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□□□□□□□□□□□□□μķμ»□□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

```
//¶□□□□□□□□
//¿□□□□□□□□□F
#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

```
//¸¡µÃ□°□¯□□¿□
#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){
```

```
32      return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
33  }
34
35
36  //□□□□μF□□
37  int dots_inline(point p1,point p2,point p3){
38      return zero(xmult(p1,p2,p3));
39  }
40  int dots_inline(double x1,double y1,double x2,double y2,double x3,double y3){
41      return zero(xmult(x1,y1,x2,y2,x3,y3));
42  }
43
44
45  //□□□□□□□□□□□□□□□,°□(¶□□
46  int dot_online_in(point p,line l){
47      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;
48  }
49  int dot_online_in(point p,point l1,point l2){
50      return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
51  }
52  int dot_online_in(double x,double y,double x1,double y1,double x2,double y2){
53      return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)*(x2-x)<eps&&(y1-y)*(y2-y)<eps;
54  }
55
56
57  //□□□□□□□□□□□□□□□,²»°□(¶□□
58  int dot_online_ex(point p,line l){
59      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));
60  }
61  int dot_online_ex(point p,point l1,point l2){
62      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));
63  }
64  int dot_online_ex(double x,double y,double x1,double y1,double x2,double y2){
65      return dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!
        zero(y-y2));
66  }
67
68
69  //□□}μ□□□□□□□²□,μ□□□□□□□□□μ»□0
70  int same_side(point p1,point p2,line l){
71      return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
72  }
73  int same_side(point p1,point p2,point l1,point l2){
74      return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
75  }
76
77
78  //□□}μ□□□□□□□□□□,μ□□□□□□□□□μ»□0
79  int opposite_side(point p1,point p2,line l){
80      return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
81  }
82  int opposite_side(point p1,point p2,point l1,point l2){
83      return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
84  }
85
86
```

```
87  //□□}□□□□□□
88  int parallel(line u,line v){
89      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));
90  }
91  int parallel(point u1,point u2,point v1,point v2){
92      return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
93  }
94
95
96  //□□}□¼□
97  int perpendicular(line u,line v){
98      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));
99  }
100 int perpendicular(point u1,point u2,point v1,point v2){
101     return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
102 }
103
104
105 //□□}□□□□□¸°□(¶□□□¿·□□□□
106 int intersect_in(line u,line v){
107     if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
108         return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
109     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);
110 }
111 int intersect_in(point u1,point u2,point v1,point v2){
112     if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
113         return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
114     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);
115 }
116
117
118 //□□}□□□□□¸²»°□(¶□□□¿·□□□□
119 int intersect_ex(line u,line v){
120     return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
121 }
122 int intersect_ex(point u1,point u2,point v1,point v2){
123     return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
124 }
125
126
127 //¼□□□}□□»µ□,□□□□□□□□□□□□□□□□□□□□□□□□□□□□!
128 //□□□»µ□□□□□□□□□□□□□□□□□□(□□»¹□□□□□□□□□□□□□□□!)
129 point intersection(line u,line v){
130     point ret=u.a;
131     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))
132             /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
133     ret.x+=(u.b.x-u.a.x)*t;
134     ret.y+=(u.b.y-u.a.y)*t;
135     return ret;
136 }
137 point intersection(point u1,point u2,point v1,point v2){
138     point ret=u1;
139     double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
140             /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
141     ret.x+=(u2.x-u1.x)*t;
142     ret.y+=(u2.y-u1.y)*t;
143     return ret;
```

```
144 }
145
146
147 //μF□□□□□□□□□μ□
148 point ptoline(point p,line l){
149     point t=p;
150     t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
151     return intersection(p,t,l.a,l.b);
152 }
153 point ptoline(point p,point l1,point l2){
154     point t=p;
155     t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
156     return intersection(p,t,l1,l2);
157 }
158
159
160 //μF□□□□□□
161 double disptoline(point p,line l){
162     return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
163 }
164 double disptoline(point p,point l1,point l2){
165     return fabs(xmult(p,l1,l2))/distance(l1,l2);
166 }
167 double disptoline(double x,double y,double x1,double y1,double x2,double y2){
168     return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);
169 }
170
171
172 //μF□□□□□□□□□□μ□
173 point ptoseg(point p,line l){
174     point t=p;
175     t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
176     if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
177         return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
178     return intersection(p,t,l.a,l.b);
179 }
180 point ptoseg(point p,point l1,point l2){
181     point t=p;
182     t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
183     if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
184         return distance(p,l1)<distance(p,l2)?l1:l2;
185     return intersection(p,t,l1,l2);
186 }
187
188
189 //μF□□□□□□□
190 double disptoseg(point p,line l){
191     point t=p;
192     t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
193     if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
194         return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
195     return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
196 }
197 double disptoseg(point p,point l1,point l2){
198     point t=p;
199     t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
200     if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
201         return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);
202     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¢□□□F□□□□□□£□
// 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;
```

```
262         return;
263     }
264     double xx,yy; //(xx,yy)□□□□□□□□□□□Ⓕ□□□ﾚ»µⒻ
265     xx=(b1*c2-b2*c1)/(a1*b2-a2*b1);
266     yy=(a2*c1-a1*c2)/(a1*b2-a2*b1);
267     a=n;
268     b=-m;
269     c=m*yy-xx*n;
270 }
```

0.4 面积

```
1  #include <math.h>
2  struct point{double x,y;};
3
4
5  //¼□□□cross product (P1-P0)x(P2-P0)
6  double xmult(point p1,point p2,point p0){
7      return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
8  }
9  double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
10     return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
11 }
12
13
14 //¼□□□□□½□□□□□□,□□□□□□¶¥µ□
15 double area_triangle(point p1,point p2,point p3){
16     return fabs(xmult(p1,p2,p3))/2;
17 }
18 double area_triangle(double x1,double y1,double x2,double y2,double x3,double y3){
19     return fabs(xmult(x1,y1,x2,y2,x3,y3))/2;
20 }
21
22
23 //¼□□□□□½□□□□□□,□□□□□□□±¤
24 double area_triangle(double a,double b,double c){
25     double s=(a+b+c)/2;
26     return sqrt(s*(s-a)*(s-b)*(s-c));
27 }
28
29
30 //¼□□□□□□□□□□□,¶¥µⒻ□□□□□□□□□□□□³□
31 double area_polygon(int n,point* p){
32     double s1=0,s2=0;
33     int i;
34     for (i=0;i<n;i++)
35         s1+=p[(i+1)%n].y*p[i].x,s2+=p[(i+1)%n].y*p[(i+2)%n].x;
36     return fabs(s1-s2)/2;
37 }
```

0.5 球

```
1  #include <math.h>
2  const double pi=acos(-1);
3
4
5  //¼□□□□□ﾚ□lat±□□□¶□,-90<=w<=90,lng±□□¾¶□
```

```
// · µ»□}µ□□□□□□□□□¡¶□□□□□□,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));
}


//¾□□□□□□□,r□□□Fl
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)));
}


//¾□□□□□□□□□□□,r□□□Fl
inline double sphere_dist(double r,double lng1,double lat1,double lng2,double lat2){
    return r*angle(lng1,lat1,lng2,lat2);
}


//□□□Fl□□
//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){   /**** µFl□□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□□□ι□□,¾□²»□□,·µ»□-1 */
int update ()
{
    int sign = -1, i;
    double k = 1e100, tmp;
```

```
123     point inter, t;
124     for (i = 1; i <= n; i++){ //□□%□□□□□□p□□□□
125         if (!find (cir[i], l, tmp, t)) continue;
126         if (dcmp (tmp - k) < 0) k = tmp, inter = t, sign = i;
127     }
128     //ray ±□□□
129     if (sign == -1) return sign;
130     ray l1;
131     l1.s = cir[sign].o;
132     l1.dir.x = (inter.x - l1.s.x) / cir[sign].r;
133     l1.dir.y = (inter.y - l1.s.y) / cir[sign].r;
134     l1.dir.z = (inter.z - l1.s.z) / cir[sign].r;
135     newRay (l, l1, inter);
136     return sign;
137 }
138 int main ()
139 {
140 //  freopen ("in", "r", stdin);
141     int i;
142     scanf ("%d", &n);
143     for (i = 1; i <= n; i++) //□□□□□□□□□□□□□□
144         scanf ("%lf%lf%lf%lf", &cir[i].o.x, &cir[i].o.y, &cir[i].o.z, &cir[i].r);
145     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);
146     for (i = 0; i <= 10; i++){ //□□□□□□□□´□□p□□□□i□□
147         int sign = update ();
148         if (sign == -1) break;
149         if (i == 0) printf ("%d", sign);
150         else if (i < 10) printf (" %d", sign);
151         else printf (" etc.");
152     }
153     puts ("");
154 }
```

0.6 三角形

```
1  #include <math.h>
2  struct point{double x,y;};
3  struct line{point a,b;};
4
5
6  double distance(point p1,point p2){
7      return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
8  }
9
10
11 point intersection(line u,line v){
12     point ret=u.a;
13     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))
14             /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
15     ret.x+=(u.b.x-u.a.x)*t;
16     ret.y+=(u.b.y-u.a.y)*t;
17     return ret;
18 }
19
20
21 //□□□□
22 point circumcenter(point a,point b,point c){
23     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□□□□§□□□□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;};
```

```
8
9
10  //¼□□□cross product U x V
11  point3 xmult(point3 u,point3 v){
12      point3 ret;
13      ret.x=u.y*v.z-v.y*u.z;
14      ret.y=u.z*v.x-u.x*v.z;
15      ret.z=u.x*v.y-u.y*v.x;
16      return ret;
17  }
18
19
20  //¼□□□dot product U . V
21  double dmult(point3 u,point3 v){
22      return u.x*v.x+u.y*v.y+u.z*v.z;
23  }
24
25
26  //□-²□ U - V
27  point3 subt(point3 u,point3 v){
28      point3 ret;
29      ret.x=u.x-v.x;
30      ret.y=u.y-v.y;
31      ret.z=u.z-v.z;
32      return ret;
33  }
34
35
36  //□□□Ⓕ□□-
37  point3 pvec(plane3 s){
38      return xmult(subt(s.a,s.b),subt(s.b,s.c));
39  }
40  point3 pvec(point3 s1,point3 s2,point3 s3){
41      return xmult(subt(s1,s2),subt(s2,s3));
42  }
43
44
45  //}µ□□□□,µ¥²□□□□□□-´□□
46  double distance(point3 p1,point3 p2){
47      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)
        );
48  }
49
50
51  //□□-´□□
52  double vlen(point3 p){
53      return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
54  }
55
56
57  //□□□□µⒻ□□
58  int dots_inline(point3 p1,point3 p2,point3 p3){
59      return vlen(xmult(subt(p1,p2),subt(p2,p3)))<eps;
60  }
61
62
63  //□□□□Ⓕ□□
64  int dots_onplane(point3 a,point3 b,point3 c,point3 d){
65      return zero(dmult(pvec(a,b,c),subt(d,a)));
```

```
66 }
67
68
69 //□□□□□□□□□□□□□□□,°□(¶□□□²□□
70 int dot_online_in(point3 p,line3 l){
71     return zero(vlen(xmult(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&
72         (l.a.y-p.y)*(l.b.y-p.y)<eps&&(l.a.z-p.z)*(l.b.z-p.z)<eps;
73 }
74 int dot_online_in(point3 p,point3 l1,point3 l2){
75     return zero(vlen(xmult(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-p.x)<eps&&
76         (l1.y-p.y)*(l2.y-p.y)<eps&&(l1.z-p.z)*(l2.z-p.z)<eps;
77 }
78
79
80 //□□□□□□□□□□□□□□□,²»°□(¶□□
81 int dot_online_ex(point3 p,line3 l){
82     return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y)||!zero(p.z-l.a.z))&&
83         (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
84 }
85 int dot_online_ex(point3 p,point3 l1,point3 l2){
86     return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y)||!zero(p.z-l1.z))
       &&
87         (!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
88 }
89
90
91 //□□□□□□□□□□□□□½□□□□□,°□(±□□,□□μⒻ□□□□□□□□
92 int dot_inplane_in(point3 p,plane3 s){
93     return zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-vlen(xmult(subt(p,s.a),subt(p,
       s.b)))-
94         vlen(xmult(subt(p,s.b),subt(p,s.c)))-vlen(xmult(subt(p,s.c),subt(p,s.a))));
95 }
96 int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
97     return zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-vlen(xmult(subt(p,s1),subt(p,s2)))
       -
98         vlen(xmult(subt(p,s2),subt(p,s3)))-vlen(xmult(subt(p,s3),subt(p,s1))));
99 }
100
101
102 //□□□□□□□□□□□□□□□½□□□□□,²»°□(±□□,□□μⒻ□□□□□□□□
103 int dot_inplane_ex(point3 p,plane3 s){
104     return dot_inplane_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>eps&&
105         vlen(xmult(subt(p,s.b),subt(p,s.c)))>eps&&vlen(xmult(subt(p,s.c),subt(p,s.a)))>
       eps;
106 }
107 int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
108     return dot_inplane_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>eps&&
109         vlen(xmult(subt(p,s2),subt(p,s3)))>eps&&vlen(xmult(subt(p,s3),subt(p,s1)))>eps;
110 }
111
112
113 //□□}μ□□□□□□□²□,μ□□□□□□□□□μ»□0,²»¹²□□□□□□□□
114 int same_side(point3 p1,point3 p2,line3 l){
115     return dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))>
       eps;
116 }
117 int same_side(point3 p1,point3 p2,point3 l1,point3 l2){
118     return dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))>eps;
119 }
```

