计算几何模板

ACM@HIT

Author: jerrybond

基础部分

1.几何公式

1.1 三角形

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

1.2 四边形

D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2

(以下对圆的内接四边形)

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

1.3 正 n 边形

R 为外接圆半径,r 为内切圆半径

1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

1.4 圆

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

1.5 棱柱

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

1.6 棱锥

1. 体积 V=Ah/3,A 为底面积,h 为高

(以下对正棱锥)

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

1.7 棱台

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

(以下为正棱台)

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

1.8 圆柱

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

1.9 圆锥

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

1.10 圆台

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

1.11 球

1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3

1.12 球台

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

1.13 球扇形

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

2.直线与线段

2.0 预备函数

```
//结构定义与宏定义
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#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;
};
```

//计算 cross product (P1-P0)x(P2-P0)

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

//计算 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);
}
```

//两点距离

```
double distance(point p1,point p2)
{
	return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double distance(double x1,double y1,double x2,double y2)
{
	return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
```

2.1 判三点是否共线

```
int dots_inline(point p1,point p2,point p3)
{
	return zero(xmult(p1,p2,p3));
}
```

2.2 判点是否在线段上

```
//判点是否在线段上,包括端点（下面为两种接口模式）
int dot_online_in(point p,line l)
{
	return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<eps;
}
int dot_online_in(point p,point l1,point l2)
{
	return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
//判点是否在线段上,不包括端点
int dot_online_ex(point p,line l)
{
	return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))
			&&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));
}
```

2.3 判断两点在线段的同一侧

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

2.4 判断两点是否在线段的异侧

```
//判两点在线段异侧,点在线段上返回 0
int opposite_side(point p1,point p2,line l)
{
	return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
}
int opposite_side(point p1,point p2,point l1,point l2)
{
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
```

2.5 求点关于直线的对称点

```
// 点关于直线的对称点 // by lyt
// 缺点：用了斜率
// 也可以利用"点到直线上的最近点"来做，避免使用斜率。
point symmetric_point(point p1, point l1, point l2)
{
    point ret;
    if (l1.x > l2.x - eps && l1.x < l2.x + eps)
    {
        ret.x = (2 * l1.x - p1.x);
        ret.y = p1.y;
    }
    else
    {
        double k = (l1.y - l2.y ) / (l1.x - l2.x);
        ret.x = (2*k*k*l1.x + 2*k*p1.y - 2*k*l1.y - k*k*p1.x + p1.x) / (1 + k*k);
        ret.y = p1.y - (ret.x - p1.x ) / k;
    }
    return ret;
}
```

2.7 判断两线段是否相交

2.7.1 常用版

```
//定义点
struct Point
{
    double x;
    double y;
};
typedef struct Point point;

//叉积
double multi(point p0, point p1, point p2)
{
    return ( p1.x - p0.x )*( p2.y - p0.y )-( p2.x - p0.x )*( p1.y - p0.y );
}
```

```
//相交返回 true,否则为 false, 接口为两线段的端点
bool isIntersected(point s1,point e1, point s2,point e2)
{
      return   (max(s1.x,e1.x) >= min(s2.x,e2.x))   &&
                  (max(s2.x,e2.x) >= min(s1.x,e1.x))   &&
                  (max(s1.y,e1.y) >= min(s2.y,e2.y))   &&
                  (max(s2.y,e2.y) >= min(s1.y,e1.y))   &&
                  (multi(s1,s2,e1)*multi(s1,e1,e2)>0) &&
                  (multi(s2,s1,e2)*multi(s2,e2,e1)>0);
}
```

2.7.2 不常用版

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

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

2.8 求两条直线的交点

```
//计算两直线交点,注意事先判断直线是否平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
```

```
point intersection(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;
}
```

2.9 点到直线的最近距离

```
point ptoline(point p,point l1,point l2)
{
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	return intersection(p,t,l1,l2);
}
```

2.10 点到线段的最近距离

```
point ptoseg(point p,point l1,point l2)
{
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
		return distance(p,l1)<distance(p,l2)?l1:l2;
	return intersection(p,t,l1,l2);
}
```

3.多边形

3.0 预备浮点函数

```
#include <stdlib.h>
#include<stdio.h>
#include<string.h>
#include <math.h>
#define MAXN 1000
```

//offset 为多变形坐标的最大绝对值

