.x==b.x&&a.y<b.y;}


int n,f;


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


void Convex(point *p,int &n)
{
    sort(p,p+n);
    int i,j,r,top,m;
    s[0] = p[0];s[1] = p[1];top = 1;
    for(i=2;i<n;i++)
    {
        while( top>0 && cp(p[i],s[top],s[top-1])>=0 ) top--;
        top++;s[top] = p[i];
    }
    m = top;
    top++;s[top] = p[n-2];
    for(i=n-3;i>=0;i--)
    {
        while( top>m && cp(p[i],s[top],s[top-1])>=0 ) top--;
        top++;s[top] = p[i];
    }
    top--;
    n = top+1;
}


// ¼«½□□□
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <math.h>
using namespace std;
#define maxn 100005
int N;
struct A
{
    int x,y;
    int v,l;
}P[maxn];
int xmult(int x1,int y1,int x2,int y2,int x3,int y3)
{
    return (y2-y1)*(x3-x1)-(y3-y1)*(x2-x1);
}
void swap(A &a,A &b)
{
    A t = a;a = b,b = t;
}
bool operator < (A a,A b)
{
    int k = xmult(P[0].x,P[0].y,a.x,a.y,b.x,b.y);
    if( k<0 )
        return 1;
    else if( k==0 )
    {
        if( abs(P[0].x-a.x)<abs(P[0].x-b.x) )
            return 1;
        if( abs(P[0].y-a.y)<abs(P[0].y-b.y) )
            return 1;
    }
    return 0;
}
void Grem_scan(int n)
{
    int i,j,k,l;
    k = 0x7fffffff;
    for(i=0;i<n;i++)
        if( P[i].x<k || P[i].x==k && P[i].y<P[l].y )
        k = P[i].x,l = i;
    swap(P[l],P[0]);
    sort(P+1,P+n);

    l = 3;
    for(i=3;i<n;i++)
    {
        while( xmult(P[l-2].x,P[l-2].y,P[l-1].x,P[l-1].y,P[i].x,P[i].y)>0 )
            l--;
        P[l++] = P[i];
    }
}
main()
```

```
{
    int i,j,k,l;
    N = 0;
    while( scanf("%d%d",&P[N].x,&P[N].y)!=EOF )
        N++;
    Grem_scan(N);
    for(i=0;i<N;i++)
        if( P[i].x==0 && P[i].y==0 )
        break;
    k = i++;
    printf("(0,0)\n");
    while( i!=k )
        printf("(%d,%d)\n",P[i].x,P[i].y),i = (i+1)%N;
}


//¾□□¹□ · ¨
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;
#define maxn 55
struct A
{
    int x,y;
}P[maxn];
int T,N;
bool B[maxn];
int as[maxn],L;
int xmult(A a,A b,A c)
{
    return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
}
main()
{
    int i,j,k,l;
    scanf("%d",&T);
    while( T-- )
    {
        scanf("%d",&N);
        k = 0x7ffffff;
        for(i=0;i<N;i++)
        {
            scanf("%d%d%d",&j,&P[i].x,&P[i].y);
            if( P[i].y<k )
                k = P[i].y,l = i;
        }
        memset(B,0,sizeof(B));
        B[l] = 1;
        as[0] = l;
        L = 1;
        while( 1 )
        {
            A a,b;
            if( L==1 )
                a.x = 0,a.y = P[as[0]].y;
            else
                a = P[as[L-2]];
            b = P[as[L-1]];


            k = -1;
            for(i=0;i<N;i++)
            {
                if( B[i] )
                    continue;
                if( xmult(a,b,P[i])<0 )
                    continue;
                if( k==-1 || xmult(P[as[L-1]],P[k],P[i])<0 || xmult(P[as[L-1]],P[k],P[i])==0 && P[i].y<P[k].y )
                    k = i;
            }
            if( k==-1 )
                break;
            B[k] = 1;
            as[L++] = k;
        }
        printf("%d ",L);
        for(i=0;i<L;i++)
            printf("%d ",as[i]+1);
        printf("\n");
    }
}
```

0.9 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;}


//¶□□□□□□□□□¸□□□□□
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;
}


//¶□□□□□□□□□¸□□□□□
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;
}
```

0.10 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;
}


//□□□□□□□□¸°□(□□□□
int intersect_line_circle(point c,double r,point l1,point l2){
    return disptoline(c,l1,l2)<r+eps;
}


//□□□□□□□□□¸°□(¶□□□□□□□
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;
}


//□□□°□□□□¸°□(□□□□
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;
}


//¼□□□□□□½µ□p□□□µ□,□□p□□□□□□□□, · µ»□p±¾□□
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;
}


//¼□□□□□□□□µĽ»µ□,±£□□□□□□□»µ□
                     □
                     □
//¼□□□□□□□□□µĽ»µ□□□□□□□□□°□□□□□□□□□□□□□□
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;
}


//¼□□□□□□□µĽ»µ□,±£□□□□□□»µ□,□□□»□□□
                    □
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);
}


//½«□□-p□□□□□□□angle½□□□
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;
}
//□□□□□□µ□□□(o,r)µ□}¸□□□□result1°□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;
```

```
}
```

0.11 矢量运算求几何模板

```
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#define MAX_N 100
using namespace std;


///////////////////////////////////////////////////////////////////
//³£-□□
const double INF        = 1e10;     // □□□□□
const double EPS        = 1e-15;    // ¼□□🄵¶□
const int LEFT          = 0;        // µ□□□□□□□□
const int RIGHT         = 1;        // µ□□□□□□□□
const int ONLINE        = 2;        // µ□□□□□□□
const int CROSS         = 0;        // }□□□□
const int COLINE        = 1;        // }□□²□□
const int PARALLEL      = 2;        // }□□□□□
const int NOTCOPLANAR   = 3;        // }□§¹²□□
const int INSIDE        = 1;        // µ□□□□□□□¿□
const int OUTSIDE       = 2;        // µ□□□□□□□□
const int BORDER        = 3;        // µ□□□□□□□□
const int BAOHAN        = 1;        // ´□□°□°¬□□
const int NEIQIE        = 2;        // □□□□
const int XIANJIAO      = 3;        // □□
const int WAIQIE        = 4;        // □□□□
const int XIANLI        = 5;        // □□□□
const double pi        = acos(-1.0)  //□□□□□
///////////////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////////
//□□□□¨□□□□
struct Point {              // ¶□□µ□□□-
    double x, y;
    double angle, dis;
    Point() {}
    Point(double x0, double y0): x(x0), y(y0) {}
};
struct Point3D {            //□□□µ□□□-
    double x, y, z;
    Point3D() {}
    Point3D(double x0, double y0, double z0): x(x0), y(y0), z(z0) {}
};
struct Line {               // ¶□□µ□□□□□□□
    Point p1, p2;
    Line() {}
    Line(Point p10, Point p20): p1(p10), p2(p20) {}
};
struct Line3D {             // □□□µ□□□□□□□
    Point3D p1, p2;
    Line3D() {}
    Line3D(Point3D p10, Point3D p20): p1(p10), p2(p20) {}
};
```

```
struct Rect {              // □ó¤¿□□□¾□□□ķ½·¨ w, h·□□□¿□□□□□
    double w, h;
 Rect() {}
 Rect(double _w,double _h) : w(_w),h(_h) {}
};
struct Rect_2 {             // ±□□¾□□□¬□□□½□□□±□□□(xl, yl)£¬□□□□□□□□±□□□(xh, yh)
    double xl, yl, xh, yh;
 Rect_2() {}
 Rect_2(double _xl,double _yl,double _xh,double _yh) : xl(_xl),yl(_yl),xh(_xh),yh(_yh)
    {}
};
struct Circle {            //□
 Point c;
 double r;
 Circle() {}
 Circle(Point _c,double _r) :c(_c),r(_r) {}
};
typedef vector<Point> Polygon;      // ¶□□¶□□□□
typedef vector<Point> Points;       // ¶□□µ🄵
typedef vector<Point3D> Points3D;   // □□□µ🄵
///////////////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////////
//»□±¾°¯□□□□
inline double max(double x,double y)
{
    return x > y ? x : y;
}
inline double min(double x, double y)
{
    return x > y ? y : x;
}
inline bool ZERO(double x)              // x == 0
{
    return (fabs(x) < EPS);
}
inline bool ZERO(Point p)               // p == 0
{
    return (ZERO(p.x) && ZERO(p.y));
}
inline bool ZERO(Point3D p)              // p == 0
{
    return (ZERO(p.x) && ZERO(p.y) && ZERO(p.z));
}
inline bool EQ(double x, double y)      // eqaul, x == y
{
    return (fabs(x - y) < EPS);
}
inline bool NEQ(double x, double y)     // not equal, x != y
{
    return (fabs(x - y) >= EPS);
}
inline bool LT(double x, double y)     // less than, x < y
{
    return ( NEQ(x, y) && (x < y) );
}
inline bool GT(double x, double y)     // greater than, x > y
{
    return ( NEQ(x, y) && (x > y) );
}
inline bool LEQ(double x, double y)     // less equal, x <= y
{
    return ( EQ(x, y) || (x < y) );
}
inline bool GEQ(double x, double y)     // greater equal, x >= y
{
    return ( EQ(x, y) || (x > y) );
}
// □□□£¡£¡
// □□□□□□¸□□□µ□°µ□¡µ□□□
// ±£□□□□□□□□□□□□□°□□□□□-0.000□□□□µ□□□□£¬
// □□□□□□□¸
// □□□□□´□□□¡£¡£¡£¡£¡
// □□□□□□□□□□£¬□¶¨□µ□□ô□¯□□½□□□□□□□£¡
inline double FIX(double x)
{
    return (fabs(x) < EPS) ? 0 : x;
}
///////////////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////////////////
//¶□□□-□□□□
bool operator==(Point p1, Point p2)
{
    return ( EQ(p1.x, p2.x) &&  EQ(p1.y, p2.y) );
}
bool operator!=(Point p1, Point p2)
{
    return ( NEQ(p1.x, p2.x) ||  NEQ(p1.y, p2.y) );
}
bool operator<(Point p1, Point p2)
{
    if (NEQ(p1.x, p2.x)) {
        return (p1.x < p2.x);
    } else {
        return (p1.y < p2.y);
    }
}
Point operator+(Point p1, Point p2)
{
    return Point(p1.x + p2.x, p1.y + p2.y);
}
Point operator-(Point p1, Point p2)
{
    return Point(p1.x - p2.x, p1.y - p2.y);
}
double operator*(Point p1, Point p2) // ¼□□□□□ p1 ¡□ p2
{
    return (p1.x * p2.y - p2.x * p1.y);
}
double operator&(Point p1, Point p2) { // ¼□□□□□ p1¡¤p2
    return (p1.x * p2.x + p1.y * p2.y);
}
double Norm(Point p) // ¼□□□□-pµ□ǵ
{
    return sqrt(p.x * p.x + p.y * p.y);
```

```
}
// °□□-p□□□½□□angle (»¡¶□□□)
// angle > 0±□□□□□□□□□□
// angle < 0±□□□□□□□□□
Point Rotate(Point p, double angle)
{
    Point result;
    result.x = p.x * cos(angle) - p.y * sin(angle);
    result.y = p.x * sin(angle) + p.y * cos(angle);
    return result;
}
//////////////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////////////////
//□□□□-□□□□
bool operator==(Point3D p1, Point3D p2)
{
    return ( EQ(p1.x, p2.x) && EQ(p1.y, p2.y) && EQ(p1.z, p2.z) );
}
bool operator<(Point3D p1, Point3D p2)
{
    if (NEQ(p1.x, p2.x)) {
        return (p1.x < p2.x);
    } else if (NEQ(p1.y, p2.y)) {
        return (p1.y < p2.y);
    } else {
        return (p1.z < p2.z);
    }
}
Point3D operator+(Point3D p1, Point3D p2)
{
    return Point3D(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z);
}
Point3D operator-(Point3D p1, Point3D p2)
{
    return Point3D(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);
}
Point3D operator*(Point3D p1, Point3D p2) // ¼□□□□□ p1 x p2
{
    return Point3D(p1.y * p2.z - p1.z * p2.y,
        p1.z * p2.x - p1.x * p2.z,
        p1.x * p2.y - p1.y * p2.x );
}
double operator&(Point3D p1, Point3D p2) { // ¼□□□□□ p1¡¤p2
    return (p1.x * p2.x + p1.y * p2.y + p1.z * p2.z);
}
double Norm(Point3D p) // ¼□□□□-pµ□ǵ
{
    return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}