```
120
121
122 //□□}μ□□□□□□□□□□,μ□□□□□□□□□μ»□0,²»¹²□□□□□□□□
123 int opposite_side(point3 p1,point3 p2,line3 l){
124     return dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))<-
    eps;
125 }
126 int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2){
127     return dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))<-eps;
128 }
129
130
131 //□□}μ□□□□□□□□,μ□□□□□□□□□μ»□0
132 int same_side(point3 p1,point3 p2,plane3 s){
133     return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))>eps;
134 }
135 int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
136     return dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))>eps;
137 }
138
139
140 //□□}μ□□□□□□□□□,μ□□□□□□□□□μ»□0
141 int opposite_side(point3 p1,point3 p2,plane3 s){
142     return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))<-eps;
143 }
144 int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
145     return dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))<-eps;
146 }
147
148
149 //□□}□□□□□□
150 int parallel(line3 u,line3 v){
151     return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<eps;
152 }
153 int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
154     return vlen(xmult(subt(u1,u2),subt(v1,v2)))<eps;
155 }
156
157
158 //□□}□□□□□□□
159 int parallel(plane3 u,plane3 v){
160     return vlen(xmult(pvec(u),pvec(v)))<eps;
161 }
162 int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
163     return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;
164 }
165
166
167 //□□□□□□□□□□□□□□
168 int parallel(line3 l,plane3 s){
169     return zero(dmult(subt(l.a,l.b),pvec(s)));
170 }
171 int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
172     return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));
173 }
174
175
176 //□□}□┴□
177 int perpendicular(line3 u,line3 v){
```

```
178     return zero(dmult(subt(u.a,u.b),subt(v.a,v.b)));
179 }
180 int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){
181     return zero(dmult(subt(u1,u2),subt(v1,v2)));
182 }
183
184
185 //□□}□□逗□
186 int perpendicular(plane3 u,plane3 v){
187     return zero(dmult(pvec(u),pvec(v)));
188 }
189 int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
190     return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));
191 }
192
193
194 //□□□□□□□□□□□□□□□
195 int perpendicular(line3 l,plane3 s){
196     return vlen(xmult(subt(l.a,l.b),pvec(s)))<eps;
197 }
198 int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
199     return vlen(xmult(subt(l1,l2),pvec(s1,s2,s3)))<eps;
200 }
201
202
203 //□□}□□□□□¸°□(¶□□□¿·□□□□
204 int intersect_in(line3 u,line3 v){
205     if (!dots_onplane(u.a,u.b,v.a,v.b))
206         return 0;
207     if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
208         return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
209     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);
210 }
211 int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2){
212     if (!dots_onplane(u1,u2,v1,v2))
213         return 0;
214     if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
215         return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
216     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);
217 }
218
219
220 //□□}□□□□□¸²»°□(¶□□□¿·□□□□
221 int intersect_ex(line3 u,line3 v){
222     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);
223 }
224 int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
225     return dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,
    u1,u2);
226 }
227
228
229 //□□□□□□□□□□□□½□□□□□¸°□(½»□□)□□□²¿·□)°□°¬
230 int intersect_in(line3 l,plane3 s){
231     return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
232         !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□□□□□□□□□□□Fµ□,□□□□□□□□□□□□□□□□□□,²¢±£□□µF¹²□□!
                                                  □
//□□□□□□□□□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;
```

```
291     return ret;
292 }
293
294
295 //¼□□□}□□Ⓕ□□,□□□□□□□□□□□□□□□□□□,²¢±£□□μⒻ¹²□□!
296 line3 intersection(plane3 u,plane3 v){
297     line3 ret;
298     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);
299     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);
300     return ret;
301 }
302 line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
303     line3 ret;
304     ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v1,v2,u1,
        u2,u3);
305     ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v3,v1,u1,
        u2,u3);
306     return ret;
307 }
308
309
310 //μⒻ□□□□□
311 double ptoline(point3 p,line3 l){
312     return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
313 }
314 double ptoline(point3 p,point3 l1,point3 l2){
315     return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
316 }
317
318
319 //μⒻ□□□□□□
320 double ptoplane(point3 p,plane3 s){
321     return fabs(dmult(pvec(s),subt(p,s.a)))/vlen(pvec(s));
322 }
323 double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){
324     return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
325 }
326
327
328 //□½□□□□□□
329 double linetoline(line3 u,line3 v){
330     point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));
331     return fabs(dmult(subt(u.a,v.a),n))/vlen(n);
332 }
333 double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){
334     point3 n=xmult(subt(u1,u2),subt(v1,v2));
335     return fabs(dmult(subt(u1,v1),n))/vlen(n);
336 }
337
338
339 //}□□□□cos
340 double angle_cos(line3 u,line3 v){
341     return dmult(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));
342 }
343 double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2){
344     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];
    }
```

```
35      top--;
36      n = top+1;
37  }
38
39
40  // ¼«½□□□
41  #include <stdio.h>
42  #include <string.h>
43  #include <algorithm>
44  #include <math.h>
45  using namespace std;
46  #define maxn 100005
47  int N;
48  struct A
49  {
50      int x,y;
51      int v,l;
52  }P[maxn];
53  int xmult(int x1,int y1,int x2,int y2,int x3,int y3)
54  {
55      return (y2-y1)*(x3-x1)-(y3-y1)*(x2-x1);
56  }
57  void swap(A &a,A &b)
58  {
59      A t = a;a = b,b = t;
60  }
61  bool operator < (A a,A b)
62  {
63      int k = xmult(P[0].x,P[0].y,a.x,a.y,b.x,b.y);
64      if( k<0 )
65          return 1;
66      else if( k==0 )
67      {
68          if( abs(P[0].x-a.x)<abs(P[0].x-b.x) )
69              return 1;
70          if( abs(P[0].y-a.y)<abs(P[0].y-b.y) )
71              return 1;
72      }
73      return 0;
74  }
75  void Grem_scan(int n)
76  {
77      int i,j,k,l;
78      k = 0x7fffffff;
79      for(i=0;i<n;i++)
80          if( P[i].x<k || P[i].x==k && P[i].y<P[l].y )
81          k = P[i].x,l = i;
82      swap(P[l],P[0]);
83      sort(P+1,P+n);
84
85      l = 3;
86      for(i=3;i<n;i++)
87      {
88          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 )
89              l--;
90          P[l++] = P[i];
91      }
92  }
93  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 = 0x7fffffff;
        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]];
```

```
153
154
155             k = -1;
156             for(i=0;i<N;i++)
157             {
158                 if( B[i] )
159                     continue;
160                 if( xmult(a,b,P[i])<0 )
161                     continue;
162                 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 )
163                     k = i;
164             }
165             if( k==-1 )
166                 break;
167             B[k] = 1;
168             as[L++] = k;
169         }
170         printf("%d ",L);
171         for(i=0;i<L;i++)
172             printf("%d ",as[i]+1);
173         printf("\n");
174     }
175 }
```

0.9 网格

```
1  #define abs(x) ((x)>0?(x):-(x))
2  struct point{int x,y;};
3
4
5  int gcd(int a,int b){return b?gcd(b,a%b):a;}
6
7
8  //¶□□□□□□□□□□¸□□□□□
9  int grid_onedge(int n,point* p){
10     int i,ret=0;
11     for (i=0;i<n;i++)
12         ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));
13     return ret;
14 }
15
16
17 //¶□□□□□□□□□□¸□□□□□
18 int grid_inside(int n,point* p){
19     int i,ret=0;
20     for (i=0;i<n;i++)
21         ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
22     return (abs(ret)-grid_onedge(n,p))/2+1;
23 }
```

0.10 圆

```
1  #include <math.h>
2  #define eps 1e-8
3  struct point{double x,y;};
4
```

```
5
6  double xmult(point p1,point p2,point p0){
7      return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
8  }
9
10
11 double distance(point p1,point p2){
12     return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
13 }
14
15
16 double disptoline(point p,point l1,point l2){
17     return fabs(xmult(p,l1,l2))/distance(l1,l2);
18 }
19
20
21 point intersection(point u1,point u2,point v1,point v2){
22     point ret=u1;
23     double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
24             /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
25     ret.x+=(u2.x-u1.x)*t;
26     ret.y+=(u2.y-u1.y)*t;
27     return ret;
28 }
29
30
31 //□□□□□□□□□□°□(□□□□
32 int intersect_line_circle(point c,double r,point l1,point l2){
33     return disptoline(c,l1,l2)<r+eps;
34 }
35
36
37 //□□□□□□□□□□°□(¶□□□□□□□□
38 int intersect_seg_circle(point c,double r,point l1,point l2){
39     double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
40     point t=c;
41     if (t1<eps||t2<eps)
42         return t1>-eps||t2>-eps;
43     t.x+=l1.y-l2.y;
44     t.y+=l2.x-l1.x;
45     return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
46 }
47
48
49 //□□□°□□□□°□(□□□□
50 int intersect_circle_circle(point c1,double r1,point c2,double r2){
51     return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
52 }
53
54
55 //¼□□□□□□½µ□p□□□µ□,□□p□□□□□□□□,·µ»□p±¾□□
56 point dot_to_circle(point c,double r,point p){
57     point u,v;
58     if (distance(p,c)<eps)
59         return p;
60     u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
61     u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
62     v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
63     v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
```

```
64      return distance(u,p)<distance(v,p)?u:v;
65  }
66
67
68  //¼□□□□□□□□□□µ□»µ□,±£□□□□□□□□»µ□
                                □
                                □
69  //¼□□□□□□□□□□µ□»µ□□□□□□□□□□°□□□□□□□□□□□□□□□□
70  void intersection_line_circle(point c,double r,point l1,point l2,point& p1,point& p2){
71      point p=c;
72      double t;
73      p.x+=l1.y-l2.y;
74      p.y+=l2.x-l1.x;
75      p=intersection(p,c,l1,l2);
76      t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(l1,l2);
77      p1.x=p.x+(l2.x-l1.x)*t;
78      p1.y=p.y+(l2.y-l1.y)*t;
79      p2.x=p.x-(l2.x-l1.x)*t;
80      p2.y=p.y-(l2.y-l1.y)*t;
81  }
82
83
84  //¼□□□□□□□□µ□»µ□,±£□□□□□□□»µ□,□□□»□□□
                                □
85  void intersection_circle_circle(point c1,double r1,point c2,double r2,point& p1,point&
        p2){
86      point u,v;
87      double t;
88      t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(c1,c2))/2;
89      u.x=c1.x+(c2.x-c1.x)*t;
90      u.y=c1.y+(c2.y-c1.y)*t;
91      v.x=u.x+c1.y-c2.y;
92      v.y=u.y-c1.x+c2.x;
93      intersection_line_circle(c1,r1,u,v,p1,p2);
94  }
95
96
97  //½«□□-p□□□□□□□□angle½□□□
98  Point Rotate(Point p,double angle) {
99      Point res;
100     res.x=p.x*cos(angle)-p.y*sin(angle);
101     res.y=p.x*sin(angle)+p.y*cos(angle);
102     return res;
103 }
104 //□□□□□□µ□□□□(o,r)µ□}¸□□□□result1°□result2
105 void TangentPoint_PC(Point poi,Point o,double r,Point &result1,Point &result2) {
106     double line=sqrt((poi.x-o.x)*(poi.x-o.x)+(poi.y-o.y)*(poi.y-o.y));
107     double angle=acos(r/line);
108     Point unitvector,lin;
109     lin.x=poi.x-o.x;
110     lin.y=poi.y-o.y;
111     unitvector.x=lin.x/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
112     unitvector.y=lin.y/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
113     result1=Rotate(unitvector,-angle);
114     result2=Rotate(unitvector,angle);
115     result1.x+=o.x;
116     result1.y+=o.y;
117     result2.x+=o.x;
118     result2.y+=o.y;
119     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) {}
};
```