```
#define offset 10000
#define eps 1e-8

//浮点数判 0
#define zero(x) (((x)>0?(x):-(x))<eps)

//浮点数判断符
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))

//定义点
struct point
{
    double x,y;
}pt[MAXN ];

//定义线段
struct line
{
    point a,b;
};

//叉积
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);
}
```

3.1 判定是否是凸多边形

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

//判凸行，顶点按顺时针或逆时针给出,不允许相邻边共线,是凸多边形返回 1，否则返回 0
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];
}
```

3.2 判定点是否在多边形内

```
//判点在凸多边形内或多边形边上时返回 1，严格在凸多边形外返回 0
int inside_convex(point q,int n,point* p)
{
        int i,s[3]={1,1,1};
        for (i=0;i<n&&s[1]|s[2];i++)
                s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
        return s[1]|s[2];
}

//判点严格在凸多边形内返回 1,在边上或者严格在外返回 0
int inside_convex_v2(point q,int n,point* p)
{
        int i,s[3]={1,1,1};
        for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
                s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
        return s[0]&&s[1]|s[2];
}

//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge 表示点在多边形边上时的返回值, offset 为多边形坐标上限,严格在内返回 1，严格在外返回 0
int inside_polygon(point q,int n,point* p,int on_edge=2)
{
        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;
}
```

3.3 判定一条线段是否在一个任意多边形内

```
//预备函数
inline int opposite_side(point p1,point p2,point l1,point l2)
{
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}
inline int dot_online_in(point p,point l1,point l2)
{
    return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}

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

4.三角形

4.0 预备函数

```
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include<stdio.h>
//定义点
struct point
{
        double x,y;
};
typedef struct point point;

//定义直线
struct line
{
        point a,b;
};
typedef struct line line;
//两点距离
double distance(point p1,point p2)
{
        return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
//两直线求交点
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;
}
```

4.1 求三角形的外心

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

4.2 求三角形内心

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

4.3 求三角形垂心

```
point perpencenter(point a,point b,point c)
{
    line u,v;
    u.a=c;
    u.b.x=u.a.x-a.y+b.y;
    u.b.y=u.a.y+a.x-b.x;
    v.a=b;
    v.b.x=v.a.x-a.y+c.y;
    v.b.y=v.a.y+a.x-c.x;
    return intersection(u,v);
}
```

5.圆

5.0 预备函数

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define eps 1e-8
struct point
{
    double x,y;
};
typedef struct point point;
double xmult(point p1,point p2,point p0)
{
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double distance(point p1,point p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
//点到直线的距离
double disptoline(point p,point l1,point l2)
{
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
//求两直线交点
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;
}
```

5.1 判定直线是否与圆相交

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

5.2 判定线段与圆相交

```
int intersect_seg_circle(point c,double r, point l1,point l2)
{
	double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
	point t=c;
	if (t1<eps||t2<eps)
		return t1>-eps||t2>-eps;
	t.x+=l1.y-l2.y;
	t.y+=l2.x-l1.x;
	return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
}
```

5.3 判圆和圆相交

```
int intersect_circle_circle(point c1,double r1,point c2,double r2)
{
	return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
}
```

5.4 计算圆上到点 p 最近点

```
//当 p 为圆心时，返回圆心本身
point dot_to_circle(point c,double r,point p)
{
	point u,v;
	if (distance(p,c)<eps)
```

```
		return p;
	u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
	u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
	v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
	v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
	return distance(u,p)<distance(v,p)?u:v;
}
```

5.5 计算直线与圆的交点

```
//计算直线与圆的交点,保证直线与圆有交点
//计算线段与圆的交点可用这个函数后判点是否在线段上
void intersection_line_circle(point c,double r,point l1,point l2,point& p1,point& p2)
{
	point p=c;
	double t;
	p.x+=l1.y-l2.y;
	p.y+=l2.x-l1.x;
	p=intersection(p,c,l1,l2);
	t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(l1,l2);
	p1.x=p.x+(l2.x-l1.x)*t;
	p1.y=p.y+(l2.y-l1.y)*t;
	p2.x=p.x-(l2.x-l1.x)*t;
	p2.y=p.y-(l2.y-l1.y)*t;
}
```