//////////////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////////////
//µ□.□□□.□□□□□□
//
```

```
double Distance(Point p1, Point p2) //2µ□□l’□□□
{
 return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double Distance(Point3D p1, Point3D p2) //2µ□□l’□□□,□□□
{
 return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z));
}
double Distance(Point p, Line L) // □□□□□□□□□🄵□l’□□□□
{
    return ( fabs((p - L.p1) * (L.p2 - L.p1)) / Norm(L.p2 - L.p1) );
}
double Distance(Point3D p, Line3D L)// □□□□□¿□□□□🄵□l’□□□□
{
    return ( Norm((p - L.p1) * (L.p2 - L.p1)) / Norm(L.p2 - L.p1) );
}
bool OnLine(Point p, Line L) // □□□□□□□□□□□p□□□□□□□L□□
{
    return ZERO( (p - L.p1) * (L.p2 - L.p1) );
}
bool OnLine(Point3D p, Line3D L) // □□□□□□¿□□□□□p□□□□□□□L□□
{
    return ZERO( (p - L.p1) * (L.p2 - L.p1) );
}
int Relation(Point p, Line L) // ¼□□□□p□□□□Lµ□□□□□□ , · µ»□ONLINE,LEFT,RIGHT
{
    double res = (L.p2 - L.p1) * (p - L.p1);
    if (EQ(res, 0)) {
        return ONLINE;
    } else if (res > 0) {
        return LEFT;
    } else {
        return RIGHT;
    }
}
bool SameSide(Point p1, Point p2, Line L) // □□□□p1, p2□□□□□□□Lµ□□□
{
    double m1 = (p1 - L.p1) * (L.p2 - L.p1);
    double m2 = (p2 - L.p1) * (L.p2 - L.p1);
    return GT(m1 * m2, 0);
}
bool OnLineSeg(Point p, Line L) // □□□□□□□□□□□p□□□□□□□□□l□□
{
    return ( ZERO( (L.p1 - p) * (L.p2 - p) ) &&
        LEQ((p.x - L.p1.x)*(p.x - L.p2.x), 0) &&
        LEQ((p.y - L.p1.y)*(p.y - L.p2.y), 0) );
}
bool OnLineSeg(Point3D p, Line3D L) // □□□□□□¿□□□□□p□□□□□□□□□l□□
{
    return ( ZERO((L.p1 - p) * (L.p2 - p)) &&
        EQ( Norm(p - L.p1) + Norm(p - L.p2), Norm(L.p2 - L.p1)) );
}
Point SymPoint(Point p, Line L) // □□□□□□□□□□p¹□□□□□LµĶ□□□
{
    Point result;
    double a = L.p2.x - L.p1.x;
    double b = L.p2.y - L.p1.y;
    double t = ( (p.x - L.p1.x) * a + (p.y - L.p1.y) * b ) / (a*a + b*b);
    result.x = 2 * L.p1.x + 2 * a * t - p.x;
    result.y = 2 * L.p1.y + 2 * b * t - p.y;
    return result;
}
bool Coplanar(Points3D points) // □□□□¸□🄵□□□□□□□□²¿¹²□□
{
    int i;
    Point3D p;


    if (points.size() < 4) return true;
    p = (points[2] - points[0]) * (points[1] - points[0]);
    for (i = 3; i < points.size(); i++) {
        if (! ZERO(p & points[i]) ) return false;
    }
    return true;
}
bool LineIntersect(Line L1, Line L2) // □□□□□µ□}□□□□□□□
{
    return (! ZERO((L1.p1 - L1.p2)*(L2.p1 - L2.p2)) );  // □□□□□□
}
bool LineIntersect(Line3D L1, Line3D L2) // □□□□□□µ□}□□□□□□□
{
    Point3D p1 = L1.p1 - L1.p2;
    Point3D p2 = L2.p1 - L2.p2;
    Point3D p  = p1 * p2;
    if (ZERO(p)) return false;      // □□□□□□
    p = (L2.p1 - L1.p2) * (L1.p1 - L1.p2);
    return ZERO(p & L2.p2);         // □□□□□
}
bool LineSegIntersect(Line L1, Line L2) // □□□□□µ□}□□□□□□□□□□□
{
    return ( GEQ( max(L1.p1.x, L1.p2.x), min(L2.p1.x, L2.p2.x) ) &&
        GEQ( max(L2.p1.x, L2.p2.x), min(L1.p1.x, L1.p2.x) ) &&
        GEQ( max(L1.p1.y, L1.p2.y), min(L2.p1.y, L2.p2.y) ) &&
        GEQ( max(L2.p1.y, L2.p2.y), min(L1.p1.y, L1.p2.y) ) &&
        LEQ( ((L2.p1 - L1.p1) * (L1.p2 - L1.p1)) * ((L2.p2 -  L1.p1) * (L1.p2 - L1.p1))
    , 0 ) &&
        LEQ( ((L1.p1 - L2.p1) * (L2.p2 - L2.p1)) * ((L1.p2 -  L2.p1) * (L2.p2 - L2.p1))
    , 0 ) );
}
bool LineSegIntersect(Line3D L1, Line3D L2) // □□□□□□µ□}□□□□□□□□□□□
{
    // todo
    return true;
}
// ¼□□□}□□□□□□Ľ»µ□🄵½□□□□□□□□P□□µ»□
// ·µ»□{□□□□□□□□□□□□□□ù□□:  COLINE   -- ¹²□□  PARALLEL -- □□□  CROSS    -- □□
int CalCrossPoint(Line L1, Line L2, Point& P)
{
    double A1, B1, C1, A2, B2, C2;


    A1 = L1.p2.y - L1.p1.y;
    B1 = L1.p1.x - L1.p2.x;
    C1 = L1.p2.x * L1.p1.y - L1.p1.x * L1.p2.y;


    A2 = L2.p2.y - L2.p1.y;
    B2 = L2.p1.x - L2.p2.x;
    C2 = L2.p2.x * L2.p1.y - L2.p1.x * L2.p2.y;


    if (EQ(A1 * B2, B1 * A2))    {
        if (EQ( (A1 + B1) * C2, (A2 + B2) * C1 )) {
            return COLINE;
        } else {
            return PARALLEL;
        }
    } else {
        P.x = (B2 * C1 - B1 * C2) / (A2 * B1 - A1 * B2);
        P.y = (A1 * C2 - A2 * C1) / (A2 * B1 - A1 * B2);
        return CROSS;
    }
}
// ¼□□□}□□□□□□□L’»µ□🄵½□□□□□□□P□□µ»□
                 □
//  · µ»□{□□□□□□□□□□□□□□□ù□□ COLINE   -- ¹²□□  PARALLEL -- □□□  CROSS    -- □□ NONCOPLANAR
    -- ²»¹«□□
int CalCrossPoint(Line3D L1, Line3D L2, Point3D& P)
{
    // todo
    return 0;
}
// ¼□□□□Pµ½□□Lµ□□□□µ□
              □
Point NearestPointToLine(Point P, Line L)
{
    Point result;
    double a, b, t;


    a = L.p2.x - L.p1.x;
    b = L.p2.y - L.p1.y;
    t = ( (P.x - L.p1.x) * a + (P.y - L.p1.y) * b ) / (a * a + b * b);


    result.x = L.p1.x + a * t;
    result.y = L.p1.y + b * t;
    return result;
}
// ¼□□□□Pµ½□□□Lµ□□□□µ□
Point NearestPointToLineSeg(Point P, Line L)
{
    Point result;
    double a, b, t;


    a = L.p2.x - L.p1.x;
    b = L.p2.y - L.p1.y;
    t = ( (P.x - L.p1.x) * a + (P.y - L.p1.y) * b ) / (a * a + b * b);


    if ( GEQ(t, 0) && LEQ(t, 1) ) {
        result.x = L.p1.x + a * t;
        result.y = L.p1.y + b * t;
    } else {
        if ( Norm(P - L.p1) < Norm(P - L.p2) ) {
            result = L.p1;
        } else {
            result = L.p2;
        }
    }
    return result;
}
// ¼□□□□□□L1µ½□□□L2µ□□□□□□
double MinDistance(Line L1, Line L2)
{
    double d1, d2, d3, d4;


    if (LineSegIntersect(L1, L2)) {
        return 0;
    } else {
        d1 = Norm( NearestPointToLineSeg(L1.p1, L2) - L1.p1 );
        d2 = Norm( NearestPointToLineSeg(L1.p2, L2) - L1.p2 );
        d3 = Norm( NearestPointToLineSeg(L2.p1, L1) - L2.p1 );
        d4 = Norm( NearestPointToLineSeg(L2.p2, L1) - L2.p2 );

        return min( min(d1, d2), min(d3, d4) );
    }
}
// □□□□}□¬ļ□□□
                  □
//  ·µ»□ŧ0□□Pi¼□Ļ¡¶□
double Inclination(Line L1, Line L2)
{
    Point u = L1.p2 - L1.p1;
    Point v = L2.p2 - L2.p1;
    return acos( (u & v) / (Norm(u)*Norm(v)) );
}
// □□□□□}□¬ļ□□□
                  □
//  ·µ»□ŧ0□□Pi¼□Ļ¡¶□
double Inclination(Line3D L1, Line3D L2)
{
    Point3D u = L1.p2 - L1.p1;
    Point3D v = L2.p2 - L2.p1;
    return acos( (u & v) / (Norm(u)*Norm(v)) );
}
/////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////
// □□□}¸□□□□□□□□□□
// □□□□□□«□□□□□
bool Intersect(Rect_2 r1, Rect_2 r2)
{
    return ( max(r1.xl, r2.xl) < min(r1.xh, r2.xh) &&
             max(r1.yl, r2.yl) < min(r1.yh, r2.yh) );
}
// □□□□□□r2□□□□□□□□□□□□□□r1□□
// r2¿□□□□□□□□□□□□
//·¢□□□4µ□□³□ķ½·¨¹□²»□□OJ□□□□□□□□□□□□□
//□□□□□□□□□□°µ□□□□
bool IsContain(Rect r1, Rect r2)      //¾□□□□w>h
 {
     if(r1.w >r2.w && r1.h > r2.h) return true;
     else
     {
        double r = sqrt(r2.w*r2.w + r2.h*r2.h) / 2.0;
        double alpha = atan2(r2.h,r2.w);
        double sita = asin((r1.h/2.0)/r);
        double x = r * cos(sita - 2*alpha);
        double y = r * sin(sita - 2*alpha);
        if(x < r1.w/2.0 && y < r1.h/2.0 && x > 0 && y > -r1.h/2.0) return true;
        else return false;
     }
}
/////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////
//□
Point Center(const Circle & C) //□□□
{
    return C.c;
}


double Area(const Circle &C)
{
 return pi*C.r*C.r;
}


double CommonArea(const Circle & A, const Circle & B) //}¸□□µĹ«¹²□□□
{
    double area = 0.0;
    const Circle & M = (A.r > B.r) ? A : B;
    const Circle & N = (A.r > B.r) ? B : A;
    double D = Distance(Center(M), Center(N));
    if ((D < M.r + N.r) && (D > M.r - N.r))
    {
        double cosM = (M.r * M.r + D * D - N.r * N.r) / (2.0 * M.r * D);
        double cosN = (N.r * N.r + D * D - M.r * M.r) / (2.0 * N.r * D);
        double alpha = 2.0 * acos(cosM);
        double beta  = 2.0 * acos(cosN);
        double TM = 0.5 * M.r * M.r * sin(alpha);
        double TN = 0.5 * N.r * N.r * sin(beta);
        double FM = (alpha / (2*pi)) * Area(M);
        double FN = (beta / (2*pi)) * Area(N);
        area = FM + FN - TM - TN;
    }
    else if (D <= M.r - N.r)
    {
        area = Area(N);
    }
    return area;
}

bool IsInCircle(const Circle & C, const Rect_2 & rect)//□□□□□□□□□)□□□□□²»□□□□□□□□)
{
    return (GT(C.c.x - C.r, rect.xl)
  &&  LT(C.c.x + C.r, rect.xh)
  &&  GT(C.c.y - C.r, rect.yl)
  &&  LT(C.c.y + C.r, rect.yh));
```