```
55 struct Rect {              // □ó¤¿□□□¾□□□k½·¨ w, h·□□□¿□□□□□
56     double w, h;
57  Rect() {}
58  Rect(double _w,double _h) : w(_w),h(_h) {}
59 };
60 struct Rect_2 {             // ±□□¾□□□¬□□□¾□□□±□□□(xl, yl)£¬□□□□□□□±□□□(xh, yh)
61     double xl, yl, xh, yh;
62  Rect_2() {}
63  Rect_2(double _xl,double _yl,double _xh,double _yh) : xl(_xl),yl(_yl),xh(_xh),yh(_yh)
       {}
64 };
65 struct Circle {            //□
66  Point c;
67  double r;
68  Circle() {}
69  Circle(Point _c,double _r) :c(_c),r(_r) {}
70 };
71 typedef vector<Point> Polygon;      // ¶□□¶□□□□
72 typedef vector<Point> Points;       // ¶□□µF
73 typedef vector<Point3D> Points3D;   // □□□µF
74 ////////////////////////////////////////////////////////////////////
75
76
77 ////////////////////////////////////////////////////////////////////
78 //»□±¾°¯□□□□
79 inline double max(double x,double y)
80 {
81     return x > y ? x : y;
82 }
83 inline double min(double x, double y)
84 {
85     return x > y ? y : x;
86 }
87 inline bool ZERO(double x)              // x == 0
88 {
89     return (fabs(x) < EPS);
90 }
91 inline bool ZERO(Point p)               // p == 0
92 {
93     return (ZERO(p.x) && ZERO(p.y));
94 }
95 inline bool ZERO(Point3D p)              // p == 0
96 {
97     return (ZERO(p.x) && ZERO(p.y) && ZERO(p.z));
98 }
99 inline bool EQ(double x, double y)      // eqaul, x == y
100 {
101     return (fabs(x - y) < EPS);
102 }
103 inline bool NEQ(double x, double y)     // not equal, x != y
104 {
105     return (fabs(x - y) >= EPS);
106 }
107 inline bool LT(double x, double y)     // less than, x < y
108 {
109     return ( NEQ(x, y) && (x < y) );
110 }
111 inline bool GT(double x, double y)     // greater than, x > y
112 {
```

```
113     return ( NEQ(x, y) && (x > y) );
114 }
115 inline bool LEQ(double x, double y)     // less equal, x <= y
116 {
117     return ( EQ(x, y) || (x < y) );
118 }
119 inline bool GEQ(double x, double y)     // greater equal, x >= y
120 {
121     return ( EQ(x, y) || (x > y) );
122 }
123 // □□□£¡£¡
124 // □□□□□□□¸□□□µ□°µ□¡µ□□□
125 // ±£□□□□□□□□□□□□□°□□□□□-0.000□□□□µ□□□□£¬
126 // □□□□□□□□¸
127 // □□□□□´□□□¡£¡£¡£¡£¡
128 // □□□□□□□□□□£¬□¶¨□µ□□ô□¯□□½□□□□□□□£¡
129 inline double FIX(double x)
130 {
131     return (fabs(x) < EPS) ? 0 : x;
132 }
133 ////////////////////////////////////////////////////////////////////////////////
134
135
136 ////////////////////////////////////////////////////////////////////////////////
137 //¶□□□-□□□□
138 bool operator==(Point p1, Point p2)
139 {
140     return ( EQ(p1.x, p2.x) &&  EQ(p1.y, p2.y) );
141 }
142 bool operator!=(Point p1, Point p2)
143 {
144     return ( NEQ(p1.x, p2.x) ||  NEQ(p1.y, p2.y) );
145 }
146 bool operator<(Point p1, Point p2)
147 {
148     if (NEQ(p1.x, p2.x)) {
149         return (p1.x < p2.x);
150     } else {
151         return (p1.y < p2.y);
152     }
153 }
154 Point operator+(Point p1, Point p2)
155 {
156     return Point(p1.x + p2.x, p1.y + p2.y);
157 }
158 Point operator-(Point p1, Point p2)
159 {
160     return Point(p1.x - p2.x, p1.y - p2.y);
161 }
162 double operator*(Point p1, Point p2) // ¼□□□□□ p1 ¡□ p2
163 {
164     return (p1.x * p2.y - p2.x * p1.y);
165 }
166 double operator&(Point p1, Point p2) { // ¼□□□□□ p1¡¤p2
167     return (p1.x * p2.x + p1.y * p2.y);
168 }
169 double Norm(Point p) // ¼□□□□-pµ□ģ
170 {
171     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□ģ
{
    return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}