5.6 计算两个圆的交点

```
//计算圆与圆的交点,保证圆与圆有交点,圆心不重合
void intersection_circle_circle(point c1,double r1,point c2,double r2,point& p1,point& p2)
{
	point u,v;
	double t;
	t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(c1,c2))/2;
	u.x=c1.x+(c2.x-c1.x)*t;
	u.y=c1.y+(c2.y-c1.y)*t;
	v.x=u.x+c1.y-c2.y;
	v.y=u.y-c1.x+c2.x;
	intersection_line_circle(c1,r1,u,v,p1,p2);
}
```

6.球面

6.0 给出地球经度纬度，计算圆心角

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

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

6.1 已知经纬度，计算地球上两点直线距离

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

6.2 已知经纬度，计算地球上两点球面距离

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

7.三维几何的若干模板

7.0 预备函数

```
//三维几何函数库
#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;};

//计算 cross product U x V
point3 xmult(point3 u,point3 v){
    point3 ret;
    ret.x=u.y*v.z-v.y*u.z;
    ret.y=u.z*v.x-u.x*v.z;
    ret.z=u.x*v.y-u.y*v.x;
    return ret;
}

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

//矢量差  U - V
point3 subt(point3 u,point3 v){
    point3 ret;
    ret.x=u.x-v.x;
    ret.y=u.y-v.y;
    ret.z=u.z-v.z;
    return ret;
}

//取平面法向量
point3 pvec(plane3 s){
    return xmult(subt(s.a,s.b),subt(s.b,s.c));
}
point3 pvec(point3 s1,point3 s2,point3 s3){
    return xmult(subt(s1,s2),subt(s2,s3));
}
```

```
//两点距离,单参数取向量大小
double distance(point3 p1,point3 p2){
	return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z));
}

//向量大小
double vlen(point3 p){
	return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}
```

7.1 判定三点是否共线

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

7.2 判定四点是否共面

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

7.1 判定点是否在线段上

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

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

7.2 判断点是否在空间三角形上

```
//判点是否在空间三角形上,包括边界,三点共线无意义
int dot_inplane_in(point3 p,plane3 s){
	return zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-vlen(xmult(subt(p,s.a),subt(p,s.b)))-
		vlen(xmult(subt(p,s.b),subt(p,s.c)))-vlen(xmult(subt(p,s.c),subt(p,s.a))));
}
int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
	return zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-vlen(xmult(subt(p,s1),subt(p,s2)))-
		vlen(xmult(subt(p,s2),subt(p,s3)))-vlen(xmult(subt(p,s3),subt(p,s1))));
}


//判点是否在空间三角形上,不包括边界,三点共线无意义
int dot_inplane_ex(point3 p,plane3 s){
	return dot_inplane_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>eps&&
		vlen(xmult(subt(p,s.b),subt(p,s.c)))>eps&&vlen(xmult(subt(p,s.c),subt(p,s.a)))>eps;
}
int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
	return dot_inplane_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>eps&&
		vlen(xmult(subt(p,s2),subt(p,s3)))>eps&&vlen(xmult(subt(p,s3),subt(p,s1)))>eps;
}
```

7.3 判断两点是否在线段同侧

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

7.4 判断两点是否在线段异侧

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

7.5 判断两点是否在平面同侧

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

7.6 判断两点是否在平面异侧

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

7.7 判断两空间直线是否平行

```
//判两直线平行
int parallel(line3 u,line3 v){
	return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<eps;
}
int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
	return vlen(xmult(subt(u1,u2),subt(v1,v2)))<eps;
}
```

7.8 判断两平面是否平行

```
//判两平面平行
int parallel(plane3 u,plane3 v){
    return vlen(xmult(pvec(u),pvec(v)))<eps;
}
int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;
}
```

7.9 判断直线是否与平面平行

```
//判直线与平面平行
int parallel(line3 l,plane3 s){
    return zero(dmult(subt(l.a,l.b),pvec(s)));
}
int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));
}
```

7.10 判断两直线是否垂直

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

7.11 判断两平面是否垂直

```
//判两平面垂直
int perpendicular(plane3 u,plane3 v){
    return zero(dmult(pvec(u),pvec(v)));
}
int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));
}
```

7.12 判断两条空间线段是否相交

//判两线段相交,包括端点和部分重合

```
int intersect_in(line3 u,line3 v){
	if (!dots_onplane(u.a,u.b,v.a,v.b))
		return 0;
	if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
		return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
	return dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2){
	if (!dots_onplane(u1,u2,v1,v2))
		return 0;
	if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
		return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
	return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u2);
}
```