```
}


//□□□2□µ□□□ù□□
// · µ»□:
//BAOHAN   = 1;         // ´□□°□°¬□□
//NEIQIE   = 2;         // □□□□
//XIANJIAO = 3;         // □□
//WAIQIE   = 4;         // □□□□
//XIANLI   = 5;         // □□□□
int CirCir(const Circle &c1, const Circle &c2)//□□□2□µ□□□ù□□
{
 double dis = Distance(c1.c,c2.c);
 if(LT(dis,fabs(c1.r-c2.r))) return BAOHAN;
 if(EQ(dis,fabs(c1.r-c2.r))) return NEIQIE;
 if(LT(dis,c1.r+c2.r) && GT(dis,fabs(c1.r-c2.r))) return XIANJIAO;
 if(EQ(dis,c1.r+c2.r)) return WAIQIE;
 return XIANLI;
}
////////////////////////////////////////////////////////////////////////


int main()
{
 return 0;
}
```

0.12 结构体表示几何图形

```
//¼□□F°□(¶□□)
#include <cmath>
#include <cstdio>
#include <algorithm>
using namespace std;


typedef double TYPE;
#define Abs(x) (((x)>0)?(x):(-(x)))
#define Sgn(x) (((x)<0)?(-1):(1))
#define Max(a,b) (((a)>(b))?(a):(b))
#define Min(a,b) (((a)<(b))?(a):(b))
#define Epsilon 1e-8
#define Infinity 1e+10
#define PI acos(-1.0)//3.14159265358979323846
TYPE Deg2Rad(TYPE deg){return (deg * PI / 180.0);}
TYPE Rad2Deg(TYPE rad){return (rad * 180.0 / PI);}
TYPE Sin(TYPE deg){return sin(Deg2Rad(deg));}
TYPE Cos(TYPE deg){return cos(Deg2Rad(deg));}
TYPE ArcSin(TYPE val){return Rad2Deg(asin(val));}
TYPE ArcCos(TYPE val){return Rad2Deg(acos(val));}
TYPE Sqrt(TYPE val){return sqrt(val);}


//µ□
struct POINT
{
  TYPE x;
  TYPE y;
  POINT() : x(0), y(0) {};
  POINT(TYPE _x_, TYPE _y_) : x(_x_), y(_y_) {};
};
// }¸□□Ľ□□□
TYPE Distance(const POINT & a, const POINT & b)
{
  return Sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
//□□□
struct SEG
{
  POINT a; //□□□
  POINT b; //□□□
  SEG() {};
  SEG(POINT _a_, POINT _b_):a(_a_),b(_b_) {};
};
//{)□□µ□□)
struct LINE
{
  POINT a;
  POINT b;
  LINE() {};
  LINE(POINT _a_, POINT _b_) : a(_a_), b(_b_) {};
};
//)□□□□°□□)
struct LINE2
{
  TYPE A,B,C;
  LINE2() {};
  LINE2(TYPE _A_, TYPE _B_, TYPE _C_) : A(_A_), B(_B_), C(_C_) {};
};


//}µ□□»¯□°□□
LINE2 Line2line(const LINE & L) // y=kx+c k=y/x
{
  LINE2 L2;
  L2.A = L.b.y - L.a.y;
  L2.B = L.a.x - L.b.x;
  L2.C = L.b.x * L.a.y - L.a.x * L.b.y;
  return L2;
}


// □□□÷µ»□□□ Ax + By + C =0 µ□□□□
void Coefficient(const LINE & L, TYPE & A, TYPE & B, TYPE & C)
{
  A = L.b.y - L.a.y;
  B = L.a.x - L.b.x;
  C = L.b.x * L.a.y - L.a.x * L.b.y;
}
void Coefficient(const POINT & p,const TYPE a,TYPE & A,TYPE & B,TYPE & C)
{
  A = Cos(a);
  B = Sin(a);
  C = - (p.y * B + p.x * A);
}
/□□□(£¬µ␌(□□
bool IsEqual(TYPE a, TYPE b)
```

```
{
  return (Abs(a - b) <Epsilon);
}
bool IsEqual(const POINT & a, const POINT & b)
{
  return (IsEqual(a.x, b.x) && IsEqual(a.y, b.y));
}
bool IsEqual(const LINE & A, const LINE & B)
{
  TYPE A1, B1, C1;
  TYPE A2, B2, C2;
  Coefficient(A, A1, B1, C1);
  Coefficient(B, A2, B2, C2);
  return IsEqual(A1 * B2, A2 * B1) && IsEqual(A1 * C2, A2 * C1) && IsEqual(B1 * C2, B2
    * C1);
}
// ¾□□□
struct RECT
{
  POINT a; // □□□µ□
  POINT b; // □□□□□
  RECT() {};
  RECT(const POINT & _a_, const POINT & _b_) { a = _a_; b = _b_; }
};


//¾□□□¯±□□
RECT Stdrect(const RECT & q)
{
  TYPE t;
  RECT p=q;
  if(p.a.x > p.b.x) swap(p.a.x , p.b.x);
  if(p.a.y > p.b.y) swap(p.a.y , p.b.y);
  return p;
}


//¸□¾□□±🄵□«□□□ı□
SEG Edge(const RECT & rect, int idx)
{
  SEG edge;
  while (idx < 0) idx += 4;
  switch (idx % 4)
  {
  case 0: //□±□
    edge.a = rect.a;
    edge.b = POINT(rect.b.x, rect.a.y);
    break;
  case 1: //□□□
    edge.a = POINT(rect.b.x, rect.a.y);
    edge.b = rect.b;
    break;
  case 2: //□□□
    edge.a = rect.b;
    edge.b = POINT(rect.a.x, rect.b.y);
    break;
  case 3: //□□□
    edge.a = POINT(rect.a.x, rect.b.y);
    edge.b = rect.a;
    break;
  default:
    break;
  }
  return edge;
}


//¾□□□□□□□
TYPE Area(const RECT & rect)
{
  return (rect.b.x - rect.a.x) * (rect.b.y - rect.a.y);
}


//}¸□□□□Ĺ«¹²□□□
TYPE CommonArea(const RECT & A, const RECT & B)
{
  TYPE area = 0.0;
  POINT LL(Max(A.a.x, B.a.x), Max(A.a.y, B.a.y));
  POINT UR(Min(A.b.x, B.b.x), Min(A.b.y, B.b.y));
  if( (LL.x <= UR.x) && (LL.y <= UR.y) )
  {
    area = Area(RECT(LL, UR));
  }
  return area;
}
//□□□□□□□□□)□□□□□²»□□□□□□□□)
bool IsInCircle(const CIRCLE & circle, const RECT & rect)
{
  return (circle.x - circle.r > rect.a.x) &&
    (circle.x + circle.r < rect.b.x) &&
    (circle.y - circle.r > rect.a.y) &&
    (circle.y + circle.r < rect.b.y);
}


//□□□□□□□□□□□□□□(²»□□□□□□□□)
bool IsInRect(const CIRCLE & circle, const RECT & rect)
{
  POINT c,d;
  c.x=rect.a.x; c.y=rect.b.y;
  d.x=rect.b.x; d.y=rect.a.y;
  return (Distance( Center(circle) , rect.a ) < circle.r) &&
    (Distance( Center(circle) , rect.b ) < circle.r) &&
    (Distance( Center(circle) , c ) < circle.r) &&
    (Distance( Center(circle) , d ) < circle.r);
}


//□□□□□□□□□□□□□□□□(²»□□□□□□□□)
bool Isoutside(const CIRCLE & circle, const RECT & rect)
{
  POINT c,d;
  c.x=rect.a.x; c.y=rect.b.y;
  d.x=rect.b.x; d.y=rect.a.y;
  return (Distance( Center(circle) , rect.a ) > circle.r) &&
    (Distance( Center(circle) , rect.b ) > circle.r) &&
    (Distance( Center(circle) , c ) > circle.r) &&
    (Distance( Center(circle) , d ) > circle.r) &&
    (rect.a.x > circle.x || circle.x > rect.b.x || rect.a.y > circle.y || circle.y > rect.b.y) ||
    ((circle.x - circle.r > rect.b.x) ||
    (circle.x + circle.r < rect.a.x) ||
    (circle.y - circle.r > rect.b.y) ||
    (circle.y + circle.r < rect.a.y));
}
```

1 Codes

1.1 最小圆覆盖

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


const int maxn = 1005;
//const double eps = 1e-6;


struct TPoint
{
    double x, y;
    TPoint operator-(TPoint &a)
    {
        TPoint p1;
        p1.x = x - a.x;
        p1.y = y - a.y;
        return p1;
    }
};


struct TCircle
{
    double r;
    TPoint centre;
};


struct TTriangle
{
    TPoint t[3];
};


TCircle c;
TPoint a[maxn];


double distance(TPoint p1, TPoint p2)
{
    TPoint p3;
    p3.x = p2.x - p1.x;
    p3.y = p2.y - p1.y;
    return sqrt(p3.x * p3.x + p3.y * p3.y);
}


double triangleArea(TTriangle t)
{
    TPoint p1, p2;
    p1 = t.t[1] - t.t[0];
    p2 = t.t[2] - t.t[0];
    return fabs(p1.x * p2.y - p1.y * p2.x) / 2;
}
```

```


TCircle circumcircleOfTriangle(TTriangle t)
{
    //□□½□□□□□□□□
    TCircle tmp;
    double a, b, c, c1, c2;
    double xA, yA, xB, yB, xC, yC;
    a = distance(t.t[0], t.t[1]);
    b = distance(t.t[1], t.t[2]);
    c = distance(t.t[2], t.t[0]);
    //¸□¾□S = a * b * c / R / 4;□□🄵R
    tmp.r = a * b * c / triangleArea(t) / 4;

    xA = t.t[0].x; yA = t.t[0].y;
    xB = t.t[1].x; yB = t.t[1].y;
    xC = t.t[2].x; yC = t.t[2].y;
    c1 = (xA * xA + yA * yA - xB * xB - yB * yB) / 2;
    c2 = (xA * xA + yA * yA - xC * xC - yC * yC) / 2;

    tmp.centre.x = (c1 * (yA - yC) - c2 * (yA - yB)) /
         ((xA - xB) * (yA - yC) - (xA - xC) * (yA - yB));
    tmp.centre.y = (c1 * (xA - xC) - c2 * (xA - xB)) /
         ((yA - yB) * (xA - xC) - (yA - yC) * (xA - xB));

    return tmp;
}


TCircle MinCircle2(int tce, TTriangle ce)
{
    TCircle tmp;
    if(tce == 0) tmp.r = -2;
    else if(tce == 1)
    {
        tmp.centre = ce.t[0];
        tmp.r = 0;
    }
    else if(tce == 2)
    {
        tmp.r = distance(ce.t[0], ce.t[1]) / 2;
        tmp.centre.x = (ce.t[0].x + ce.t[1].x) / 2;
        tmp.centre.y = (ce.t[0].y + ce.t[1].y) / 2;
    }
    else if(tce == 3) tmp = circumcircleOfTriangle(ce);
    return tmp;
}


void MinCircle(int t, int tce, TTriangle ce)
{
    int i, j;
    TPoint tmp;
    c = MinCircle2(tce, ce);
    if(tce == 3) return;
    for(i = 1;i <= t;i++)
    {
        if(distance(a[i], c.centre) > c.r)
        {
            ce.t[tce] = a[i];
            MinCircle(i - 1, tce + 1, ce);
            tmp = a[i];
            for(j = i;j >= 2;j--)
            {
                a[j] = a[j - 1];
            }
            a[1] = tmp;
        }
    }
}


void run(int n)
{
    TTriangle ce;
    int i;
    MinCircle(n, 0, ce);
    printf("%.2lf %.2lf %.2lf\n", c.centre.x, c.centre.y, c.r);
}


int main()
{
    freopen("circle.in", "r", stdin);
    freopen("out.txt", "w", stdout);
    int n;
    while(scanf("%d", &n) != EOF && n)
    {
        for(int i = 1;i <= n;i++)
            scanf("%lf%lf", &a[i].x, &a[i].y);
        run(n);
    }
    return 0;
}
```