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


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

```
231 double Distance(Point p1, Point p2) //2µ□□ℓ□□□
232 {
233  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
234 }
235 double Distance(Point3D p1, Point3D p2) //2µ□□ℓ□□□,□□□
236 {
237  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));
238 }
239 double Distance(Point p, Line L) // □□□□□□□□□□Fℓ□□□□
240 {
241     return ( fabs((p - L.p1) * (L.p2 - L.p1)) / Norm(L.p2 - L.p1) );
242 }
243 double Distance(Point3D p, Line3D L)// □□□□□¿□□□□Fℓ□□□□
244 {
245     return ( Norm((p - L.p1) * (L.p2 - L.p1)) / Norm(L.p2 - L.p1) );
246 }
247 bool OnLine(Point p, Line L) // □□□□□□□□□□□p□□□□□□□L□□
248 {
249     return ZERO( (p - L.p1) * (L.p2 - L.p1) );
250 }
251 bool OnLine(Point3D p, Line3D L) // □□□□□□¿□□□□p□□□□□□L□□
252 {
253     return ZERO( (p - L.p1) * (L.p2 - L.p1) );
254 }
255 int Relation(Point p, Line L) // ¾□□□p□□□Lµ□□□□□ , · µ»□ONLINE,LEFT,RIGHT
256 {
257     double res = (L.p2 - L.p1) * (p - L.p1);
258     if (EQ(res, 0)) {
259         return ONLINE;
260     } else if (res > 0) {
261         return LEFT;
262     } else {
263         return RIGHT;
264     }
265 }
266 bool SameSide(Point p1, Point p2, Line L) // □□□p1, p2□□□□□□L□□
267 {
268     double m1 = (p1 - L.p1) * (L.p2 - L.p1);
269     double m2 = (p2 - L.p1) * (L.p2 - L.p1);
270     return GT(m1 * m2, 0);
271 }
272 bool OnLineSeg(Point p, Line L) // □□□□□□□□□□□p□□□□□□□□l□□
273 {
274     return ( ZERO( (L.p1 - p) * (L.p2 - p) ) &&
275         LEQ((p.x - L.p1.x)*(p.x - L.p2.x), 0) &&
276         LEQ((p.y - L.p1.y)*(p.y - L.p2.y), 0) );
277 }
278 bool OnLineSeg(Point3D p, Line3D L) // □□□□□□¿□□□□p□□□□□□□□l□□
279 {
280     return ( ZERO((L.p1 - p) * (L.p2 - p)) &&
281         EQ( Norm(p - L.p1) + Norm(p - L.p2), Norm(L.p2 - L.p1)) );
282 }
283 Point SymPoint(Point p, Line L) // □□□□□□□□□□p¹□□□□Lµ□□□
284 {
285     Point result;
286     double a = L.p2.x - L.p1.x;
287     double b = L.p2.y - L.p1.y;
288     double t = ( (p.x - L.p1.x) * a + (p.y - L.p1.y) * b ) / (a*a + b*b);
289     result.x = 2 * L.p1.x + 2 * a * t - p.x;
```

```
290     result.y = 2 * L.p1.y + 2 * b * t - p.y;
291     return result;
292 }
293 bool Coplanar(Points3D points) // □□□□¸□Ⓕ□□□□□□□□□²¿¹²□□
294 {
295     int i;
296     Point3D p;
297 
298 
299     if (points.size() < 4) return true;
300     p = (points[2] - points[0]) * (points[1] - points[0]);
301     for (i = 3; i < points.size(); i++) {
302         if (! ZERO(p & points[i]) ) return false;
303     }
304     return true;
305 }
306 bool LineIntersect(Line L1, Line L2) // □□□□□µ□}□□□□□□□
307 {
308     return (! ZERO((L1.p1 - L1.p2)*(L2.p1 - L2.p2)) );  // □□□□□□□
309 }
310 bool LineIntersect(Line3D L1, Line3D L2) // □□□□□□□µ□}□□□□□□□
311 {
312     Point3D p1 = L1.p1 - L1.p2;
313     Point3D p2 = L2.p1 - L2.p2;
314     Point3D p  = p1 * p2;
315     if (ZERO(p)) return false;      // □□□□□□□
316     p = (L2.p1 - L1.p2) * (L1.p1 - L1.p2);
317     return ZERO(p & L2.p2);         // □□□□□□
318 }
319 bool LineSegIntersect(Line L1, Line L2) // □□□□□µ□}□□□□□□□□□□□
320 {
321     return ( GEQ( max(L1.p1.x, L1.p2.x), min(L2.p1.x, L2.p2.x) ) &&
322         GEQ( max(L2.p1.x, L2.p2.x), min(L1.p1.x, L1.p2.x) ) &&
323         GEQ( max(L1.p1.y, L1.p2.y), min(L2.p1.y, L2.p2.y) ) &&
324         GEQ( max(L2.p1.y, L2.p2.y), min(L1.p1.y, L1.p2.y) ) &&
325         LEQ( ((L2.p1 - L1.p1) * (L1.p2 - L1.p1)) * ((L2.p2 -  L1.p1) * (L1.p2 - L1.p1))
    , 0 ) &&
326         LEQ( ((L1.p1 - L2.p1) * (L2.p2 - L2.p1)) * ((L1.p2 -  L2.p1) * (L2.p2 - L2.p1))
    , 0 ) );
327 }
328 bool LineSegIntersect(Line3D L1, Line3D L2) // □□□□□□□µ□}□□□□□□□□□□□
329 {
330     // todo
331     return true;
332 }
333 // ¼□□□}□□□□□□Ľ»µ□Ⓕ½□□□□□□□□P□□µ»□
334 // ·µ»□□{□□□□□□□□□□□□□□□□□□ù□□:  COLINE   -- ¹²□□  PARALLEL -- □□□  CROSS    -- □□
335 int CalCrossPoint(Line L1, Line L2, Point& P)
336 {
337     double A1, B1, C1, A2, B2, C2;
338 
339 
340     A1 = L1.p2.y - L1.p1.y;
341     B1 = L1.p1.x - L1.p2.x;
342     C1 = L1.p2.x * L1.p1.y - L1.p1.x * L1.p2.y;
343 
344 
345     A2 = L2.p2.y - L2.p1.y;
```

```
346     B2 = L2.p1.x - L2.p2.x;
347     C2 = L2.p2.x * L2.p1.y - L2.p1.x * L2.p2.y;
348
349
350     if (EQ(A1 * B2, B1 * A2))    {
351         if (EQ( (A1 + B1) * C2, (A2 + B2) * C1 )) {
352             return COLINE;
353         } else {
354             return PARALLEL;
355         }
356     } else {
357         P.x = (B2 * C1 - B1 * C2) / (A2 * B1 - A1 * B2);
358         P.y = (A1 * C2 - A2 * C1) / (A2 * B1 - A1 * B2);
359         return CROSS;
360     }
361 }
362 // ¼□□□}□□□□□□□□□□»µ□Ⓕ½□□□□□□□□P□□µ»□
363 //  ·µ»□□{□□□□□□□□□□□□□□□□□□ù□□ COLINE    -- ¹²□□  PARALLEL -- □□□  CROSS    -- □□ NONCOPLANAR
    -- ²»¹«□□
364 int CalCrossPoint(Line3D L1, Line3D L2, Point3D& P)
365 {
366     // todo
367     return 0;
368 }
369 // ¼□□□□Pµ½□□Lµ□□□□µ□
370 Point NearestPointToLine(Point P, Line L)
371 {
372     Point result;
373     double a, b, t;
374
375
376     a = L.p2.x - L.p1.x;
377     b = L.p2.y - L.p1.y;
378     t = ( (P.x - L.p1.x) * a + (P.y - L.p1.y) * b ) / (a * a + b * b);
379
380
381     result.x = L.p1.x + a * t;
382     result.y = L.p1.y + b * t;
383     return result;
384 }
385 // ¼□□□□Pµ½□□□Lµ□□□□µ□
386 Point NearestPointToLineSeg(Point P, Line L)
387 {
388     Point result;
389     double a, b, t;
390
391
392     a = L.p2.x - L.p1.x;
393     b = L.p2.y - L.p1.y;
394     t = ( (P.x - L.p1.x) * a + (P.y - L.p1.y) * b ) / (a * a + b * b);
395
396
397     if ( GEQ(t, 0) && LEQ(t, 1) ) {
398         result.x = L.p1.x + a * t;
399         result.y = L.p1.y + b * t;
400     } else {
401         if ( Norm(P - L.p1) < Norm(P - L.p2) ) {
402             result = L.p1;
```

```
403         } else {
404             result = L.p2;
405         }
406     }
407     return result;
408 }
409 // ¾□□□□□□□L1µ½□□□L2µ□□□□□□□
410 double MinDistance(Line L1, Line L2)
411 {
412     double d1, d2, d3, d4;
413 
414 
415     if (LineSegIntersect(L1, L2)) {
416         return 0;
417     } else {
418         d1 = Norm( NearestPointToLineSeg(L1.p1, L2) - L1.p1 );
419         d2 = Norm( NearestPointToLineSeg(L1.p2, L2) - L1.p2 );
420         d3 = Norm( NearestPointToLineSeg(L2.p1, L1) - L2.p1 );
421         d4 = Norm( NearestPointToLineSeg(L2.p2, L1) - L2.p2 );
422 
423         return min( min(d1, d2), min(d3, d4) );
424     }
425 }
426 // □□□□□}□¬¸□□□□
                  □
427 //  · µ»□□Ð□□Pi¼□Ļ¡¶□
428 double Inclination(Line L1, Line L2)
429 {
430     Point u = L1.p2 - L1.p1;
431     Point v = L2.p2 - L2.p1;
432     return acos( (u & v) / (Norm(u)*Norm(v)) );
433 }
434 // □□□□□□}□¬¸□□□□
                  □
435 //  · µ»□□Ð□□Pi¼□Ļ¡¶□
436 double Inclination(Line3D L1, Line3D L2)
437 {
438     Point3D u = L1.p2 - L1.p1;
439     Point3D v = L2.p2 - L2.p1;
440     return acos( (u & v) / (Norm(u)*Norm(v)) );
441 }
442 ////////////////////////////////////////////////////////////////////////////////
443 
444 
445 ////////////////////////////////////////////////////////////////////////////////
446 // □□□}¸□□□□□□□□□
447 // □□□□□□□«□□□□□
448 bool Intersect(Rect_2 r1, Rect_2 r2)
449 {
450     return ( max(r1.xl, r2.xl) < min(r1.xh, r2.xh) &&
451              max(r1.yl, r2.yl) < min(r1.yh, r2.yh) );
452 }
453 // □□□□□□□r2□□□□□□□□□□□□□□□□r1□□
454 // r2¿□□□□□□□□□□□□
455 // · ¢□□□4µ□□³□k½ · ¨¹□²»□□OJ□□□□□□□□□□□□□□
456 //□□□□□□□□□□□°µ□□□□
457 bool IsContain(Rect r1, Rect r2)      //¾□□□□w>h
458  {
459      if(r1.w >r2.w && r1.h > r2.h) return true;
```

```
460       else
461       {
462          double r = sqrt(r2.w*r2.w + r2.h*r2.h) / 2.0;
463          double alpha = atan2(r2.h,r2.w);
464          double sita = asin((r1.h/2.0)/r);
465          double x = r * cos(sita - 2*alpha);
466          double y = r * sin(sita - 2*alpha);
467          if(x < r1.w/2.0 && y < r1.h/2.0 && x > 0 && y > -r1.h/2.0) return true;
468          else return false;
469       }
470  }
471  ////////////////////////////////////////////////////////////////////////
472
473
474  ////////////////////////////////////////////////////////////////////////
475  //□
476  Point Center(const Circle & C) //□□□□
477  {
478      return C.c;
479  }
480
481
482  double Area(const Circle &C)
483  {
484   return pi*C.r*C.r;
485  }
486
487
488  double CommonArea(const Circle & A, const Circle & B) //}¸□□µĹ«¹²□□□
489  {
490      double area = 0.0;
491      const Circle & M = (A.r > B.r) ? A : B;
492      const Circle & N = (A.r > B.r) ? B : A;
493      double D = Distance(Center(M), Center(N));
494      if ((D < M.r + N.r) && (D > M.r - N.r))
495      {
496          double cosM = (M.r * M.r + D * D - N.r * N.r) / (2.0 * M.r * D);
497          double cosN = (N.r * N.r + D * D - M.r * M.r) / (2.0 * N.r * D);
498          double alpha = 2.0 * acos(cosM);
499          double beta  = 2.0 * acos(cosN);
500          double TM = 0.5 * M.r * M.r * sin(alpha);
501          double TN = 0.5 * N.r * N.r * sin(beta);
502          double FM = (alpha / (2*pi)) * Area(M);
503          double FN = (beta / (2*pi)) * Area(N);
504          area = FM + FN - TM - TN;
505      }
506      else if (D <= M.r - N.r)
507      {
508          area = Area(N);
509      }
510      return area;
511  }
512
513  bool IsInCircle(const Circle & C, const Rect_2 & rect)//□□□□□□□□□□)□□□□□²»□□□□□□□□)
514  {
515      return (GT(C.c.x - C.r, rect.xl)
516    &&  LT(C.c.x + C.r, rect.xh)
517    &&  GT(C.c.y - C.r, rect.yl)
518    &&  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□□Fº□(¶□□)
#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;
```

```
30    POINT() : x(0), y(0) {};
31    POINT(TYPE _x_, TYPE _y_) : x(_x_), y(_y_) {};
32  };
33  // }¸□□?□□□
34  TYPE Distance(const POINT & a, const POINT & b)
35  {
36    return Sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
37  }
38  //□□□
39  struct SEG
40  {
41    POINT a; //□□□
42    POINT b; //□□□
43    SEG() {};
44    SEG(POINT _a_, POINT _b_):a(_a_),b(_b_) {};
45  };
46  //¶)□□µ□□)
47  struct LINE
48  {
49    POINT a;
50    POINT b;
51    LINE() {};
52    LINE(POINT _a_, POINT _b_) : a(_a_), b(_b_) {};
53  };
54  //Ù□□□°□□)
55  struct LINE2
56  {
57    TYPE A,B,C;
58    LINE2() {};
59    LINE2(TYPE _A_, TYPE _B_, TYPE _C_) : A(_A_), B(_B_), C(_C_) {};
60  };
61
62
63  //}µ□□»¯□°□□
64  LINE2 Line2line(const LINE & L) // y=kx+c k=y/x
65  {
66    LINE2 L2;
67    L2.A = L.b.y - L.a.y;
68    L2.B = L.a.x - L.b.x;
69    L2.C = L.b.x * L.a.y - L.a.x * L.b.y;
70    return L2;
71  }
72
73
74  // □□□÷µ»□□□ Ax + By + C =0 µ□□□□
75  void Coefficient(const LINE & L, TYPE & A, TYPE & B, TYPE & C)
76  {
77    A = L.b.y - L.a.y;
78    B = L.a.x - L.b.x;
79    C = L.b.x * L.a.y - L.a.x * L.b.y;
80  }
81  void Coefficient(const POINT & p,const TYPE a,TYPE & A,TYPE & B,TYPE & C)
82  {
83    A = Cos(a);
84    B = Sin(a);
85    C = - (p.y * B + p.x * A);
86  }
87  /□□□(Æ¬µ🄵Q□□
88  bool IsEqual(TYPE a, TYPE b)
```

```
89  {
90    return (Abs(a - b) <Epsilon);
91  }
92  bool IsEqual(const POINT & a, const POINT & b)
93  {
94    return (IsEqual(a.x, b.x) && IsEqual(a.y, b.y));
95  }
96  bool IsEqual(const LINE & A, const LINE & B)
97  {
98    TYPE A1, B1, C1;
99    TYPE A2, B2, C2;
100   Coefficient(A, A1, B1, C1);
101   Coefficient(B, A2, B2, C2);
102   return IsEqual(A1 * B2, A2 * B1) && IsEqual(A1 * C2, A2 * C1) && IsEqual(B1 * C2, B2
        * C1);
103 }
104 // ¾□□□
105 struct RECT
106 {
107   POINT a; // □□□μ□
108   POINT b; // □□□□□
109   RECT() {};
110   RECT(const POINT & _a_, const POINT & _b_) { a = _a_; b = _b_; }
111 };
112
113
114 //¾□□□¯±□□
115 RECT Stdrect(const RECT & q)
116 {
117   TYPE t;
118   RECT p=q;
119   if(p.a.x > p.b.x) swap(p.a.x , p.b.x);
120   if(p.a.y > p.b.y) swap(p.a.y , p.b.y);
121   return p;
122 }
123
124
125 //¸□¾□□±Ⓕ□«□□□ı□
126 SEG Edge(const RECT & rect, int idx)
127 {
128   SEG edge;
129   while (idx < 0) idx += 4;
130   switch (idx % 4)
131   {
132   case 0: //□±□
133     edge.a = rect.a;
134     edge.b = POINT(rect.b.x, rect.a.y);
135     break;
136   case 1: //□□□
137     edge.a = POINT(rect.b.x, rect.a.y);
138     edge.b = rect.b;
139     break;
140   case 2: //□□□
141     edge.a = rect.b;
142     edge.b = POINT(rect.a.x, rect.b.y);
143     break;
144   case 3: //□□□
145     edge.a = POINT(rect.a.x, rect.b.y);
146     edge.b = rect.a;
```

```
147       break;
148     default:
149       break;
150     }
151     return edge;
152   }
153
154
155   //¾ØÐÎÃæ»ý
156   TYPE Area(const RECT & rect)
157   {
158     return (rect.b.x - rect.a.x) * (rect.b.y - rect.a.y);
159   }
160
161
162   //¾ØÐÎÓëÔ²µÄ¹«¹²Ãæ»ý
163   TYPE CommonArea(const RECT & A, const RECT & B)
164   {
165     TYPE area = 0.0;
166     POINT LL(Max(A.a.x, B.a.x), Max(A.a.y, B.a.y));
167     POINT UR(Min(A.b.x, B.b.x), Min(A.b.y, B.b.y));
168     if( (LL.x <= UR.x) && (LL.y <= UR.y) )
169     {
170       area = Area(RECT(LL, UR));
171     }
172     return area;
173   }
174   //ÅÐ¶ÏÔ²ÐÎÊÇ·ñÍêÈ«°üº¬ÔÚ¾ØÐÎÄÚ)
175   bool IsInCircle(const CIRCLE & circle, const RECT & rect)
176   {
177     return (circle.x - circle.r > rect.a.x) &&
178       (circle.x + circle.r < rect.b.x) &&
179       (circle.y - circle.r > rect.a.y) &&
180       (circle.y + circle.r < rect.b.y);
181   }
182
183
184   //ÅÐ¶ÏÔ²ÐÎºÍ¾ØÐÎÊÇ·ñÏà½»(°üº¬)
185   bool IsInRect(const CIRCLE & circle, const RECT & rect)
186   {
187     POINT c,d;
188     c.x=rect.a.x; c.y=rect.b.y;
189     d.x=rect.b.x; d.y=rect.a.y;
190     return (Distance( Center(circle) , rect.a ) < circle.r) &&
191       (Distance( Center(circle) , rect.b ) < circle.r) &&
192       (Distance( Center(circle) , c ) < circle.r) &&
193       (Distance( Center(circle) , d ) < circle.r);
194   }
195
196
197   //ÅÐ¶ÏÔ²ÐÎºÍ¾ØÐÎÊÇ·ñÏà½»(°üº¬)
198   bool Isoutside(const CIRCLE & circle, const RECT & rect)
199   {
200     POINT c,d;
201     c.x=rect.a.x; c.y=rect.b.y;
202     d.x=rect.b.x; d.y=rect.a.y;
203     return (Distance( Center(circle) , rect.a ) > circle.r) &&
204       (Distance( Center(circle) , rect.b ) > circle.r) &&
205       (Distance( Center(circle) , c ) > circle.r) &&
```

```
206     (Distance( Center(circle) , d ) > circle.r) &&
207     (rect.a.x > circle.x || circle.x > rect.b.x || rect.a.y > circle.y || circle.y >
        rect.b.y) ||
208     ((circle.x - circle.r > rect.b.x) ||
209     (circle.x + circle.r < rect.a.x) ||
210     (circle.y - circle.r > rect.b.y) ||
211     (circle.y + circle.r < rect.a.y));
212 }
```