//判两线段相交,不包括端点和部分重合

```
int intersect_ex(line3 u,line3 v){
	return dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
	return
dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}
```

7.13 判断线段是否与空间三角形相交

//判线段与空间三角形相交,包括交于边界和(部分)包含

```
int intersect_in(line3 l,plane3 s){
	return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
		!same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
	return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
		!same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
}
```

//判线段与空间三角形相交,不包括交于边界和(部分)包含

```
int intersect_ex(line3 l,plane3 s){
	return opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&
		opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
	return opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&
		opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
}
```

7.14 计算两条直线的交点

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

7.15 计算直线与平面的交点

```
//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
//线段和空间三角形交点请另外判断
point3 intersection(line3 l,plane3 s){
	point3 ret=pvec(s);
	double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-l.a.z))/
		(ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
	ret.x=l.a.x+(l.b.x-l.a.x)*t;
	ret.y=l.a.y+(l.b.y-l.a.y)*t;
	ret.z=l.a.z+(l.b.z-l.a.z)*t;
	return ret;
}
point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    point3 ret=pvec(s1,s2,s3);
    double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
        (ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
    ret.x=l1.x+(l2.x-l1.x)*t;
    ret.y=l1.y+(l2.y-l1.y)*t;
    ret.z=l1.z+(l2.z-l1.z)*t;
    return ret;
}
```

7.16 计算两平面的交线

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

7.17 点到直线的距离

```
//点到直线距离
double ptoline(point3 p,line3 l){
    return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
}
double ptoline(point3 p,point3 l1,point3 l2){
    return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
}
```

7.18 计算点到平面的距离

```
//点到平面距离
```

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

7.19 计算直线到直线的距离

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

7.20 空间两直线夹角的 cos 值

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

7.21 两平面夹角的 cos 值

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

7.22 直线与平面夹角 sin 值

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

1.最远曼哈顿距离

```
#include <stdio.h>
#define INF 9999999999999.0
struct Point
{
	double x[5];
}pt[100005];
double dis[32][100005], coe[5], minx[32], maxx[32];
//去掉绝对值后有 2^D 种可能
void GetD(int N, int D)
{
	int s, i, j, tot=(1<<D);
	for (s=0;s<tot;s++)
	{
		for (i=0;i<D;i++)
			if (s&(1<<i))
				coe[i]=-1.0;
			else coe[i]=1.0;
		for (i=0;i<N;i++)
		{
			dis[s][i]=0.0;
			for (j=0;j<D;j++)
				dis[s][i]=dis[s][i]+coe[j]*pt[i].x[j];
		}
	}
}
//取每种可能中的最大差距
void Solve(int N, int D)
{
	int s, i, tot=(1<<D);
	double tmp, ans;
```

```
        for (s=0;s<tot;s++)
        {
                minx[s]=INF;
                maxx[s]=-INF;
                for (i=0; i<N; i++)
                {
                        if (minx[s]>dis[s][i]) minx[s]=dis[s][i];
                        if (maxx[s]<dis[s][i]) maxx[s]=dis[s][i];
                }
        }
        ans=0.0;
        for (s=0; s<tot; s++)
        {
                tmp=maxx[s]-minx[s];
                if (tmp>ans) ans=tmp;
        }
        printf("%.2lf\n", ans);
}
int main (void)
{
        int n, i;
        while (scanf("%d",&n)==1)
        {
                for (i=0;i<n;i++)
                    scanf("%lf%lf%lf%lf%lf",&pt[i].x[0],&pt[i].x[1],&pt[i].x[2],&pt[i].x[3],&pt[i].x[4]);
                GetD(n, 5);
                Solve(n, 5);
        }
        return 0;
}
```

2.最近点对

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define Max(x,y) (x)>(y)?(x):(y)
struct Q
{
        double x, y;
}q[100001], sl[10], sr[10];

int cntl, cntr, lm, rm;
```

```
double ans;

int cmp(const void*p1, const void*p2)
{
        struct Q*a1=(struct Q*)p1;
        struct Q*a2=(struct Q*)p2;
        if (a1->x<a2->x)return -1;
        else if (a1->x==a2->x)return 0;
        else return 1;
}

double CalDis(double x1, double y1, double x2, double y2)
{
        return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}

void MinDis(int l, int r)
{
        if (l==r) return;
        double dis;
        if (l+1==r)
        {
                dis=CalDis(q[l].x,q[l].y,q[r].x,q[r].y);
                if (ans>dis) ans=dis;
                return;
        }
        int mid=(l+r)>>1, i, j;
        MinDis(l,mid);
        MinDis(mid+1,r);