1.2 直线旋转

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


#define pi acos(-1.0)
#define eps 1e-6
#define inf 1e250
#define Maxn 10005


typedef struct TPoint
{
    double x, y;
}TPoint;


typedef struct TPolygon
{
    TPoint p[Maxn];
    int n;
```

```
}TPolygon;


typedef struct TLine
{
    double a, b, c;
}TLine;


double max(double a, double b)
{
    if(a > b) return a;
    return b;
}


double min(double a, double b)
{
    if(a < b) return a;
    return b;
}


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


TLine lineFromSegment(TPoint p1, TPoint p2)
{
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}


double polygonArea(TPolygon p)
{
    int i, n;
    double area;
    n = p.n;
    area = 0;
    for(i = 1;i <= n;i++)
        area += (p.p[i - 1].x * p.p[i % n].y - p.p[i % n].x * p.p[i - 1].y);


    return area / 2;
}


void ChangeClockwise(TPolygon &polygon)
{
    TPoint tmp;
    int i;
    for(i = 0;i <= (polygon.n - 1) / 2;i++)
    {
        tmp = polygon.p[i];
        polygon.p[i] = polygon.p[polygon.n - 1 - i];
        polygon.p[polygon.n - 1 - i] = tmp;
    }
}


double disPointToSeg(TPoint p1, TPoint p2, TPoint p3)
{
    double a = distance(p1, p2);
    double b = distance(p1, p3);
    double c = distance(p2, p3);
    if(fabs(a + b - c) < eps) return 0;
    if(fabs(a + c - b) < eps || fabs(b + c - a) < eps) return min(a, b);
    double t1 = -a * a + b * b + c * c;
    double t2 = a * a - b * b + c * c;
    if(t1 <= 0 || t2 <= 0) return min(a, b);

    TLine l1 = lineFromSegment(p2, p3);
    return fabs(l1.a * p1.x + l1.b * p1.y + l1.c) / sqrt(l1.a * l1.a + l1.b * l1.b);
}


double disPallSeg(TPoint p1, TPoint p2, TPoint p3, TPoint p4)
{
    return min(min(disPointToSeg(p1, p3, p4), disPointToSeg(p2, p3, p4)),
     min(disPointToSeg(p3, p1, p2), disPointToSeg(p4, p1, p2)));
}


double angle(TPoint p1, TPoint p2, double SlewRate)
{
    double ang, tmp;
    TPoint p;
    p.x = p2.x - p1.x;
    p.y = p2.y - p1.y;
    if(fabs(p.x) < eps)
    {
        if(p.y > 0) ang = pi / 2;
        else ang = 3 * pi / 2;
    }
    else
    {
        ang = atan(p.y / p.x);
        if(p.x < 0) ang += pi;
    }
    while(ang < 0) ang += 2 * pi;
    if(ang >= pi) SlewRate += pi;
    if(ang > SlewRate) tmp = ang - SlewRate;
    else tmp = pi - (SlewRate - ang);
    while(tmp >= pi) tmp -= pi;
    if(fabs(tmp - pi) < eps) tmp = 0;
    return tmp;
}


int main()
{
    int n, m, i;
    TPolygon polygon1, polygon2;
    double ymin1, ymax2, ans, d;
    int k1, k2;
    while(scanf("%d%d", &n, &m) && n)
    {
        polygon1.n = n;
        polygon2.n = m;
        for(i = 0;i < n;i++)
            scanf("%lf%lf", &polygon1.p[i].x, &polygon1.p[i].y);
        for(i = 0;i < m;i++)
            scanf("%lf%lf", &polygon2.p[i].x, &polygon2.p[i].y);
        if(polygonArea(polygon1) < 0) ChangeClockwise(polygon1);
        if(polygonArea(polygon2) < 0) ChangeClockwise(polygon2);
        ymin1 = inf, ymax2 = -inf;
        for(i = 0;i < n;i++)
            if(polygon1.p[i].y < ymin1) ymin1 = polygon1.p[i].y , k1 = i;
        for(i = 0;i < m;i++)
            if(polygon2.p[i].y > ymax2) ymax2 = polygon2.p[i].y , k2 = i;
        double SlewRate = 0;
        double angle1, angle2;
        ans = inf;
        double Slope = 0;
        while(Slope <= 360)
        {
            while(SlewRate >= pi) SlewRate -= pi;
            if(fabs(pi - SlewRate) < eps) SlewRate = 0;
            angle1 = angle(polygon1.p[k1], polygon1.p[(k1 + 1) % n], SlewRate);
            angle2 = angle(polygon2.p[k2], polygon2.p[(k2 + 1) % m], SlewRate);
            if(fabs(angle1 - angle2) < eps)
            {
                d = disPallSeg(polygon1.p[k1], polygon1.p[(k1 + 1) % n], polygon2.p[k2], polygon2.p[(k2 + 1) % m]);
                if(d < ans) ans = d;
                k1++;
                k1 %= n;
                k2++;
                k2 %= m;
                SlewRate += angle1;
                Slope += angle1;
            }
            else if(angle1 < angle2)
            {
                d = disPointToSeg(polygon2.p[k2], polygon1.p[k1], polygon1.p[(k1 + 1) % n]);
                if(d < ans) ans = d;
                k1++;
                k1 %= n;
                SlewRate += angle1;
                Slope += angle1;
            }
            else
            {
                d = disPointToSeg(polygon1.p[k1], polygon2.p[k2], polygon2.p[(k2 + 1) % m]);
                if(d < ans) ans = d;
                k2++;
                k2 %= m;
                SlewRate += angle2;
                Slope += angle2;
            }
        }
        printf("%.5lf\n", ans);
    }
    return 0;
}
```

1.3 扇形重心

```
//Xc = 2*R*sinA/3/A
//A□□□Ľ□□□°□
#include <stdio.h>
#include <math.h>
int main()
{
    double r, angle;
    while(scanf("%lf%lf", &r, &angle) != EOF){
        angle /= 2;
        printf("%.6lf\n", 2 * r * sin(angle) / 3 / angle);
    }
    return 0;
}
```

1.4 根据经度纬度求球面距离

```
/*
¼□□□□□□□□□□□壬
□□□□□□□□µ□Ľ¶□□lambda,□¶□□phi£¬
□□□□□Ŀ□□□□±□□□
x=cos(phi)*cos(lambda)
y=cos(phi)*sin(lambda)
z=sin(phi)
□□□□□□□}µ□Ŀ□□□□±□□□(x1,y1,z1),(x2,y2,z2)
□□□□🄵□R*sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)+(z2-z1)*(z2-z1)),
□□□□□□ļ□□□
A = acos(x1 * x2 + y1 * y2 + z1 * z2)£¬
□□}µ□□□□□ A * R£¬□□□□R□µ□□□□¾□°🄵6371
*/


/*
□□□□□□±🄵□³□□□🄵R£¬µ«□□□□□□□□□□¬□□□□□□¶¼û□□□
□□□□□□□□□□□□□±□□□□¸°°;¶□□¶□Ĺ涨□□□
pku_3407
*/
#include <stdio.h>
#include <math.h>


const double pi = acos(-1.0);


struct TPoint
{
   double x, y, z;
};
```

```

int main()
{
    double w1, wm1, j1, jm1, wd1, wd2;
    double w2, wm2, j2, jm2, jd1, jd2;
    TPoint p1, p2;
    char chr1, chr2;
    while(scanf("%lf%lf ", &w1, &wm1) != EOF){
        scanf("%c ", &chr1);
        scanf("%lf %lf %c", &j1, &jm1, &chr2);
        wd1 = (w1 + wm1 / 60) * pi / 180;
        jd1 = (j1 + jm1 / 60) * pi / 180;
        if(chr1 == 'S') wd1 *= -1.0;
        if(chr2 == 'W') jd1 *= -1.0;
        p1.x = cos(wd1) * cos(jd1);
        p1.y = cos(wd1) * sin(jd1);
        p1.z = sin(wd1);
        scanf("%lf %lf %c %lf %lf %c", &w2, &wm2, &chr1, &j2, &jm2, &chr2);
        wd2 = (w2 + wm2 / 60) * pi / 180;
        jd2 = (j2 + jm2 / 60) * pi / 180;
        if(chr1 == 'S') wd2 *= -1.0;
        if(chr2 == 'W') jd2 *= -1.0;
        p2.x = cos(wd2) * cos(jd2);
        p2.y = cos(wd2) * sin(jd2);
        p2.z = sin(wd2);
        double a = acos(p1.x * p2.x + p1.y * p2.y + p1.z * p2.z);
        printf("%.3lf\n", a * 6370.0);
    }
    return 0;
}
```

1.5 多边形重心

```
/*
□□Ŀ□□□□
□□□¸□□□□□□□□□□N¶□□□□(3 <= N <= 1000000)£¬¿□□¼□¿□□□□£¬µ«û□□□□□ɛ🄵
□□□□□□N¸□)□□□□□□□□□□□□□□□)¶¥µ□□□□±□ɛ£¬µ□j¸□□±□□£¨Xi £¬ Yi £©£¬□□□□
□□ (|Xi|, |Yi| <= 20000)£¬□□□□□□□□□□□□□□□□□ġ£
  ½□□□□³□°
  ´□□1¸·□l½□□i, i+1¸□□□□□½□□□Ti,1 < i < n,
  □¹²n-2¸□□□½□□□□□°□□□□l□□□□¸¬·□□□□ÿ¸□□□½□□□□□□□Si£¬
  □□□□□□¸□□□□□□□□□
  ¸□¾□□□□□□□µã°n¸□□(xi,yi)ÿ¸□□□-□□mi,□□□□□□□□
  X = (x1*M1+x2*M2+...+xn*Mn) / (M1+M2+....+Mn)
  Y = (y1*M1+y2*M2+...+yn*Mn) / (M1+M2+....+Mn)
  □□□□□□□õ□□□□°
  □□3µ□□□□□½□□□□□□□□£¬□□□□µ□□□p[0].x, p[0].y p[1].x, p[1].y p[1].x, p[1].y
  □□□s =[ p[0].x*p[1].y-p[1].x*p[0].y + p[1].x*p[2].y-p[2].x*p[1].y + p[2].x*p[0].y-p
    [0].x*p[2].y ] / 2 ,
  □□□□□□3¸□□□□□□□□□□ɛ£¬□□□□□¸°¡£
  □□3µ□□□□□□ǵ¬x = (p[0].x+p[1].x+p[2].x)/3.0 , y = (p[0].y+p[1].y+p[2].y)/3.0
  □□□□□□□□□Ĺ□³□□□¬²»□□□¿¾□□□□□□□□□□□□□□□□🄵□□□□□□🄵□□□°□□□□□□□¬
  »¹□□□□ □µ□□□»±□□□□□ÿ¸□□□□½□□□□□□□□□¶¼³□□□3£¬¿□□□□□□□□□□□□