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;
}
```

```
56
57
58  TCircle circumcircleOfTriangle(TTriangle t)
59  {
60      //□□½□□□□□□□□□
61      TCircle tmp;
62      double a, b, c, c1, c2;
63      double xA, yA, xB, yB, xC, yC;
64      a = distance(t.t[0], t.t[1]);
65      b = distance(t.t[1], t.t[2]);
66      c = distance(t.t[2], t.t[0]);
67      //¸□¾□S = a * b * c / R / 4;□□Ⓕ R
68      tmp.r = a * b * c / triangleArea(t) / 4;
69
70      xA = t.t[0].x; yA = t.t[0].y;
71      xB = t.t[1].x; yB = t.t[1].y;
72      xC = t.t[2].x; yC = t.t[2].y;
73      c1 = (xA * xA + yA * yA - xB * xB - yB * yB) / 2;
74      c2 = (xA * xA + yA * yA - xC * xC - yC * yC) / 2;
75
76      tmp.centre.x = (c1 * (yA - yC) - c2 * (yA - yB)) /
77           ((xA - xB) * (yA - yC) - (xA - xC) * (yA - yB));
78      tmp.centre.y = (c1 * (xA - xC) - c2 * (xA - xB)) /
79           ((yA - yB) * (xA - xC) - (yA - yC) * (xA - xB));
80
81      return tmp;
82  }
83
84
85  TCircle MinCircle2(int tce, TTriangle ce)
86  {
87      TCircle tmp;
88      if(tce == 0) tmp.r = -2;
89      else if(tce == 1)
90      {
91          tmp.centre = ce.t[0];
92          tmp.r = 0;
93      }
94      else if(tce == 2)
95      {
96          tmp.r = distance(ce.t[0], ce.t[1]) / 2;
97          tmp.centre.x = (ce.t[0].x + ce.t[1].x) / 2;
98          tmp.centre.y = (ce.t[0].y + ce.t[1].y) / 2;
99      }
100     else if(tce == 3) tmp = circumcircleOfTriangle(ce);
101     return tmp;
102 }
103
104
105 void MinCircle(int t, int tce, TTriangle ce)
106 {
107     int i, j;
108     TPoint tmp;
109     c = MinCircle2(tce, ce);
110     if(tce == 3) return;
111     for(i = 1;i <= t;i++)
112     {
113         if(distance(a[i], c.centre) > c.r)
114         {
```

```
            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++)
```

```
80      {
81          tmp = polygon.p[i];
82          polygon.p[i] = polygon.p[polygon.n - 1 - i];
83          polygon.p[polygon.n - 1 - i] = tmp;
84      }
85  }
86
87
88  double disPointToSeg(TPoint p1, TPoint p2, TPoint p3)
89  {
90      double a = distance(p1, p2);
91      double b = distance(p1, p3);
92      double c = distance(p2, p3);
93      if(fabs(a + b - c) < eps) return 0;
94      if(fabs(a + c - b) < eps || fabs(b + c - a) < eps) return min(a, b);
95      double t1 = -a * a + b * b + c * c;
96      double t2 = a * a - b * b + c * c;
97      if(t1 <= 0 || t2 <= 0) return min(a, b);
98
99      TLine l1 = lineFromSegment(p2, p3);
100     return fabs(l1.a * p1.x + l1.b * p1.y + l1.c) / sqrt(l1.a * l1.a + l1.b * l1.b);
101 }
102
103
104 double disPallSeg(TPoint p1, TPoint p2, TPoint p3, TPoint p4)
105 {
106     return min(min(disPointToSeg(p1, p3, p4), disPointToSeg(p2, p3, p4)),
107      min(disPointToSeg(p3, p1, p2), disPointToSeg(p4, p1, p2)));
108 }
109
110
111 double angle(TPoint p1, TPoint p2, double SlewRate)
112 {
113     double ang, tmp;
114     TPoint p;
115     p.x = p2.x - p1.x;
116     p.y = p2.y - p1.y;
117     if(fabs(p.x) < eps)
118     {
119         if(p.y > 0) ang = pi / 2;
120         else ang = 3 * pi / 2;
121     }
122     else
123     {
124         ang = atan(p.y / p.x);
125         if(p.x < 0) ang += pi;
126     }
127     while(ang < 0) ang += 2 * pi;
128     if(ang >= pi) SlewRate += pi;
129     if(ang > SlewRate) tmp = ang - SlewRate;
130     else tmp = pi - (SlewRate - ang);
131     while(tmp >= pi) tmp -= pi;
132     if(fabs(tmp - pi) < eps) tmp = 0;
133     return tmp;
134 }
135
136
137 int main()
138 {
```

```
139     int n, m, i;
140     TPolygon polygon1, polygon2;
141     double ymin1, ymax2, ans, d;
142     int k1, k2;
143     while(scanf("%d%d", &n, &m) && n)
144     {
145         polygon1.n = n;
146         polygon2.n = m;
147         for(i = 0;i < n;i++)
148             scanf("%lf%lf", &polygon1.p[i].x, &polygon1.p[i].y);
149         for(i = 0;i < m;i++)
150             scanf("%lf%lf", &polygon2.p[i].x, &polygon2.p[i].y);
151         if(polygonArea(polygon1) < 0) ChangeClockwise(polygon1);
152         if(polygonArea(polygon2) < 0) ChangeClockwise(polygon2);
153         ymin1 = inf, ymax2 = -inf;
154         for(i = 0;i < n;i++)
155             if(polygon1.p[i].y < ymin1) ymin1 = polygon1.p[i].y , k1 = i;
156         for(i = 0;i < m;i++)
157             if(polygon2.p[i].y > ymax2) ymax2 = polygon2.p[i].y , k2 = i;
158         double SlewRate = 0;
159         double angle1, angle2;
160         ans = inf;
161         double Slope = 0;
162         while(Slope <= 360)
163         {
164             while(SlewRate >= pi) SlewRate -= pi;
165             if(fabs(pi - SlewRate) < eps) SlewRate = 0;
166             angle1 = angle(polygon1.p[k1], polygon1.p[(k1 + 1) % n], SlewRate);
167             angle2 = angle(polygon2.p[k2], polygon2.p[(k2 + 1) % m], SlewRate);
168             if(fabs(angle1 - angle2) < eps)
169             {
170                 d = disPallSeg(polygon1.p[k1], polygon1.p[(k1 + 1) % n], polygon2.p[k2
    ], polygon2.p[(k2 + 1) % m]);
171                 if(d < ans) ans = d;
172                 k1++;
173                 k1 %= n;
174                 k2++;
175                 k2 %= m;
176                 SlewRate += angle1;
177                 Slope += angle1;
178             }
179             else if(angle1 < angle2)
180             {
181                 d = disPointToSeg(polygon2.p[k2], polygon1.p[k1], polygon1.p[(k1 + 1) %
     n]);
182                 if(d < ans) ans = d;
183                 k1++;
184                 k1 %= n;
185                 SlewRate += angle1;
186                 Slope += angle1;
187             }
188             else
189             {
190                 d = disPointToSeg(polygon1.p[k1], polygon2.p[k2], polygon2.p[(k2 + 1) %
     m]);
191                 if(d < ans) ans = d;
192                 k2++;
193                 k2 %= m;
194                 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;
};
```

```
32
33  int main()
34  {
35      double w1, wm1, j1, jm1, wd1, wd2;
36      double w2, wm2, j2, jm2, jd1, jd2;
37      TPoint p1, p2;
38      char chr1, chr2;
39      while(scanf("%lf%lf ", &w1, &wm1) != EOF){
40          scanf("%c ", &chr1);
41          scanf("%lf %lf %c", &j1, &jm1, &chr2);
42          wd1 = (w1 + wm1 / 60) * pi / 180;
43          jd1 = (j1 + jm1 / 60) * pi / 180;
44          if(chr1 == 'S') wd1 *= -1.0;
45          if(chr2 == 'W') jd1 *= -1.0;
46          p1.x = cos(wd1) * cos(jd1);
47          p1.y = cos(wd1) * sin(jd1);
48          p1.z = sin(wd1);
49          scanf("%lf %lf %c %lf %lf %c", &w2, &wm2, &chr1, &j2, &jm2, &chr2);
50          wd2 = (w2 + wm2 / 60) * pi / 180;
51          jd2 = (j2 + jm2 / 60) * pi / 180;
52          if(chr1 == 'S') wd2 *= -1.0;
53          if(chr2 == 'W') jd2 *= -1.0;
54          p2.x = cos(wd2) * cos(jd2);
55          p2.y = cos(wd2) * sin(jd2);
56          p2.z = sin(wd2);
57          double a = acos(p1.x * p2.x + p1.y * p2.y + p1.z * p2.z);
58          printf("%.3lf\n", a * 6370.0);
59      }
60      return 0;
61  }
```

1.5 多边形重心