*/
/*fzu_1132*/
#include <stdio.h>
```

```
#include <math.h>


typedef struct TPoint
{
    double x;
    double y;
}TPoint;


double triangleArea(TPoint p0, TPoint p1, TPoint p2)
{
    //□□□□½□□□□□□¸□□□±□□□□□½□□□□□□□
    double k = p0.x * p1.y + p1.x * p2.y
        + p2.x * p0.y - p1.x * p0.y
        - p2.x * p1.y - p0.x * p2.y;
    //if(k >= 0) return k / 2;
//  else return -k / 2;
       return k / 2;
}


int main()
{
    int i, n, test;
    TPoint p0, p1, p2, center;
    double area, sumarea, sumx, sumy;
    scanf("%d", &test);
    while(test--){
        scanf("%d", &n);
        scanf("%lf%lf", &p0.x, &p0.y);
        scanf("%lf%lf", &p1.x, &p1.y);
        sumx = 0;
        sumy = 0;
        sumarea = 0;
        for(i = 2;i < n;i++){
            scanf("%lf%lf", &p2.x, &p2.y);
            center.x = p0.x + p1.x + p2.x;
            center.y = p0.y + p1.y + p2.y;
            area =  triangleArea(p0, p1, p2);
            sumarea += area;
            sumx += center.x * area;
            sumy += center.y * area;
            p1 = p2;
        }
        printf("%.2lf %.2lf\n", sumx / sumarea / 3, sumy / sumarea / 3);
    }
    return 0;
}
```

1.6 存在一个平面把两堆点分开

```
#include <stdio.h>
struct point
{
    double x, y, z;
}pa[201], pb[201];
int main()
```

```
{
    int n, m, i;
    while (scanf("%d", &n), n != -1)
    {
        for (i = 0; i < n; i++)
            scanf("%lf%lf%lf", &pa[i].x, &pa[i].y, &pa[i].z);
        scanf("%d", &m);
        for (i = 0; i < m; i++)
            scanf("%lf%lf%lf", &pb[i].x, &pb[i].y, &pb[i].z);
        int cnt = 0, finish = 0;
        double a = 0, b = 0, c = 0, d = 0;
        while (cnt < 100000 && !finish)
        {
            finish = 1;
            for (i = 0; i < n; i++)
                if (a * pa[i].x + b * pa[i].y + c * pa[i].z + d > 0)
                {
                    a -= pa[i].x;
                    b -= pa[i].y;
                    c -= pa[i].z;
                    d -= 3;
                    finish = 0;
                }
            for (i = 0; i < m; i++)
                if (a * pb[i].x + b * pb[i].y + c * pb[i].z + d <= 0)
                {
                    a += pb[i].x;
                    b += pb[i].y;
                    c += pb[i].z;
                    d += 3;
                    finish = 0;
                }
            cnt++;
        }
        printf("%lf %lf %lf %lf\n", a, b, c, d);
    }
    return 0;
}
```

1.7 判断多边形的核是否存在

```
/*¶□□□□ĺ□*/
#include <stdio.h>
#include <math.h>


#define Maxn 3005
const double eps = 1e-10;


typedef struct TPodouble
{
    double x;
    double y;
}TPoint;


typedef struct TPolygon
```

```
{
    TPoint p[Maxn];
    int n;
};


typedef struct TLine
{
    double a, b, c;
}TLine;


bool same(TPoint p1, TPoint p2)
{
    if(p1.x != p2.x) return false;
    if(p1.y != p2.y) return false;
    return true;
}

double multi(TPoint p1, TPoint p2, TPoint p0)
{
    //□□□-[p0, p1], [p0, p2]µ□□□
    //p0□□¥µ□
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
    //□□□□µ□□□0£¬□□□□□□□µ🄵□□
    //□□□□´□□□0£¬□□p0p2□□p0p1µ□□□□□🄵□□
    //□□□□□□□□0£¬□□p0p2□□p0p1µ□□□□🄵□□
}


TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //□□□□□□□□□, ·µ»□□□□³½□□□¸□□□
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}


TPoint LineInter(TLine l1, TLine l2)
{
    //□□}□ý»µ□□□□±□
    TPoint tmp;
    double a1 = l1.a;
    double b1 = l1.b;
    double c1 = l1.c;
    double a2 = l2.a;
    double b2 = l2.b;
    double c2 = l2.c;
    //□□□□□□□b1 = 0
    if(fabs(b1) < eps){
        tmp.x = -c1 / a1;
        tmp.y = (-c2 - a2 * tmp.x) / b2;
    }
    else{
        tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
        tmp.y = (-c1 - a1 * tmp.x) / b1;
    }
    return tmp;
}


TPolygon Cut_polygon(TPoint p1, TPoint p2, TPolygon polygon)
{
    TPolygon new_polygon;
    TPoint interp;
    TLine l1, l2;
    int i, j;
    double t1, t2;
    new_polygon.n = 0;
    for(i = 0;i <= polygon.n - 1;i++){
        t1 = multi(p2, polygon.p[i], p1);
        t2 = multi(p2, polygon.p[i + 1], p1);
        if(fabs(t1) < eps || fabs(t2) < eps){
            if(fabs(t1) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i];
            if(fabs(t2) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
        }
        else if(t1 < 0 && t2 < 0){
            new_polygon.p[new_polygon.n++] = polygon.p[i];
            new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
        }
        else if(t1 * t2  < 0){
            l1 = lineFromSegment(p1, p2);
            l2 = lineFromSegment(polygon.p[i], polygon.p[i + 1]);
            interp = LineInter(l1, l2);
            if(t1 < 0) {
                new_polygon.p[new_polygon.n++] = polygon.p[i];
                new_polygon.p[new_polygon.n++] = interp;
            }
            else {
                new_polygon.p[new_polygon.n++] = interp;
                new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
            }
        }
    }
    polygon.n = 0;
    if(new_polygon.n == 0) return polygon;
    polygon.p[polygon.n++] = new_polygon.p[0];
    for(i = 1;i < new_polygon.n;i++){
        if(!same(new_polygon.p[i], new_polygon.p[i - 1])){
            polygon.p[polygon.n++] = new_polygon.p[i];
        }
    }
    if(polygon.n != 1 && same(polygon.p[polygon.n - 1], polygon.p[0])) polygon.n--;
    polygon.p[polygon.n] = polygon.p[0];
    return polygon;
}


double polygonArea(TPolygon p)
{
    //□□¶□□□□□□□±□□□□□□□□
    int i, n;
    double area;
    n = p.n;
    area = 0;
    for(i = 1;i <= n;i++){
        area += (p.p[i - 1].x * p.p[i % n].y - p.p[i % n].x * p.p[i - 1].y);
    }
    return area / 2;
}


void ChangeClockwise(TPolygon &polygon)
{
    TPoint tmp;
    int i;
    for(i = 0;i <= (polygon.n - 1) / 2;i++){
        tmp = polygon.p[i];
        polygon.p[i] = polygon.p[polygon.n - 1 - i];
        polygon.p[polygon.n - 1 - i] = tmp;
    }
}


int main()
{
    int test, i, j;
    double area;
    TPolygon polygon, new_polygon;
    scanf("%d", &test);
    while(test--){
        scanf("%d", &polygon.n);
        for(i = 0;i <= polygon.n - 1;i++){
            scanf("%lf%lf", &polygon.p[i].x, &polygon.p[i].y);
        }
        /*□□□□□□□□□□□¯«□□□□□*/
        if(polygonArea(polygon) > 0) ChangeClockwise(polygon);
        polygon.p[polygon.n] = polygon.p[0];
        new_polygon = polygon;
        for(i = 0;i <= polygon.n - 1;i++){
            new_polygon = Cut_polygon(polygon.p[i], polygon.p[i + 1], new_polygon);
        }
        area = polygonArea(new_polygon);
        if(area < 0) printf("%.2lf\n", -area);
        else printf("%.2lf\n", area);
    }
    return 0;
}


//□□□□□□


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


#define Maxn 3005
const double eps = 1e-10;


typedef struct TPodouble
{
    double x;
    double y;
}TPoint;


typedef struct TPolygon
{
    TPoint p[Maxn];
    int n;
};


typedef struct TLine
{
    double a, b, c;
}TLine;


bool same(TPoint p1, TPoint p2)
{
    if(p1.x != p2.x) return false;
    if(p1.y != p2.y) return false;
    return true;
}

double multi(TPoint p1, TPoint p2, TPoint p0)
{
    //□□□-[p0, p1], [p0, p2]µ□□□
    //p0□□¥µ□
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
    //□□□□µ□□□0£¬□□□□□□□µ🄵□□
    //□□□□´□□□0£¬□□p0p2□□p0p1µ□□□□□🄵□□
    //□□□□□□□□0£¬□□p0p2□□p0p1µ□□□□🄵□□
}


TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //□□□□□□□□□, · µ»□□□³½□□□¸□□□
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}


TPoint LineInter(TLine l1, TLine l2)
{
    //□□}□ý»µ□□□□±□
    TPoint tmp;
    double a1 = l1.a;
    double b1 = l1.b;
    double c1 = l1.c;
    double a2 = l2.a;
    double b2 = l2.b;
    double c2 = l2.c;
    //□□□□□□□b1 = 0
    if(fabs(b1) < eps){
        tmp.x = -c1 / a1;
        tmp.y = (-c2 - a2 * tmp.x) / b2;
    }
    else{
        tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
        tmp.y = (-c1 - a1 * tmp.x) / b1;
    }
    return tmp;
}


TPolygon Cut_polygon(TPoint p1, TPoint p2, TPolygon polygon)
{
    TPolygon new_polygon;
    TPoint interp;
    TLine l1, l2;
    int i, j;
    double t1, t2;
    new_polygon.n = 0;
    for(i = 0;i <= polygon.n - 1;i++){
        t1 = multi(p2, polygon.p[i], p1);
        t2 = multi(p2, polygon.p[i + 1], p1);
        if(fabs(t1) < eps || fabs(t2) < eps){
            if(fabs(t1) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i];
            if(fabs(t2) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
        }
        else if(t1 < 0 && t2 < 0){
            new_polygon.p[new_polygon.n++] = polygon.p[i];
            new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
        }
        else if(t1 * t2  < 0){
            l1 = lineFromSegment(p1, p2);
            l2 = lineFromSegment(polygon.p[i], polygon.p[i + 1]);
            interp = LineInter(l1, l2);
            if(t1 < 0) {
                new_polygon.p[new_polygon.n++] = polygon.p[i];
                new_polygon.p[new_polygon.n++] = interp;
            }
            else {
                new_polygon.p[new_polygon.n++] = interp;
                new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
            }
        }
    }
    polygon.n = 0;
    if(new_polygon.n == 0) return polygon;
    polygon.p[polygon.n++] = new_polygon.p[0];
    for(i = 1;i < new_polygon.n;i++){
        if(!same(new_polygon.p[i], new_polygon.p[i - 1])){
            polygon.p[polygon.n++] = new_polygon.p[i];
        }
    }
    if(polygon.n != 1 && same(polygon.p[polygon.n - 1], polygon.p[0])) polygon.n--;
    polygon.p[polygon.n] = polygon.p[0];
    return polygon;
}


void ChangeClockwise(TPolygon &polygon)
{
    TPoint tmp;
    int i;
    for(i = 0;i <= (polygon.n - 1) / 2;i++){
        tmp = polygon.p[i];
        polygon.p[i] = polygon.p[polygon.n - 1 - i];
        polygon.p[polygon.n - 1 - i] = tmp;
    }
}


double polygonArea(TPolygon p)
{
    //□□¶□□□□□□□±□□□□□□□□
    double area;
    int i, n;
    n = p.n;
    area = 0;
    for(i = 1;i <= n;i++){
        area += (p.p[i - 1].x * p.p[i % n].y - p.p[i % n].x * p.p[i - 1].y);
    }
    return area / 2;
}


int main()
{
    int i, j;
    TPolygon polygon, new_polygon;
    while(scanf("%d", &polygon.n) && polygon.n){
        for(i = 0;i <= polygon.n - 1;i++){
            scanf("%lf%lf", &polygon.p[i].x, &polygon.p[i].y);
        }
        /*□□□□□□□□□□¯«□□□□□*/
        if(polygonArea(polygon) > 0) ChangeClockwise(polygon);
        polygon.p[polygon.n] = polygon.p[0];
        new_polygon = polygon;
        for(i = 0;i <= polygon.n - 1;i++){
            new_polygon = Cut_polygon(polygon.p[i], polygon.p[i + 1], new_polygon);
        }
        if(new_polygon.n > 0) printf("1\n");
        else printf("0\n");
    }
    return 0;
}
```

1.8 共线最多的点的个数

```
/*
2617120 chenhaifeng 1118 Accepted 512K 1890MS C++ 977B 2007-09-04 18:43:26
½□O(n^3)³¬□£¬□□□¸□□□□□□□🄵±□□i,j□□□□□□□□□²□□□□□£¬¿□□□□□□¹□
´□ţµ□□□□
□□□□□□□□ O(n3) µģ¬³¬□¡£□□□□□□□□ö¾□ÿ¸□🄵
□°□□□□□□□□µ□□□□l□□□□□□□¬□□□□□□□□□□□□¾□ö½¾¹□
¸õ□□□□□□□□□□□□¾¸¸□🄵□□□□□□□¾□□□□□□£¸´
□□□□□ O(n2logn) µġ£°□□□□□□□ hash£¬
¿□□□□Ż¯µ½ O(n2)¡£
2617134 chenhaifeng 1118 Accepted 276K 312MS G++ 1394B 2007-09-04 18:49:08
*/
```

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


bool f[705][705];
int a[705];


int main()
{
    int n, i, j, s, num, maxn;
    int x[705], y[705];
    int t, m;



    while(scanf("%d", &n) != EOF && n){
        for(i = 0;i <= n - 1;i++){
            scanf("%d%d", &x[i], &y[i]);
        }
        maxn = -1;
        for(i = 0;i <= n - 1;i++){
            for(j = i;j <= n - 1;j++){
                f[i][j] = false;
            }
        }
        for(i = 0;i <= n - 1;i++){
            for(j = i + 1;j <= n - 1;j++){
                if(f[i][j] == true) continue;
                if(n - j < maxn) break;
                num = 2;
                t = 2;
                a[0] = i;
                a[1] = j;
                f[i][j] = true;
                for(s = j + 1;s <= n - 1;s++){
                    if(f[i][s] == true || f[j][s] == true) continue;
                    if((y[i] - y[s]) * (x[j] - x[s]) == (x[i] - x[s]) * (y[j] - y[s])){
                         num++;
                         a[t] = s;
                         for(m = 0;m <= t - 1;m++){
                                f[m][s] = true;
                        }
                        t++;
                    }
                }
                if(num > maxn) maxn = num;
            }
        }
        printf("%d\n", maxn);
    }
    return 0;
}
```

1.9 线段围成的区域可储水量

```
/*
}□□□□□□¬
```

```
□□□□□□□Ŀ□»□□¬£□
½»µ□□□□□□□□□□□□□□□□±□□□□□□□
□□□□□□□□ · □□□
*/
#include <stdio.h>
#include <math.h>


#define eps 1e-8


struct TPoint
{
    double x, y;
};
struct TLine
{
    double a, b, c;
};


int same(TPoint p1, TPoint p2)
{
    if(fabs(p1.x - p2.x) > eps) return 0;
    if(fabs(p1.y - p2.y) > eps) return 0;
    return 1;
}


double min(double x, double y)
{
    if(x < y) return x;
    else return y;
}


double max(double x, double y)
{
    if(x > y) return x;
    else return y;
}


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


bool isIntersected(TPoint s1, TPoint e1, TPoint s2, TPoint e2)
{
    if(
    (max(s1.x, e1.x) >= min(s2.x, e2.x)) &&
    (max(s2.x, e2.x) >= min(s1.x, e1.x)) &&
    (max(s1.y, e1.y) >= min(s2.y, e2.y)) &&
    (max(s2.y, e2.y) >= min(s1.y, e1.y)) &&
    (multi(s2, e1, s1) * multi(e1, e2, s1) >= 0) &&
    (multi(s1, e2, s2) * multi(e2, e1, s2) >= 0)
    )  return true;

    return false;
}


TLine lineFromSegment(TPoint p1, TPoint p2)
{
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}


TPoint LineInter(TLine l1, TLine l2)
{
    TPoint tmp;
    double a1 = l1.a;
    double b1 = l1.b;
    double c1 = l1.c;
    double a2 = l2.a;
    double b2 = l2.b;
    double c2 = l2.c;
    if(fabs(b1) < eps){
        tmp.x = -c1 / a1;
        tmp.y = (-c2 - a2 * tmp.x) / b2;
    }
    else{
        tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
        tmp.y = (-c1 - a1 * tmp.x) / b1;
    }
    return tmp;
}


double triangleArea(TPoint p1, TPoint p2, TPoint p3)
{
    TPoint p4, p5;
    p4.x = p2.x - p1.x;
    p4.y = p2.y - p1.y;
    p5.x = p3.x - p1.x;
    p5.y = p3.y - p1.y;
    return fabs(p5.x * p4.y - p5.y * p4.x) / 2;
}


double find_x(double y, TLine line)
{
    return (-line.c - line.b * y) / line.a;
}


double find_y(double x, TLine line)
{
    if(fabs(line.b) < eps)
    {
        return -1e250;
    }
    else
    {
        return (-line.c - line.a  * x) / line.b;
    }
}


int main()
{
    //freopen("in.in", "r", stdin);
    //freopen("out.out", "w", stdout);
    int test;
    double miny, y;
    TLine l1, l2;
    TPoint p1, p2, p3, p4, inter;
    TPoint tp1, tp2;
    scanf("%d", &test);
    while(test--)
    {
        scanf("%lf%lf%lf%lf%lf%lf%lf%lf", &p1.x, &p1.y,
        &p2.x, &p2.y, &p3.x, &p3.y, &p4.x, &p4.y);
        if(same(p1, p2) || same(p3, p4)
           || !isIntersected(p1, p2, p3, p4)
           || fabs(p1.y - p2.y) < eps //□□□□□x□□
           || fabs(p3.y - p4.y) < eps
          )
        {
            printf("0.00\n");
            continue;
        }
        l1 = lineFromSegment(p1, p2);
        l2 = lineFromSegment(p3, p4);
        inter = LineInter(l1, l2);
        if(p1.y > p2.y) tp1 = p1;
        else tp1 = p2;
        if(p3.y > p4.y) tp2 = p3;
        else tp2 = p4;
        if(tp1.y < tp2.y)
        {
            if(tp1.x >= min(p4.x, p3.x) && tp1.x <= max(p4.x, p3.x))
            {
                y = find_y(tp1.x, l2);
                if(y >= tp1.y)
                {
                    printf("0.00\n");
                    continue;
                }
            }
            miny = tp1.y;
        }
        else
        {
            if(tp2.x >= min(p1.x, p2.x) && tp2.x <= max(p1.x, p2.x))
            {
                y = find_y(tp2.x, l1);
                if(y >= tp2.y)
                {
                    printf("0.00\n");
                    continue;
                }
            }
            miny = tp2.y;
        }
        if(fabs(miny - inter.y) < eps)
        {
            printf("0.00\n");
            continue;
        }
        tp1.x = find_x(miny, l1);
        tp2.x = find_x(miny, l2);
        tp1.y = tp2.y = miny;
        printf("%.2lf\n", triangleArea(tp1, tp2, inter));
    }
    return 0;
}
```

1.10 皮克公式

```
//A = b / 2 + i -1  □□□□ b □□ i  ·□e±□□□□□□□□°□□¿µ□□□□□□ç□
//http://www.hwp.idv.tw/bbs1/htm/%A6V%B6q%B7L%BFn%A4%C0/%A6V%B6q%B7L%BFn%A4%C0.htm
// http://acm.pku.edu.cn/JudgeOnline/problem?id=2954


#include <stdio.h>
#include <stdlib.h>


typedef struct TPoint
{
    int x;
    int y;
}TPoint;


typedef struct TLine
{
    int a, b, c;
}TLine;


int triangleArea(TPoint p1, TPoint p2, TPoint p3)
{
    //□□□□½□□□□□□¸□□□±□□□□□□½□□□□□□□
    int k = p1.x * p2.y + p2.x * p3.y + p3.x * p1.y
      - p2.x * p1.y - p3.x * p2.y - p1.x * p3.y;
    if(k < 0) return -k;
    else return k;
}


TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //□□□□□□□□□□, · µ»□□□□½□□□¸□□□
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}


void swap(int &a, int &b)
{
    int t;
    t = a;
    a = b;
    b = t;
}


int Count(TPoint p1, TPoint p2)
{
    int i, sum = 0, y;
    TLine l1 =  lineFromSegment(p1, p2);
    if(l1.b == 0) return abs(p2.y - p1.y) + 1;
    if(p1.x > p2.x) swap(p1.x, p2.x); //□□□□û□□»»»WA}´□
    for(i = p1.x;i <= p2.x;i++){
        y = -l1.c - l1.a * i;
        if(y % l1.b == 0) sum++;
    }
    return sum;
}


int main()
{
    //freopen("in.in", "r", stdin);
    //freopen("OUT.out", "w", stdout);
    TPoint p1, p2, p3;
    while(scanf("%d%d%d%d%d%d", &p1.x, &p1.y, &p2.x, &p2.y, &p3.x, &p3.y) != EOF){
        if(p1.x == 0 && p1.y == 0 && p2.x == 0 && p2.y == 0 && p3.x == 0 && p3.y == 0)
    break;
        int  A = triangleArea(p1, p2, p3);//A□□□□µ□}±¶
        int b = 0;
        int i;
        b = Count(p1, p2) + Count(p1, p3) + Count(p3, p2) - 3;//3¸□□□□□□□´□
        //i = A / 2- b / 2 + 1;
        i = (A - b) / 2 + 1;
        printf("%d\n", i);
    }
    return 0;
}
```

1.11 N 点中三个点组成三角形面积最大

```

//Rotating Calipers algorithm




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

```
#define MaxNode 50005


int stack[MaxNode];
int top;
double max;


typedef struct TPoint
{
    int x;
    int y;
}TPoint;
TPoint point[MaxNode];


void swap(TPoint point[], int i, int j)
{
    TPoint tmp;
    tmp = point[i];
    point[i] = point[j];
    point[j] = tmp;
}


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


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


int cmp(const void *a, const void *b)
{
    TPoint *c = (TPoint *)a;
    TPoint *d = (TPoint *)b;
    double k = multi(*c, *d, point[0]);
    if(k< 0) return 1;
    else if(k == 0 && distance(*c, point[0]) >= distance(*d, point[0]))
            return 1;
    else return -1;
}


void grahamScan(int n)
{
    //Graham□□□□□□°□
    int i, u;

    //½«□□□□□µ□□□□□µ½p[0]µ□□□□
    u = 0;
    for(i = 1;i <= n - 1;i++){
        if((point[i].y < point[u].y) ||
             (point[i].y == point[u].y && point[i].x  < point[u].x))
            u = i;
    }
    swap(point, 0, u);

    //½«□p[1]µ½p[n - 1]°´°´¾«½□□□□□□□□ÿ□□□□□□□
    qsort(point + 1, n - 1, sizeof(point[0]), cmp);

    for(i = 0;i <= 2;i++) stack[i] = i;
    top = 2;
    for(i = 3;i <= n - 1;i++){
        while(multi(point[i], point[stack[top]], point[stack[top - 1]]) >= 0){
            top--;
            if(top == 0) break;
        }
        top++;
        stack[top] = i;
    }
}


int main()
{
    double triangleArea(int i, int j, int k);
    void PloygonTriangle();
    int i, n;
    while(scanf("%d", &n) && n != -1){
        for(i = 0;i < n;i++)
            scanf("%d%d", &point[i].x, &point[i].y);
        if(n <= 2){
            printf("0.00\n");
            continue;
        }
        if(n == 3){
            printf("%.2lf\n", triangleArea(0, 1, 2));
            continue;
        }
        grahamScan(n);
        PloygonTriangle();
        printf("%.2lf\n", max);
    }
    return 0;
}


void PloygonTriangle()
{
    double triangleArea(int i, int j, int k);
    int i, j , k;
    double area, area1;
    max = -1;
    for(i = 0;i <= top - 2;i++){
        k = -1;
        for(j = i + 1; j <= top - 1;j++){
            if(k <= j) k= j + 1;
            area = triangleArea(stack[i], stack[j], stack[k]);
            if(area > max) max = area;
            while(k + 1 <= top){
                area1= triangleArea(stack[i], stack[j], stack[k + 1]);
                if(area1 < area) break;
                if(area1 > max) max = area1;
                area = area1;
                k++;
            }
        }
    }
}


double triangleArea(int i, int j, int k)
{
    //□□□□½□□□□□□¸□□□±□□□□□½□□□□□□□
    double l = fabs(point[i].x * point[j].y + point[j].x * point[k].y
        + point[k].x * point[i].y - point[j].x * point[i].y
        - point[k].x * point[j].y - point[i].x * point[k].y) / 2;
    return l;
}
```

1.12 直线关于圆的反射