```
1   /*
2   □□Ł□□□□
3   □□□¸□□□□□□□□□□□N¶□□□□(3 <= N <= 1000000)£¬¿□□¾□¿□□□□£¬µ«û□□□□□®Ⓕ
4   □□□□□□□N¸□)□□□□□□□□□□□□□□□)¶¥µ□□□□±□□£¬µ□j¸□□±□□£¨Xi £¬ Yi £©£¬□□□□
5   □□ (|Xi|, |Yi| <= 20000)£¬□□□□□□□□□□□□□□□□□ġ£
6     ½□□□□³°
7     ´□□1¸·□l½□□i, i+1¸□□□□½□□□Ti,1 < i < n,
8     □¹²n-2¸□□□½□□□□□°□□□□l□□□□¸¬·□□□□ÿ¸□□□½□□□□□□□Si£¬
9     □□□□□□¸□□□□□□□□□
10    ¸□¾□□□□□□□µã°n¸□□(xi,yi)ÿ¸□□□-□□mi,□□□□□□□□□
11    X = (x1*M1+x2*M2+...+xn*Mn) / (M1+M2+....+Mn)
12    Y = (y1*M1+y2*M2+...+yn*Mn) / (M1+M2+....+Mn)
13    □□□□□□□□õ□□□□°
14    □□Bµ□□□□□½□□□□□□□£¬□□□µ□□□p[0].x, p[0].y p[1].x, p[1].y p[1].x, p[1].y
15    □□□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 ,
16    □□□□□□3¸□□□□□□□□□□□£¬□□□□□¸°¡£
17    □□Bµ□□□□□□ġ¬x = (p[0].x+p[1].x+p[2].x)/3.0 , y = (p[0].y+p[1].y+p[2].y)/3.0
18    □□□□□□□□□□Ĺ□³□□□¬²»□□□¿¼□□□□□□□□□□□□□□□□□□□Ⓕ□□□□□□□Ⓕ□□□°□□□□□□□¬
19    »¹□□□□ □µ□□□»±□□□□□ÿ¸□□□□½□□□□□□□□□□¶¾³□□□3£¬¿□□□□□□□□□□□□□
20
21  */
22  /*fzu_1132*/
23  #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

```
/*¶□□□□Í□*/
#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
```

```
18  {
19      TPoint p[Maxn];
20      int n;
21  };
22
23
24  typedef struct TLine
25  {
26      double a, b, c;
27  }TLine;
28
29
30  bool same(TPoint p1, TPoint p2)
31  {
32      if(p1.x != p2.x) return false;
33      if(p1.y != p2.y) return false;
34      return true;
35  }
36
37  double multi(TPoint p1, TPoint p2, TPoint p0)
38  {
39      //□□□-[p0, p1], [p0, p2]μ□□□
40      //p0□□¥μ□
41      return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
42      //□□□□μ□□□0£¬□□□□□□μF□□
43      //□□□□´□□□0£¬□□p0p2□□p0p1μ□□□□□F□□
44      //□□□□□□□0£¬□□p0p2□□p0p1μ□□□□F□□
45  }
46
47
48  TLine lineFromSegment(TPoint p1, TPoint p2)
49  {
50      //□□□□□□□□□,·μ»□□□■½□□□¸□□□
51      TLine tmp;
52      tmp.a = p2.y - p1.y;
53      tmp.b = p1.x - p2.x;
54      tmp.c = p2.x * p1.y - p1.x * p2.y;
55      return tmp;
56  }
57
58
59  TPoint LineInter(TLine l1, TLine l2)
60  {
61      //□□}□ý»μ□□□□±□
62      TPoint tmp;
63      double a1 = l1.a;
64      double b1 = l1.b;
65      double c1 = l1.c;
66      double a2 = l2.a;
67      double b2 = l2.b;
68      double c2 = l2.c;
69      //□□□□□□□b1 = 0
70      if(fabs(b1) < eps){
71          tmp.x = -c1 / a1;
72          tmp.y = (-c2 - a2 * tmp.x) / b2;
73      }
74      else{
75          tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
76          tmp.y = (-c1 - a1 * tmp.x) / b1;
```

```
77      }
78      return tmp;
79  }
80
81
82  TPolygon Cut_polygon(TPoint p1, TPoint p2, TPolygon polygon)
83  {
84      TPolygon new_polygon;
85      TPoint interp;
86      TLine l1, l2;
87      int i, j;
88      double t1, t2;
89      new_polygon.n = 0;
90      for(i = 0;i <= polygon.n - 1;i++){
91          t1 = multi(p2, polygon.p[i], p1);
92          t2 = multi(p2, polygon.p[i + 1], p1);
93          if(fabs(t1) < eps || fabs(t2) < eps){
94              if(fabs(t1) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i];
95              if(fabs(t2) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
96          }
97          else if(t1 < 0 && t2 < 0){
98              new_polygon.p[new_polygon.n++] = polygon.p[i];
99              new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
100         }
101         else if(t1 * t2  < 0){
102             l1 = lineFromSegment(p1, p2);
103             l2 = lineFromSegment(polygon.p[i], polygon.p[i + 1]);
104             interp = LineInter(l1, l2);
105             if(t1 < 0) {
106                 new_polygon.p[new_polygon.n++] = polygon.p[i];
107                 new_polygon.p[new_polygon.n++] = interp;
108             }
109             else {
110                 new_polygon.p[new_polygon.n++] = interp;
111                 new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
112             }
113         }
114     }
115     polygon.n = 0;
116     if(new_polygon.n == 0) return polygon;
117     polygon.p[polygon.n++] = new_polygon.p[0];
118     for(i = 1;i < new_polygon.n;i++){
119         if(!same(new_polygon.p[i], new_polygon.p[i - 1])){
120             polygon.p[polygon.n++] = new_polygon.p[i];
121         }
122     }
123     if(polygon.n != 1 && same(polygon.p[polygon.n - 1], polygon.p[0])) polygon.n--;
124     polygon.p[polygon.n] = polygon.p[0];
125     return polygon;
126 }
127
128
129 double polygonArea(TPolygon p)
130 {
131     //□□¶□□□□□□□□±□□□□□□□□
132     int i, n;
133     double area;
134     n = p.n;
135     area = 0;
```

```
136     for(i = 1;i <= n;i++){
137         area += (p.p[i - 1].x * p.p[i % n].y - p.p[i % n].x * p.p[i - 1].y);
138     }
139     return area / 2;
140 }
141 
142 
143 void ChangeClockwise(TPolygon &polygon)
144 {
145     TPoint tmp;
146     int i;
147     for(i = 0;i <= (polygon.n - 1) / 2;i++){
148         tmp = polygon.p[i];
149         polygon.p[i] = polygon.p[polygon.n - 1 - i];
150         polygon.p[polygon.n - 1 - i] = tmp;
151     }
152 }
153 
154 
155 int main()
156 {
157     int test, i, j;
158     double area;
159     TPolygon polygon, new_polygon;
160     scanf("%d", &test);
161     while(test--){
162         scanf("%d", &polygon.n);
163         for(i = 0;i <= polygon.n - 1;i++){
164             scanf("%lf%lf", &polygon.p[i].x, &polygon.p[i].y);
165         }
166         /*□□□□□□□□□□□□-«□□□□□□*/
167         if(polygonArea(polygon) > 0) ChangeClockwise(polygon);
168         polygon.p[polygon.n] = polygon.p[0];
169         new_polygon = polygon;
170         for(i = 0;i <= polygon.n - 1;i++){
171             new_polygon = Cut_polygon(polygon.p[i], polygon.p[i + 1], new_polygon);
172         }
173         area = polygonArea(new_polygon);
174         if(area < 0) printf("%.2lf\n", -area);
175         else printf("%.2lf\n", area);
176     }
177     return 0;
178 }
179 
180 
181 //□□□□□□□
182 
183 
184 #include <stdio.h>
185 #include <math.h>
186 
187 
188 #define Maxn 3005
189 const double eps = 1e-10;
190 
191 
192 typedef struct TPodouble
193 {
194     double x;
```

```
195     double y;
196 }TPoint;
197
198
199 typedef struct TPolygon
200 {
201     TPoint p[Maxn];
202     int n;
203 };
204
205
206 typedef struct TLine
207 {
208     double a, b, c;
209 }TLine;
210
211
212 bool same(TPoint p1, TPoint p2)
213 {
214     if(p1.x != p2.x) return false;
215     if(p1.y != p2.y) return false;
216     return true;
217 }
218
219 double multi(TPoint p1, TPoint p2, TPoint p0)
220 {
221     //□□□-[p0, p1], [p0, p2]μ□□□
222     //p0□□¥μ□
223     return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
224     //□□□□μ□□□0£¬□□□□□□μⒻ□□
225     //□□□□´□□□0£¬□□p0p2□□p0p1μ□□□□□Ⓕ□□
226     //□□□□□□□□0£¬□□p0p2□□p0p1μ□□□□Ⓕ□□
227 }
228
229
230 TLine lineFromSegment(TPoint p1, TPoint p2)
231 {
232     //□□□□□□□□□□,·μ»□□□⅜½□□□¸□□□
233     TLine tmp;
234     tmp.a = p2.y - p1.y;
235     tmp.b = p1.x - p2.x;
236     tmp.c = p2.x * p1.y - p1.x * p2.y;
237     return tmp;
238 }
239
240
241 TPoint LineInter(TLine l1, TLine l2)
242 {
243     //□□}□ý»μ□□□□±□
244     TPoint tmp;
245     double a1 = l1.a;
246     double b1 = l1.b;
247     double c1 = l1.c;
248     double a2 = l2.a;
249     double b2 = l2.b;
250     double c2 = l2.c;
251     //□□□□□□□b1 = 0
252     if(fabs(b1) < eps){
253         tmp.x = -c1 / a1;
```

```
254         tmp.y = (-c2 - a2 * tmp.x) / b2;
255     }
256     else{
257         tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
258         tmp.y = (-c1 - a1 * tmp.x) / b1;
259     }
260     return tmp;
261 }
262 
263 
264 TPolygon Cut_polygon(TPoint p1, TPoint p2, TPolygon polygon)
265 {
266     TPolygon new_polygon;
267     TPoint interp;
268     TLine l1, l2;
269     int i, j;
270     double t1, t2;
271     new_polygon.n = 0;
272     for(i = 0;i <= polygon.n - 1;i++){
273         t1 = multi(p2, polygon.p[i], p1);
274         t2 = multi(p2, polygon.p[i + 1], p1);
275         if(fabs(t1) < eps || fabs(t2) < eps){
276             if(fabs(t1) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i];
277             if(fabs(t2) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
278         }
279         else if(t1 < 0 && t2 < 0){
280             new_polygon.p[new_polygon.n++] = polygon.p[i];
281             new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
282         }
283         else if(t1 * t2  < 0){
284             l1 = lineFromSegment(p1, p2);
285             l2 = lineFromSegment(polygon.p[i], polygon.p[i + 1]);
286             interp = LineInter(l1, l2);
287             if(t1 < 0) {
288                 new_polygon.p[new_polygon.n++] = polygon.p[i];
289                 new_polygon.p[new_polygon.n++] = interp;
290             }
291             else {
292                 new_polygon.p[new_polygon.n++] = interp;
293                 new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
294             }
295         }
296     }
297     polygon.n = 0;
298     if(new_polygon.n == 0) return polygon;
299     polygon.p[polygon.n++] = new_polygon.p[0];
300     for(i = 1;i < new_polygon.n;i++){
301         if(!same(new_polygon.p[i], new_polygon.p[i - 1])){
302             polygon.p[polygon.n++] = new_polygon.p[i];
303         }
304     }
305     if(polygon.n != 1 && same(polygon.p[polygon.n - 1], polygon.p[0])) polygon.n--;
306     polygon.p[polygon.n] = polygon.p[0];
307     return polygon;
308 }
309 
310 
311 void ChangeClockwise(TPolygon &polygon)
312 {
```

```
    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
¾□O(n^3)³¬□£¬□□□¸□□□□□□□Ⓕ±□□i,j□□□□□□□□□²□□□□£¬¿□□□□□¹□
´□ţµ□□□□
□□□□□□□□□ O(n3) µģ¬³¬□¡£□□□□□□□□ö¾□ÿ¸□Ⓕ
□°□□□□□□□□µ□□□□□l□□□□□□□¬□□□□□□□□□□□¾□ö½¾¹□
¸õ□□□□□□□□□□¶□¼¸¸□Ⓕ□□□□□□□¾□□□□□£¸´
□□□□□□ O(n2logn) µģ£°□□□□□□□ 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

```
/*
}□□□§□□¶¬
```

```
3  □□□□□□□□└□»□□¬£□
4  ⅗»µ□□□□□□□□□□□□□□□□□□±□□□□□□□□
5  □□□□□□□□□ · □□□
6  */
7  #include <stdio.h>
8  #include <math.h>
9
10
11 #define eps 1e-8
12
13
14 struct TPoint
15 {
16     double x, y;
17 };
18 struct TLine
19 {
20     double a, b, c;
21 };
22
23
24 int same(TPoint p1, TPoint p2)
25 {
26     if(fabs(p1.x - p2.x) > eps) return 0;
27     if(fabs(p1.y - p2.y) > eps) return 0;
28     return 1;
29 }
30
31
32 double min(double x, double y)
33 {
34     if(x < y) return x;
35     else return y;
36 }
37
38
39 double max(double x, double y)
40 {
41     if(x > y) return x;
42     else return y;
43 }
44
45
46 double multi(TPoint p1, TPoint p2, TPoint p0)
47 {
48     return (p1.x - p0.x) * (p2.y - p0.y)
49          - (p2.x - p0.x) * (p1.y - p0.y);
50 }
51
52
53 bool isIntersected(TPoint s1, TPoint e1, TPoint s2, TPoint e2)
54 {
55     if(
56     (max(s1.x, e1.x) >= min(s2.x, e2.x)) &&
57     (max(s2.x, e2.x) >= min(s1.x, e1.x)) &&
58     (max(s1.y, e1.y) >= min(s2.y, e2.y)) &&
59     (max(s2.y, e2.y) >= min(s1.y, e1.y)) &&
60     (multi(s2, e1, s1) * multi(e1, e2, s1) >= 0) &&
61     (multi(s1, e2, s2) * multi(e2, e1, s2) >= 0)
```

```
62      )  return true;
63
64      return false;
65  }
66
67
68  TLine lineFromSegment(TPoint p1, TPoint p2)
69  {
70      TLine tmp;
71      tmp.a = p2.y - p1.y;
72      tmp.b = p1.x - p2.x;
73      tmp.c = p2.x * p1.y - p1.x * p2.y;
74      return tmp;
75  }
76
77
78  TPoint LineInter(TLine l1, TLine l2)
79  {
80      TPoint tmp;
81      double a1 = l1.a;
82      double b1 = l1.b;
83      double c1 = l1.c;
84      double a2 = l2.a;
85      double b2 = l2.b;
86      double c2 = l2.c;
87      if(fabs(b1) < eps){
88          tmp.x = -c1 / a1;
89          tmp.y = (-c2 - a2 * tmp.x) / b2;
90      }
91      else{
92          tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
93          tmp.y = (-c1 - a1 * tmp.x) / b1;
94      }
95      return tmp;
96  }
97
98
99  double triangleArea(TPoint p1, TPoint p2, TPoint p3)
100 {
101     TPoint p4, p5;
102     p4.x = p2.x - p1.x;
103     p4.y = p2.y - p1.y;
104     p5.x = p3.x - p1.x;
105     p5.y = p3.y - p1.y;
106     return fabs(p5.x * p4.y - p5.y * p4.x) / 2;
107 }
108
109
110 double find_x(double y, TLine line)
111 {
112     return (-line.c - line.b * y) / line.a;
113 }
114
115
116 double find_y(double x, TLine line)
117 {
118     if(fabs(line.b) < eps)
119     {
120         return -1e250;
```

```
121     }
122     else
123     {
124         return (-line.c - line.a  * x) / line.b;
125     }
126 }
127
128
129 int main()
130 {
131     //freopen("in.in", "r", stdin);
132     //freopen("out.out", "w", stdout);
133     int test;
134     double miny, y;
135     TLine l1, l2;
136     TPoint p1, p2, p3, p4, inter;
137     TPoint tp1, tp2;
138     scanf("%d", &test);
139     while(test--)
140     {
141         scanf("%lf%lf%lf%lf%lf%lf%lf%lf", &p1.x, &p1.y,
142         &p2.x, &p2.y, &p3.x, &p3.y, &p4.x, &p4.y);
143         if(same(p1, p2) || same(p3, p4)
144            || !isIntersected(p1, p2, p3, p4)
145            || fabs(p1.y - p2.y) < eps //□□□□□x□□
146            || fabs(p3.y - p4.y) < eps
147           )
148         {
149             printf("0.00\n");
150             continue;
151         }
152         l1 = lineFromSegment(p1, p2);
153         l2 = lineFromSegment(p3, p4);
154         inter = LineInter(l1, l2);
155         if(p1.y > p2.y) tp1 = p1;
156         else tp1 = p2;
157         if(p3.y > p4.y) tp2 = p3;
158         else tp2 = p4;
159         if(tp1.y < tp2.y)
160         {
161             if(tp1.x >= min(p4.x, p3.x) && tp1.x <= max(p4.x, p3.x))
162             {
163                 y = find_y(tp1.x, l2);
164                 if(y >= tp1.y)
165                 {
166                     printf("0.00\n");
167                     continue;
168                 }
169             }
170             miny = tp1.y;
171         }
172         else
173         {
174             if(tp2.x >= min(p1.x, p2.x) && tp2.x <= max(p1.x, p2.x))
175             {
176                 y = find_y(tp2.x, l1);
177                 if(y >= tp2.y)
178                 {
179                     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□□□□□□¸□□
//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»□□□ª½□□□¸□□□
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
```