```
/*
fzu_1035
1.□□□□µĽ»µ□
2.µ□□□□□□Ķ□□□□
3.µ🄵□ľ□□□□□
4.□□³½□
*/
#include <iostream>


#include <cmath>


using namespace std;


#define INF 999999999
const double eps = 1e-6;


int up;


typedef struct TPoint
{
    double x;
    double y;
}TPoint;


typedef struct TCircle
{
    TPoint center;
    double r;
}TCircle;

```

```

typedef struct TLine
{
    //□□□□□□□□□□
    double a, b, c;
}TLine;


void SloveLine(TLine &line, TPoint start, TPoint dir)
{
    //¸□¾□□□□□µ□□□½ķ□□□□□□ķ½³□□
    if(dir.x == 0){
        line.a = 1;
        line.b = 0;
        line.c = start.x;
    }
    else {
        double k = dir.y / dir.x;
        line.a = k;
        line.b = -1;
        line.c = start.y - k * start.x;
    }
}


TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //□□□□□□□□□, · µ»□□□□³½□□□¸□□□
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}


TPoint symmetricalPointofLine(TPoint p, TLine L)
{
    //pµ□□□□□□Lµķ□□□
    TPoint 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;
}


double distanc(TPoint p1, TPoint p2)
{
    //¼□□□□□□□□}¸□□¼□ľ□□□
   return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}


bool samedir(TPoint dir, TPoint start, TPoint point)
{
    //□□□½□□
    TPoint tmp;
    tmp.x = point.x - start.x;
    tmp.y = point.y - start.y;
    if(tmp.x != 0 && dir.x != 0){
        if(tmp.x / dir.x > 0) return true;
        else return false;
    }
    else if(tmp.y != 0 && dir.y != 0){
        if(tmp.y / dir.y > 0) return true;
        else return false;
    }
    return true;
}


bool Intersected(TPoint &point, TLine line, const TCircle circle[],
                 TPoint start, TPoint dir, int which)
{
    //□□□□□□□□)□□□□□½»µ□)½»µ□□□□□□-□point□□
    double a = line.a, b = line.b, c = line.c;
    double x0 = circle[which].center.x, y0 = circle[which].center.y;
    double r =  circle[which].r;
    //□□»µ🄵□□□□
    double x2front = b * b + a * a;
    double x1front = -2 * x0 * b * b + 2 * a * b * y0 + 2 * a * c;
    double front = x0 * x0 * b * b + y0 * y0 * b * b
        + c * c + 2 * c * y0 * b - b * b * r * r;
    double d = x1front * x1front - 4 * x2front * front;
    TPoint p1, p2;
    bool k1, k2;
    if(fabs(d) < eps){
        //x2front²»¿□□□□□□□
       point.x = -x1front / x2front / 2;
       point.y = (-c - a * point.x) / b;
       //□□□½□□
       if(samedir(dir, start, point)) return true;
       else return false;
    }
    else if(d < 0) return false;
    else {
        p1.x = (-x1front + sqrt(d)) / 2 / x2front;
        p1.y = (-c - a * p1.x) / b;
        p2.x = (-x1front - sqrt(d)) / 2 / x2front;
        p2.y = (-c - a * p2.x) / b;
        k1 = samedir(dir, start, p1);
        k2 = samedir(dir, start, p2);
        if(k1 == false && k2 == false) return false;
        if(k1 == true && k2 == true){
            double dis1 = distanc(p1, start);
            double dis2 = distanc(p2, start);
            if(dis1 < dis2) point = p1;
            else point = p2;
            return true;
        }
        else if(k1 == true) point = p1;
        else point = p2;
        return true;
    }
```

```
}


void Reflect(int &num, TCircle circle[], TPoint start, TPoint dir, int n)
{
    //·´¸´·´□□
    int i;
    TLine line;
    TPoint interpoint, newstart;
    int u;
    SloveLine(line, start, dir);
    int tag = 0;
    double mindis = INF;
    for(i = 1;i <= n;i++){
        if(i != up && Intersected(interpoint, line, circle, start, dir, i)){
            double dis = distanc(start, interpoint);
            if(dis < mindis){
                tag = 1;
                u = i;
                mindis = dis;
                newstart = interpoint;
            }
        }
    }
    if(tag == 0){
        cout << "inf" << endl;
        return ;
    }
    else {
        if(num == 10){
            cout << "..." << endl;
           return ;
        }
        cout << u << " ";
        num++;
        //□µķ½□□
        TLine line1;
        TPoint p;
        line1 = lineFromSegment(newstart, circle[u].center);
        if(fabs(line1.a * start.x + line1.b * start.y +line1.c) <= eps){
            dir.x = -dir.x;
            dir.y = -dir.y;
        }
        else {
            p = symmetricalPointofLine(start, line1);//startµĶ□□□
            dir.x = p.x - newstart.x;
            dir.y = p.y - newstart.y;
        }

        start = newstart;
        up = u;
        Reflect(num, circle, start, dir, n);
    }
}


int main()
{
    //freopen("fzu_1035.in", "r", stdin);
    //freopen("fzu_1035.out", "w", stdout);
    int n, i, j, num, test = 1;
    TCircle circle[30];
    TPoint start, dir;
    while(cin >> n && n){
        for(i = 1;i <= n;i++){
            cin >> circle[i].center.x >> circle[i].center.y >> circle[i].r;
        }
        cin >> start.x >> start.y >> dir.x >> dir.y;

        cout << "Scene " << test++ << endl;


        num = 0;
        up = -1;
        Reflect(num, circle, start, dir, n);
        cout << endl;
    }
    return 0;
}
```

1.13 N 个点最多组成多少个正方形

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define eps 1e-6
#define pi acos(-1.0)


#define PRIME 9991


struct point
{
    int x, y;
}p[2201];
int n;


struct HASH
{
    int cnt;
    int next;
}hash[50000];
int hashl;


int Hash(int n)
{
    int i = n % PRIME;
    while(hash[i].next != -1){
        if(hash[hash[i].next].cnt == n) return 1;
        else if(hash[hash[i].next].cnt > n) break;
        i = hash[i].next;
    }
    hash[hashl].cnt = n;
    hash[hashl].next = hash[i].next;
    hash[i].next = hashl;
    hashl++;
    return 0;
}


int Hash2(int n)
{
    int i = n % PRIME;
    while(hash[i].next != -1){
        if(hash[hash[i].next].cnt == n) return 1;
        else if(hash[hash[i].next].cnt > n) return 0;
        i = hash[i].next;
    }
    return 0;
}


int check(double ax, double ay, int &x, int &y)
{
    int a0 = (int)ax;
    int b0 = (int)ay;
    int tag1 = 0, tag2 = 0;
    if(fabs(a0 - ax) < eps){
        tag1 = 1;
        x = a0;
    }
    else if(fabs(a0 + 1 - ax) < eps){
        tag1 = 1;
        x = a0 + 1;
    }
    if(fabs(b0 - ay) < eps){
        tag2 = 1;
        y = b0;
    }
    else if(fabs(b0 + 1 - ay) < eps){
        y = b0 + 1;
        tag2 = 1;
    }
    if(tag1 == 1 && tag2 == 1) return 1;
    else return 0;
}


int squares(point p1, point p2, point &p3, point &p4)
{
    double a = (double)p2.x - p1.x;
    double b = (double)p2.y - p1.y;
    double midx = ((double)p1.x + p2.x) / 2;
    double midy = ((double)p1.y + p2.y) / 2;
    double tmp = a * a + b * b;
    double x1 = sqrt(b * b) / 2;
    double y1;
    if(fabs(b) < eps) y1 = sqrt(a * a + b * b) / 2;
    else y1 = -a * x1 / b;
    x1 += midx;
    y1 += midy;
    if(check(x1, y1, p3.x, p3.y) == 0) return 0;
    x1 = 2 * midx - x1;
    y1 = 2 * midy - y1;
    if(check(x1, y1, p4.x, p4.y) == 0) return 0;
    return 1;
}


int main()
{
    int i, j, cnt;
    while(scanf("%d", &n) != EOF && n)
    {
        for(i = 0;i < PRIME;i++) hash[i].next = -1;
        hashl = PRIME;
        int x1, y1, x2, y2;
        for (i = 0; i < n; i++){
            scanf("%d%d", &p[i].x, &p[i].y);
            Hash((p[i].x + 100000) * 100000 + p[i].y + 100000);
        }
        cnt = 0;
        for (i = 0; i < n; i++){
            for (j = i + 1; j < n; j++)
            {
                point a, b;
                if(squares(p[i], p[j], a, b) == 0) continue;
                if(Hash2((a.x + 100000) * 100000 + a.y + 100000) == 0) continue;
                if(Hash2((b.x + 100000) * 100000 + b.y + 100000) == 0) continue;
                cnt++;
            }
        }
        printf("%d\n", cnt / 2);
    }
    return 0;
}
```

1.14 单位圆覆盖最多点

```
/*
□□□□□N¸□🄵□□□¸□🄵Rµ□□□¸²¸□¬□□□□□¸□□□□□🄵
±□□µ□□□□Ŀ¡£
¶□ÿ¸□□□□R□°🄵»□£¬¶□N¸□□}}□□£□□□²½O(N^2)¡£□□□□□¯«□□□²¸□□□□□□□Ļ¡¡£
¶□ÿ□¸□□£¬□□□□□□□□ÿ¶□¡□□□´□□□¡£¼□□□A□□□B□□¡£A□□[PI/3, PI/2]µ□□□¼🄵B¸²¸□(PI□□□□□□□)¡£□□ô¶□□□A□£¬□□□□□□□PI/3´¦□□□¸□+1±□□¬□□PI/2´¦□□□¸□-1±□□£
¶□□□[PI*5/3, PI*7/3]□□□□°□□0µ□□□□¼□□0□□µ🄵²□□}¶□´¿□£
½«□¸□□□□□□□□□□□□□□□□□□□¿ª□□□衣³□□ans = 0£¬□□+1±□□□ans++£¬□□-1±□□ans--¡£□□□□³□□□ansµ□□□□□□□□□□□□»¸²¸□□□□Ļ¡¡£□□□□□□□□µ□ansµ□□□□□□□□□□£
□□□□□O(N^2 * logN)
*/#include <stdio.h>
#include <math.h>


#define eps 1e-6


struct point
{
    double x, y;
};
```

```
double dis(point p1, point p2)
{
    point p3;
    p3.x = p2.x - p1.x;
    p3.y = p2.y - p1.y;
    return p3.x * p3.x + p3.y * p3.y;
}


point find_centre(point p1, point p2)
{
    point p3, mid, centre;
    double b, c, ang;
    p3.x = p2.x - p1.x;
    p3.y = p2.y - p1.y;
    mid.x = (p1.x + p2.x) / 2;
    mid.y = (p1.y + p2.y) / 2;
    b = dis(p1, mid);
    c = sqrt(1 - b);
    if(fabs(p3.y) < eps)//´¹□□□□½□90¶□
    {
        centre.x = mid.x;
        centre.y = mid.y + c;
    }
    else
    {
        ang = atan(-p3.x / p3.y);
        centre.x = mid.x + c * cos(ang);
        centre.y = mid.y + c * sin(ang);
    }
    return centre;
}


int main()
{
    int n, ans, tmpans, i, j, k;
    point p[305], centre;
    double tmp;
    while(scanf("%d", &n) && n)
    {
        for(i = 0;i < n;i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        ans = 1;
        for(i = 0;i < n;i++)
            for(j = i + 1;j < n;j++)
            {
                if(dis(p[i], p[j]) > 4) continue;
                tmpans = 0;
                centre = find_centre(p[i], p[j]);
                for(k = 0;k < n;k++)
                {
                    //if(tmpans + n - k <= ans) break;
                    tmp = dis(centre, p[k]);
                    //if(tmp < 1.0 || fabs(tmp - 1.0) < eps) tmpans++;
                    if(tmp <= 1.000001) tmpans++;
                }
                if(ans < tmpans) ans = tmpans;
            }
        printf("%d\n", ans);
    }
    return 0;
}
```

1.15 N 个点最多确定多少互不平行的直线

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


#define eps 1e-6
#define pi acos(-1)


struct point
{
    double x, y;
};


double FindSlewRate(point p1, point p2)
{
    point p;
    p.x = p2.x - p1.x;
    p.y = p2.y - p1.y;
    if(fabs(p.x) < eps) return pi / 2;
    double tmp = atan(p.y / p.x);
    if(tmp < 0) return pi + tmp;
    return tmp;
}


int cmp(const void *a, const void *b)
{
    double *c = (double *)a;
    double *d = (double *)b;
    if(*c < *d) return -1;
    return 1;
}


int main()
{
    int n, rt;
    point p[205];
    double rate[40005];
    while(scanf("%d", &n) != EOF)
    {
        for(int i = 0;i < n;i++)
            scanf("%lf%lf", &p[i].x ,&p[i].y);
        rt = 0;
        for(int i = 0;i < n;i++)
            for(int j = i + 1;j < n;j++)
                rate[rt++] = FindSlewRate(p[i], p[j]);
        qsort(rate, rt, sizeof(rate[0]), cmp);
        int ans = 1;
        for(int i = 1;i < rt;i++)
            if(rate[i] > rate[i - 1]) ans++;
        //□□□□□□□□□fabs(rate[i] - rate[i - 1]) > eps Wrong Answer
        printf("%d\n", ans);
    }
    return 0;
}
```

1.16 求凸多边形直径

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


#define eps 1e-6
#define MaX 6000


/*-----------¶□□□□□------------*/
struct POLYGON{
   int n;                                   //¶□□□□¥µ□□□
   double x[MaX],y[MaX];          //¶¥µ□□□±□
}poly;


int zd[100000][2],znum;      //□Ŷ□ļ¯°□□□Ŷ□□□□-




/*------------¸¨□□°¯□□-----------*/
double dist(int a,int b,int c)
{
   double vx1,vx2,vy1,vy2;
   vx1=poly.x[b]-poly.x[a]; vy1=poly.y[b]-poly.y[a];
   vx2=poly.x[c]-poly.x[a]; vy2=poly.y[c]-poly.y[a];
   return fabs(vx1*vy2 - vy1*vx2);
}




/*-------------□□□¶□□□□¾¶µÍ¯□□-------------*/
double DIAMETER()
{
   znum=0;
   int i,j,k=1;
   double m,tmp;
   while(dist(poly.n-1,0,k+1) > dist(poly.n-1,0,k)+eps)
      k++;
   i=0; j=k;
   while(i<=k && j<poly.n)
   {
      zd[znum][0]=i; zd[znum++][1]=j;
      while(dist(i,i+1,j+1)>dist(i,i+1,j)-eps && j<poly.n-1)
      {
         zd[znum][0]=i; zd[znum++][1]=j;
         j++;
      }
      i++;
   }
   m=-1;
   for(i=0;i<znum;i++)
   {
      tmp =(poly.x[zd[i][0]]-poly.x[zd[i][1]]) * (poly.x[zd[i][0]]-poly.x[zd[i][1]]);
      tmp+=(poly.y[zd[i][0]]-poly.y[zd[i][1]]) * (poly.y[zd[i][0]]-poly.y[zd[i][1]]);
      if(m<tmp) m=tmp;
   }
   return sqrt(m);
}


/*----------□□□□----------*/
int main()
{
   int i;
   while(scanf("%d",&poly.n)==1)
   {
      for(i=0;i<poly.n;i++)
         scanf("%lf %lf",&poly.x[i],&poly.y[i]);
      printf("%.3lf\n",DIAMETER());
   }
   return 0;
}
1.16.19 ¾□□□□□□²¢£¬□¤²¢
¼□¸½¼□5.11,5.12
1.16.20 pku2069 □□□□□□
¼□¸½¼□5.13,5.14
//□□□±□□□□
#include<stdio.h>
#include<math.h>
#include<memory>
#include<stdlib.h>
using namespace std;
const double eps = 1e-10;
struct point_type { double x, y, z; };
int npoint, nouter ;
point_type point [1000], outer[4], res;
double radius, tmp ;


inline double dist(point_type p1 , point_type p2)
{
double dx=p1.x-p2.x, dy=p1.y-p2.y,dz=p1.z-p2.z ;
return ( dx*dx + dy*dy + dz*dz ) ;
}


inline double dot( point_type p1 , point_type p2 )
{
return p1.x*p2.x + p1.y*p2.y + p1.z*p2.z;
}


void ball()
```

```
{
    point_type q[3];
    double m[3][3],sol[3],L[3],det; int i,j;
    res.x=res.y=res.z=-1000;
    radius=0;
    switch ( nouter )
        {
        case 1 : res=outer[0]; break;
        case 2 :
                 res.x=(outer[0].x+outer[1].x)/2;
                 res.y=(outer[0].y+outer[1].y)/2;
                 res.z=(outer[0].z+outer[1].z)/2;
                 radius=dist(res,outer[0]);
                 break;
        case 3 :
                 for ( i=0; i<2; ++i ) {
                    q[i].x=outer[i+1].x-outer[0].x;
                    q[i].y=outer[i+1].y-outer[0].y;
                    q[i].z=outer[i+1].z-outer[0].z;
                    }
                 for ( i=0; i<2; ++i )
                     for ( j=0; j<2; ++j )
                        m[i][j]=dot(q[i],q[j])*2 ;
                 for ( i=0; i<2; ++i ) sol[i]=dot(q[i],q[i]);
                 if (fabs(det=m[0][0]*m[1][1]-m[0][1]*m[1][0]) < eps ) return ;

                 L[0]=(sol[0]*m[1][1]-sol[1]*m[0][1])/det;
                 L[1]=(sol[1]*m[0][0]-sol[0]*m[1][0])/det;
                 res.x=outer[0].x+q[0].x*L[0]+q[1].x*L[1];
                 res.y=outer[0].y+q[0].y*L[0]+q[1].y*L[1];
                 res.z=outer[0].z+q[0].z*L[0]+q[1].z*L[1];
                 radius=dist(res,outer[0]);
                 break;
        case 4 :
                 for ( i=0; i<3; ++i ){
                    q[i].x=outer[i+1].x-outer[0].x;
                    q[i].y=outer[i+1].y-outer[0].y;
                    q[i].z=outer[i+1].z-outer[0].z;
                    sol[i]=dot(q[i],q[i]);
                    }
                 for ( i=0; i<3; ++i)
                     for ( j=0; j<3; ++j) m[i][j]=dot(q[i],q[j])*2;
                 det= m[0][0]*m[1][1]*m[2][2] + m[0][1]*m[1][2]*m[2][0]
                     +m[0][2]*m[2][1]*m[1][0] - m[0][2]*m[1][1]*m[2][0]
                     -m[0][1]*m[1][0]*m[2][2] - m[0][0]*m[1][2]*m[2][1];


                 if ( fabs( det )<eps ) return;


                 for ( j=0; j<3; ++j ){
                    for ( i=0; i<3; ++i ) m[i][j]=sol[i];
                     L[j]=( m[0][0]*m[1][1]*m[2][2] + m[0][1]*m[1][2]*m[2][0]
                          + m[0][2]*m[2][1]*m[1][0] - m[0][2]*m[1][1]*m[2][0]
                          - m[0][1]*m[1][0]*m[2][2] - m[0][0]*m[1][2]*m[2][1]
                          ) / det;
                    for( i=0; i<3; ++i ) m[i][j]=dot(q[i],q[j])*2;
                    }
                 res=outer[0];
                 for ( i=0; i<3; ++i ) {
                    res.x+=q[i].x*L[i];
                    res.y+=q[i].y*L[i];
                    res.z+=q[i].z*L[i];
                    }
                 radius=dist(res,outer[0]);
        }
}


void minball(int n)
{
  ball();
  if ( nouter <4 )
       for ( int i=0; i<n; ++i )
           if( dist(res,point[i])-radius>eps)
            {
                outer[nouter]=point[i];
                ++nouter;
                minball(i);
                --nouter;
                if(i>0)
                {
                  point_type Tt = point[i] ;
                  memmove(&point[1], &point[0] , sizeof ( point_type )*i );
                  point[0]=Tt;
                }
            }
}






int main()
{
 int i;
 while(scanf("%d",&npoint)!=EOF,npoint)
    {
     for(i=0;i<npoint;i++)
       scanf("%lf%lf%lf",&point[i].x,&point[i].y,&point[i].z);
     nouter=0;
     minball(npoint);
     printf("%.8lf\n",sqrt(radius)+eps);
    }
 return 0;
}
```

1.17 圆和多边形的交

```
/*□°□□□□□□*/
#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;
}
```

1.18 半平面交

```
//Nlgn
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <algorithm>
using namespace std;
```

```
#define maxn 20005
#define eps 1e-10


struct point
{double x,y;};
struct line
{point s,e;double k;};


line L[maxn];
point S[maxn];


int N,Q[maxn];


double cross(point a,point b,point c) // c□□□□ab□□□μ»□<0
{return (c.y-a.y)*(b.x-a.x)-(b.y-a.y)*(c.x-a.x);}


bool operator < (line a,line b) // □□se□□□□¿□□□□□
{
    if( fabs(a.k-b.k)<eps )
        return cross(b.s,b.e,a.s)<0;
    return a.k<b.k;
}


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;
}


double HalfInSec()
{
    int i,j,k,l;
    sort(L,L+N);  // ¼«½□[-pi,pi]□□□□

    for(i=1,j=0;i<N;i++)  // □μ□□□¼«½□□□
        if( fabs(L[i].k-L[j].k)>eps )
        L[++j] = L[i];
    N = j+1;


    k = 0,l = 1;
    Q[0] = 0,Q[1] = 1;
    S[1] = intersection(L[0].s,L[0].e,L[1].s,L[1].e);
    for(i=2;i<N;i++)
    {
        while( k<l && cross(L[i].s,L[i].e,S[l])>eps )
            l--;
        while( k<l && cross(L[i].s,L[i].e,S[k+1])>eps )
            k++;
        Q[++l] = i;
        S[l] = intersection(L[Q[l-1]].s,L[Q[l-1]].e,L[i].s,L[i].e);
    }


    while( k<l && cross(L[Q[k]].s,L[Q[k]].e,S[l])>eps )
        l--;
    while( k<l && cross(L[Q[l]].s,L[Q[l]].e,S[k+1])>eps )
        k++;
    S[k] = intersection(L[Q[l]].s,L[Q[l]].e,L[Q[k]].s,L[Q[k]].e);
    S[++l] = S[k];


    double s = 0;
    for(i=k;i<l;i++)
        s += S[i].y*S[i+1].x-S[i+1].y*S[i].x;
    return fabs(s/2);
}
int main()
{
    int i,j,k,l;
    scanf("%d",&N);
    for(i=0;i<N;i++)
        scanf("%lf%lf%lf%lf",&L[i].e.x,&L[i].e.y,&L[i].s.x,&L[i].s.y);
    L[N].s.x = 0,L[N].s.y = 0;
    L[N+1].s.x = 10000,L[N+1].s.y = 0;
    L[N+2].s.x = 10000,L[N+2].s.y = 10000;
    L[N+3].s.x = 0,L[N+3].s.y = 10000;
    L[N].e = L[N+3].s;
    L[N+1].e = L[N].s;
    L[N+2].e = L[N+1].s;
    L[N+3].e = L[N+2].s;
    N += 4;
    for(i=0;i<N;i++)
        L[i].k = atan2(L[i].s.y-L[i].e.y,L[i].s.x-L[i].e.x);
    printf("%.1lf\n",HalfInSec());
}
```