```
39      tmp.c = p2.x * p1.y - p1.x * p2.y;
40      return tmp;
41  }
42
43
44  void swap(int &a, int &b)
45  {
46      int t;
47      t = a;
48      a = b;
49      b = t;
50  }
51
52
53  int Count(TPoint p1, TPoint p2)
54  {
55      int i, sum = 0, y;
56      TLine l1 =  lineFromSegment(p1, p2);
57      if(l1.b == 0) return abs(p2.y - p1.y) + 1;
58      if(p1.x > p2.x) swap(p1.x, p2.x); //□□□□□û□□»»»WA}´□
59      for(i = p1.x;i <= p2.x;i++){
60          y = -l1.c - l1.a * i;
61          if(y % l1.b == 0) sum++;
62      }
63      return sum;
64  }
65
66
67  int main()
68  {
69      //freopen("in.in", "r", stdin);
70      //freopen("OUT.out", "w", stdout);
71      TPoint p1, p2, p3;
72      while(scanf("%d%d%d%d%d%d", &p1.x, &p1.y, &p2.x, &p2.y, &p3.x, &p3.y) != EOF){
73          if(p1.x == 0 && p1.y == 0 && p2.x == 0 && p2.y == 0 && p3.x == 0 && p3.y == 0)
        break;
74          int  A = triangleArea(p1, p2, p3);//A□□□□µ□}±¶
75          int b = 0;
76          int i;
77          b = Count(p1, p2) + Count(p1, p3) + Count(p3, p2) - 3;//3¸□□□□□□□´□
78          //i = A / 2- b / 2 + 1;
79          i = (A - b) / 2 + 1;
80          printf("%d\n", i);
81      }
82      return 0;
83  }
```

1.11 N 点中三个点组成三角形面积最大

```
1
2   //Rotating Calipers algorithm
3
4
5
6
7   #include <stdio.h>
8   #include <stdlib.h>
9   #include <math.h>
```

```
10
11
12  #define MaxNode 50005
13
14
15  int stack[MaxNode];
16  int top;
17  double max;
18
19
20  typedef struct TPoint
21  {
22      int x;
23      int y;
24  }TPoint;
25  TPoint point[MaxNode];
26
27
28  void swap(TPoint point[], int i, int j)
29  {
30      TPoint tmp;
31      tmp = point[i];
32      point[i] = point[j];
33      point[j] = tmp;
34  }
35
36
37  double multi(TPoint p1, TPoint p2, TPoint p0)
38  {
39      return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
40  }
41
42
43  double distance(TPoint p1, TPoint p2)
44  {
45      return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
46  }
47
48
49  int cmp(const void *a, const void *b)
50  {
51      TPoint *c = (TPoint *)a;
52      TPoint *d = (TPoint *)b;
53      double k = multi(*c, *d, point[0]);
54      if(k< 0) return 1;
55      else if(k == 0 && distance(*c, point[0]) >= distance(*d, point[0]))
56              return 1;
57      else return -1;
58  }
59
60
61  void grahamScan(int n)
62  {
63      //Graham□□□□□□°□
64      int i, u;
65
66      //½«□□□□□µ□□□□□µ½p[0]µ□□□□
67      u = 0;
68      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.□□□□μl»μ□
2.μ□□□□□□Ķ□□□□
3.μ🄵□□l□□□□
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;

```

```
37
38  typedef struct TLine
39  {
40      //□□□□□□□□□□□□
41      double a, b, c;
42  }TLine;
43
44
45  void SloveLine(TLine &line, TPoint start, TPoint dir)
46  {
47      //¸□¾□□□□□□□µ□□□□½¸□□□□□□□¸½³□□
48      if(dir.x == 0){
49          line.a = 1;
50          line.b = 0;
51          line.c = start.x;
52      }
53      else {
54          double k = dir.y / dir.x;
55          line.a = k;
56          line.b = -1;
57          line.c = start.y - k * start.x;
58      }
59  }
60
61
62  TLine lineFromSegment(TPoint p1, TPoint p2)
63  {
64      //□□□□□□□□□□□,·µ»□□□□□½□□□¸□□□
65      TLine tmp;
66      tmp.a = p2.y - p1.y;
67      tmp.b = p1.x - p2.x;
68      tmp.c = p2.x * p1.y - p1.x * p2.y;
69      return tmp;
70  }
71
72
73  TPoint symmetricalPointofLine(TPoint p, TLine L)
74  {
75      //pµ□□□□□□□Lµ¸□□□
76      TPoint p2;
77      double d;
78      d = L.a * L.a + L.b * L.b;
79      p2.x = (L.b * L.b * p.x - L.a * L.a * p.x -
80              2 * L.a * L.b * p.y - 2 * L.a * L.c) / d;
81      p2.y = (L.a * L.a * p.y - L.b * L.b * p.y -
82              2 * L.a * L.b * p.x - 2 * L.b * L.c) / d;
83      return p2;
84  }
85
86
87  double distanc(TPoint p1, TPoint p2)
88  {
89      //¾□□□□□□□□}¸□□¾□□²□□□
90     return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
91  }
92
93
94  bool samedir(TPoint dir, TPoint start, TPoint point)
95  {
```

```
96      //□□□½□□
97      TPoint tmp;
98      tmp.x = point.x - start.x;
99      tmp.y = point.y - start.y;
100     if(tmp.x != 0 && dir.x != 0){
101         if(tmp.x / dir.x > 0) return true;
102         else return false;
103     }
104     else if(tmp.y != 0 && dir.y != 0){
105         if(tmp.y / dir.y > 0) return true;
106         else return false;
107     }
108     return true;
109 }
110
111
112 bool Intersected(TPoint &point, TLine line, const TCircle circle[],
113                  TPoint start, TPoint dir, int which)
114 {
115     //□□□□□□□□)□□□□□½»µ□)½»µ□□□□□□-□point□□
116     double a = line.a, b = line.b, c = line.c;
117     double x0 = circle[which].center.x, y0 = circle[which].center.y;
118     double r =  circle[which].r;
119     //□□»µ🄵□□□
120     double x2front = b * b + a * a;
121     double x1front = -2 * x0 * b * b + 2 * a * b * y0 + 2 * a * c;
122     double front = x0 * x0 * b * b + y0 * y0 * b * b
123         + c * c + 2 * c * y0 * b - b * b * r * r;
124     double d = x1front * x1front - 4 * x2front * front;
125     TPoint p1, p2;
126     bool k1, k2;
127     if(fabs(d) < eps){
128         //x2front²»¿□□□□□□□
129        point.x = -x1front / x2front / 2;
130        point.y = (-c - a * point.x) / b;
131        //□□□½□□
132        if(samedir(dir, start, point)) return true;
133        else return false;
134     }
135     else if(d < 0) return false;
136     else {
137         p1.x = (-x1front + sqrt(d)) / 2 / x2front;
138         p1.y = (-c - a * p1.x) / b;
139         p2.x = (-x1front - sqrt(d)) / 2 / x2front;
140         p2.y = (-c - a * p2.x) / b;
141         k1 = samedir(dir, start, p1);
142         k2 = samedir(dir, start, p2);
143         if(k1 == false && k2 == false) return false;
144         if(k1 == true && k2 == true){
145             double dis1 = distanc(p1, start);
146             double dis2 = distanc(p2, start);
147             if(dis1 < dis2) point = p1;
148             else point = p2;
149             return true;
150         }
151         else if(k1 == true) point = p1;
152         else point = p2;
153         return true;
154     }
```

```
155 }
156
157
158 void Reflect(int &num, TCircle circle[], TPoint start, TPoint dir, int n)
159 {
160     //·´¸´·´□□
161     int i;
162     TLine line;
163     TPoint interpoint, newstart;
164     int u;
165     SloveLine(line, start, dir);
166     int tag = 0;
167     double mindis = INF;
168     for(i = 1;i <= n;i++){
169         if(i != up && Intersected(interpoint, line, circle, start, dir, i)){
170             double dis = distanc(start, interpoint);
171             if(dis < mindis){
172                 tag = 1;
173                 u = i;
174                 mindis = dis;
175                 newstart = interpoint;
176             }
177         }
178     }
179     if(tag == 0){
180         cout << "inf" << endl;
181         return ;
182     }
183     else {
184         if(num == 10){
185             cout << "..." << endl;
186            return ;
187         }
188         cout << u << " ";
189         num++;
190         //□µ¸½□□
191         TLine line1;
192         TPoint p;
193         line1 = lineFromSegment(newstart, circle[u].center);
194         if(fabs(line1.a * start.x + line1.b * start.y +line1.c) <= eps){
195             dir.x = -dir.x;
196             dir.y = -dir.y;
197         }
198         else {
199             p = symmetricalPointofLine(start, line1);//start¶Ô³□□□
200             dir.x = p.x - newstart.x;
201             dir.y = p.y - newstart.y;
202         }
203
204         start = newstart;
205         up = u;
206         Reflect(num, circle, start, dir, n);
207     }
208 }
209
210
211 int main()
212 {
213     //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;
```

```
36      hash[hashl].next = hash[i].next;
37      hash[i].next = hashl;
38      hashl++;
39      return 0;
40  }
41
42
43  int Hash2(int n)
44  {
45      int i = n % PRIME;
46      while(hash[i].next != -1){
47          if(hash[hash[i].next].cnt == n) return 1;
48          else if(hash[hash[i].next].cnt > n) return 0;
49          i = hash[i].next;
50      }
51      return 0;
52  }
53
54
55  int check(double ax, double ay, int &x, int &y)
56  {
57      int a0 = (int)ax;
58      int b0 = (int)ay;
59      int tag1 = 0, tag2 = 0;
60      if(fabs(a0 - ax) < eps){
61          tag1 = 1;
62          x = a0;
63      }
64      else if(fabs(a0 + 1 - ax) < eps){
65          tag1 = 1;
66          x = a0 + 1;
67      }
68      if(fabs(b0 - ay) < eps){
69          tag2 = 1;
70          y = b0;
71      }
72      else if(fabs(b0 + 1 - ay) < eps){
73          y = b0 + 1;
74          tag2 = 1;
75      }
76      if(tag1 == 1 && tag2 == 1) return 1;
77      else return 0;
78  }
79
80
81  int squares(point p1, point p2, point &p3, point &p4)
82  {
83      double a = (double)p2.x - p1.x;
84      double b = (double)p2.y - p1.y;
85      double midx = ((double)p1.x + p2.x) / 2;
86      double midy = ((double)p1.y + p2.y) / 2;
87      double tmp = a * a + b * b;
88      double x1 = sqrt(b * b) / 2;
89      double y1;
90      if(fabs(b) < eps) y1 = sqrt(a * a + b * b) / 2;
91      else y1 = -a * x1 / b;
92      x1 += midx;
93      y1 += midy;
94      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¸□F□□□¸□FRμ□□□¸²¸□¬□□□□?¸□□□□□F
±□□μ□□□□└¡£
¶□ÿ¸□□□□R□°Fº»□£¬¶□N¸□□}}□□£□□□²½O(N^2)¡£□□□□□¯«□□□²¸□□□□□□□╚¸¡¡£
¶□ÿ¸□□£¬□□□□□□□ÿ¶□¡□□□´□□□¡£¼□□□A□□□B□□□╣£A□□[PI/3, PI/2]μ□□□¼FB¸²¸□(PI□□□□□□□)¡£□□ô¶
    □□□A□£¬□□□□□□□PI/3´¦□□□¸□+1±□□¬□□PI/2´¦□□□¸□-1±□□£
¶□□□[PI*5/3, PI*7/3]□□□□°□□0μ□□□□¼□□0□□μF²□□}¶□´¿□£
½«□¸□□□□□□□□□□□□□□□□□□□¿ª□□□衣³□□ans = 0£¬□□+1±□□□ans++£¬□□-1±□□ans--¡£□□□□³□□□ansμ□□□□▒
    □□□□□□□»¸²¸□□□□╚¡¡£□□□□□□□μ□ansμ□□□□▒□□□□£
□□□□□O(N^2 * logN)
*/#include <stdio.h>
#include <math.h>


#define eps 1e-6


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

```
20
21
22  double dis(point p1, point p2)
23  {
24      point p3;
25      p3.x = p2.x - p1.x;
26      p3.y = p2.y - p1.y;
27      return p3.x * p3.x + p3.y * p3.y;
28  }
29
30
31  point find_centre(point p1, point p2)
32  {
33      point p3, mid, centre;
34      double b, c, ang;
35      p3.x = p2.x - p1.x;
36      p3.y = p2.y - p1.y;
37      mid.x = (p1.x + p2.x) / 2;
38      mid.y = (p1.y + p2.y) / 2;
39      b = dis(p1, mid);
40      c = sqrt(1 - b);
41      if(fabs(p3.y) < eps)//´¹□□□□%□90¶□
42      {
43          centre.x = mid.x;
44          centre.y = mid.y + c;
45      }
46      else
47      {
48          ang = atan(-p3.x / p3.y);
49          centre.x = mid.x + c * cos(ang);
50          centre.y = mid.y + c * sin(ang);
51      }
52      return centre;
53  }
54
55
56  int main()
57  {
58      int n, ans, tmpans, i, j, k;
59      point p[305], centre;
60      double tmp;
61      while(scanf("%d", &n) && n)
62      {
63          for(i = 0;i < n;i++)
64              scanf("%lf%lf", &p[i].x, &p[i].y);
65          ans = 1;
66          for(i = 0;i < n;i++)
67              for(j = i + 1;j < n;j++)
68              {
69                  if(dis(p[i], p[j]) > 4) continue;
70                  tmpans = 0;
71                  centre = find_centre(p[i], p[j]);
72                  for(k = 0;k < n;k++)
73                  {
74                      //if(tmpans + n - k <= ans) break;
75                      tmp = dis(centre, p[k]);
76                      //if(tmp < 1.0 || fabs(tmp - 1.0) < eps) tmpans++;
77                      if(tmp <= 1.000001) tmpans++;
78                  }
```

```
                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]);
```

```
50          qsort(rate, rt, sizeof(rate[0]), cmp);
51          int ans = 1;
52          for(int i = 1;i < rt;i++)
53              if(rate[i] > rate[i - 1]) ans++;
54          //□□□□□□□□□fabs(rate[i] - rate[i - 1]) > eps Wrong Answer
55          printf("%d\n", ans);
56      }
57      return 0;
58  }
```

1.16 求凸多边形直径

```
1   #include<stdio.h>
2   #include<math.h>
3
4
5   #define eps 1e-6
6   #define MaX 6000
7
8
9   /*-----------¶□□□□□------------*/
10  struct POLYGON{
11     int n;                                    //¶□□□□¥μ□□□
12     double x[MaX],y[MaX];        //¶¥μ□□□±□
13  }poly;
14
15
16  int zd[100000][2],znum;      //□Ŷ□¦¯°□□□Ŷ□□□□-
17
18
19
20
21  /*------------¸¨□□°¯□□-----------*/
22  double dist(int a,int b,int c)
23  {
24     double vx1,vx2,vy1,vy2;
25     vx1=poly.x[b]-poly.x[a]; vy1=poly.y[b]-poly.y[a];
26     vx2=poly.x[c]-poly.x[a]; vy2=poly.y[c]-poly.y[a];
27     return fabs(vx1*vy2 - vy1*vx2);
28  }
29
30
31
32
33  /*-------------□□□¶□□□□Ø¶μÍ¯□□-------------*/
34  double DIAMETER()
35  {
36     znum=0;
37     int i,j,k=1;
38     double m,tmp;
39     while(dist(poly.n-1,0,k+1) > dist(poly.n-1,0,k)+eps)
40        k++;
41     i=0; j=k;
42     while(i<=k && j<poly.n)
43     {
44        zd[znum][0]=i; zd[znum++][1]=j;
45        while(dist(i,i+1,j+1)>dist(i,i+1,j)-eps && j<poly.n-1)
46        {
```

```
47          zd[znum][0]=i; zd[znum++][1]=j;
48          j++;
49       }
50       i++;
51    }
52    m=-1;
53    for(i=0;i<znum;i++)
54    {
55       tmp =(poly.x[zd[i][0]]-poly.x[zd[i][1]]) * (poly.x[zd[i][0]]-poly.x[zd[i][1]]);
56       tmp+=(poly.y[zd[i][0]]-poly.y[zd[i][1]]) * (poly.y[zd[i][0]]-poly.y[zd[i][1]]);
57       if(m<tmp) m=tmp;
58    }
59    return sqrt(m);
60 }
61
62
63 /*----------□□□□----------*/
64 int main()
65 {
66    int i;
67    while(scanf("%d",&poly.n)==1)
68    {
69       for(i=0;i<poly.n;i++)
70          scanf("%lf %lf",&poly.x[i],&poly.y[i]);
71       printf("%.3lf\n",DIAMETER());
72    }
73    return 0;
74 }
75 1.16.19 ¾□□□□□□²¢£¬□□²¢
76 ¼□¸½¼□5.11,5.12
77 1.16.20 pku2069 □□□□□□
78 ¼□¸½¼□5.13,5.14
79 //□□□±□□□□
80 #include<stdio.h>
81 #include<math.h>
82 #include<memory>
83 #include<stdlib.h>
84 using namespace std;
85 const double eps = 1e-10;
86 struct point_type { double x, y, z; };
87 int npoint, nouter ;
88 point_type point [1000], outer[4], res;
89 double radius, tmp ;
90
91
92 inline double dist(point_type p1 , point_type p2)
93 {
94 double dx=p1.x-p2.x, dy=p1.y-p2.y,dz=p1.z-p2.z ;
95 return ( dx*dx + dy*dy + dz*dz ) ;
96 }
97
98
99 inline double dot( point_type p1 , point_type p2 )
100 {
101 return p1.x*p2.x + p1.y*p2.y + p1.z*p2.z;
102 }
103
104
105 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++)
```

```
67      {
68          temp = Area2(pnt[1],pnt[i],pnt[i + 1]);
69          sum += temp;
70      }
71      if (sum < 0) reverse(pnt + 1,pnt + n + 1);
72      pnt[n + 1] = pnt[1];
73  }
74
75
76  inline bool inCircle(pnt_type &s)
77  {
78      return dist(center,s) <= R;
79  }
80
81
82  bool SameSide(pnt_type a,pnt_type b)
83  {
84      if (dist(a,center) > dist(b,center)) swap(a,b);
85      return dot(a,b,center) < eps;
86  }
87
88
89  double ShadomOnCircle(pnt_type a,pnt_type b)
90  {
91      double flag = Area2(center,a,b),res = 0;
92      if (fabs(flag) < eps) return 0;
93
94
95      bool ina = inCircle(a),inb = inCircle(b);
96      if (ina && inb)
97      {
98          res = fabs(Area2(center,a,b)) / 2;
99      }
100     else if (!ina && !inb)
101     {
102         if (SameSide(a,b))
103         {
104             double theta = acos(dot(center,a,b) / dist(center,a) / dist(center,b));
105             res = R * R * theta / 2;
106         }
107         else
108         {
109             double height = fabs(Area2(center,a,b)) / dist(a,b);
110             double theta = acos(dot(center,a,b) / dist(center,a) / dist(center,b));
111             if (height >= R)
112             {
113                 res = R * R * theta / 2;
114             }
115             else
116             {
117                 double _theta = 2 * acos(height / R);
118                 res = R * R * (theta - _theta) / 2 + R * R * sin(_theta) / 2;
119             }
120         }
121     }
122     else
123     {
124         if (!ina && inb) swap(a,b);
125         double height = fabs(Area2(center,a,b)) / dist(a,b);
```

```
126         double temp = dot(a,center,b);
127         double theta = acos(dot(center,a,b) / dist(center,a) / dist(center,b)),theta1,
    theta2;
128         if (fabs(temp) < eps)
129         {
130             double _theta = acos(height / R);
131             res += R * height / 2 * sin(_theta);
132             res += R * R / 2 * (theta - _theta);
133         }
134         else
135         {
136             theta1 = asin(height / R); theta2 = asin(height / dist(a,center));
137             if (temp > 0)
138             {
139                 res += dist(center,a) * R / 2 * sin(PI - theta1 - theta2);
140                 res += R * R / 2 * (theta + theta1 + theta2 - PI);
141             }
142             else
143             {
144                 res += dist(center,a) * R / 2 * sin(theta2 - theta1);
145                 res += R * R / 2 * (theta - theta2 + theta1);
146             }
147         }
148     }
149     if (flag < 0) return -res; else return res;
150 }
151
152
153 double Cover()
154 {
155     int i;
156     double res = 0;
157     for (i=1;i<=n;i++)
158         res += ShadomOnCircle(pnt[i],pnt[i + 1]);
159     return res;
160 }
161
162
163 int main()
164 {
165     double ans;
166     while (read_data())
167     {
168         init();
169         ans = Cover();
170         printf("%.2lf\n",ans);
171     }
172     return 0;
173 }
```

1.18 半平面交

```
1 //Nlgn
2 #include <stdio.h>
3 #include <string.h>
4 #include <math.h>
5 #include <algorithm>
6 using namespace std;
```

```
7  #define maxn 20005
8  #define eps 1e-10
9
10
11 struct point
12 {double x,y;};
13 struct line
14 {point s,e;double k;};
15
16
17 line L[maxn];
18 point S[maxn];
19
20
21 int N,Q[maxn];
22
23
24 double cross(point a,point b,point c) // c□□□□ab□□□µ»□<0
25 {return (c.y-a.y)*(b.x-a.x)-(b.y-a.y)*(c.x-a.x);}
26
27
28 bool operator < (line a,line b) // □□se□□□□¿□□□□□
29 {
30     if( fabs(a.k-b.k)<eps )
31         return cross(b.s,b.e,a.s)<0;
32     return a.k<b.k;
33 }
34
35
36 point intersection(point u1,point u2,point v1,point v2){
37     point ret=u1;
38     double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
39         /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
40     ret.x+=(u2.x-u1.x)*t;
41     ret.y+=(u2.y-u1.y)*t;
42     return ret;
43 }
44
45
46 double HalfInSec()
47 {
48     int i,j,k,l;
49     sort(L,L+N);  // ¼«½□[-pi,pi]□□□□
50
51     for(i=1,j=0;i<N;i++)  // □µ□□□¼«½□□□
52         if( fabs(L[i].k-L[j].k)>eps )
53         L[++j] = L[i];
54     N = j+1;
55
56
57     k = 0,l = 1;
58     Q[0] = 0,Q[1] = 1;
59     S[1] = intersection(L[0].s,L[0].e,L[1].s,L[1].e);
60     for(i=2;i<N;i++)
61     {
62         while( k<l && cross(L[i].s,L[i].e,S[l])>eps )
63             l--;
64         while( k<l && cross(L[i].s,L[i].e,S[k+1])>eps )
65             k++;
```

```
66          Q[++l] = i;
67          S[l] = intersection(L[Q[l-1]].s,L[Q[l-1]].e,L[i].s,L[i].e);
68      }
69
70
71      while( k<l && cross(L[Q[k]].s,L[Q[k]].e,S[l])>eps )
72          l--;
73      while( k<l && cross(L[Q[l]].s,L[Q[l]].e,S[k+1])>eps )
74          k++;
75      S[k] = intersection(L[Q[l]].s,L[Q[l]].e,L[Q[k]].s,L[Q[k]].e);
76      S[++l] = S[k];
77
78
79      double s = 0;
80      for(i=k;i<l;i++)
81          s += S[i].y*S[i+1].x-S[i+1].y*S[i].x;
82      return fabs(s/2);
83  }
84  int main()
85  {
86      int i,j,k,l;
87      scanf("%d",&N);
88      for(i=0;i<N;i++)
89          scanf("%lf%lf%lf%lf",&L[i].e.x,&L[i].e.y,&L[i].s.x,&L[i].s.y);
90      L[N].s.x = 0,L[N].s.y = 0;
91      L[N+1].s.x = 10000,L[N+1].s.y = 0;
92      L[N+2].s.x = 10000,L[N+2].s.y = 10000;
93      L[N+3].s.x = 0,L[N+3].s.y = 10000;
94      L[N].e = L[N+3].s;
95      L[N+1].e = L[N].s;
96      L[N+2].e = L[N+1].s;
97      L[N+3].e = L[N+2].s;
98      N += 4;
99      for(i=0;i<N;i++)
100         L[i].k = atan2(L[i].s.y-L[i].e.y,L[i].s.x-L[i].e.x);
101     printf("%.1lf\n",HalfInSec());
102 }
```