        lm=mid+1-5;
        if (lm<l) lm=l;
        rm=mid+5;
        if (rm>r) rm=r;

        cntl=cntr=0;
        for (i=mid;i>=lm;i--)
        {
                if (q[mid+1].x-q[i].x>=ans)break;
                sl[++cntl]=q[i];
        }
        for (i=mid+1;i<=rm;i++)
        {
                if (q[i].x-q[mid].x>=ans)break;
        sr[++cntr]=q[i];
    }

    for (i=1;i<=cntl;i++)
        for (j=1;j<=cntr;j++)
        {
            dis=CalDis(sl[i].x,sl[i].y,sr[j].x,sr[j].y);
            if (dis<ans) ans=dis;
        }
}

int main (void)
{
    int n, i;
    while (scanf("%d",&n)==1&&n)
    {
        for (i=1;i<=n;i++)
            scanf("%lf %lf", &q[i].x,&q[i].y);
        qsort(q+1,n,sizeof(struct Q),cmp);
        ans=CalDis(q[1].x,q[1].y,q[2].x,q[2].y);
        MinDis(1,n);
        printf("%.2lf\n",ans/2.0);
    }
    return 0;
}
```

3.最近点对

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define Max(x,y) (x)>(y)?(x):(y)
struct Q
{
    double x, y;
}q[100001], sl[10], sr[10];

int cntl, cntr, lm, rm;
double ans;

int cmp(const void*p1, const void*p2)
{
    struct Q*a1=(struct Q*)p1;
    struct Q*a2=(struct Q*)p2;
    if (a1->x<a2->x)return -1;
    else if (a1->x==a2->x)return 0;
    else return 1;
}

double CalDis(double x1, double y1, double x2, double y2)
{
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}

void MinDis(int l, int r)
{
    if (l==r) return;
    double dis;
    if (l+1==r)
    {
        dis=CalDis(q[l].x,q[l].y,q[r].x,q[r].y);
        if (ans>dis) ans=dis;
        return;
    }
    int mid=(l+r)>>1, i, j;
    MinDis(l,mid);
    MinDis(mid+1,r);

    lm=mid+1-5;
    if (lm<l) lm=l;
    rm=mid+5;
    if (rm>r) rm=r;

    cntl=cntr=0;
    for (i=mid;i>=lm;i--)
    {
        if (q[mid+1].x-q[i].x>=ans)break;
        sl[++cntl]=q[i];
    }
    for (i=mid+1;i<=rm;i++)
    {
        if (q[i].x-q[mid].x>=ans)break;
        sr[++cntr]=q[i];
    }

    for (i=1;i<=cntl;i++)
        for (j=1;j<=cntr;j++)
        {
            dis=CalDis(sl[i].x,sl[i].y,sr[j].x,sr[j].y);
            if (dis<ans) ans=dis;
        }
}

int main (void)
{
    int n, i;
    while (scanf("%d",&n)==1&&n)
    {
        for (i=1;i<=n;i++)
            scanf("%lf %lf", &q[i].x,&q[i].y);
        qsort(q+1,n,sizeof(struct Q),cmp);
        ans=CalDis(q[1].x,q[1].y,q[2].x,q[2].y);
        MinDis(1,n);
        printf("%.2lf\n",ans/2.0);
    }
    return 0;
}
```

4.最小包围圆

```
#include<stdio.h>
#include<string.h>
#include<math.h>
struct Point
{
    double x;
    double y;
}pt[1005];
struct Traingle
{
    struct Point p[3];
};
struct Circle
{
    struct Point center;
    double r;
}ans;
//计算两点距离
double Dis(struct Point p, struct Point q)
{
    double dx=p.x-q.x;
    double dy=p.y-q.y;
    return sqrt(dx*dx+dy*dy);
}
//计算三角形面积
double Area(struct Traingle ct)
{
    return
fabs((ct.p[1].x-ct.p[0].x)*(ct.p[2].y-ct.p[0].y)-(ct.p[2].x-ct.p[0].x)*(ct.p[1].y-ct.p[0].y))/2.0;
}
//求三角形的外接圆，返回圆心和半径(存在结构体"圆"中)
struct Circle CircumCircle(struct Traingle t)
{
    struct Circle tmp;
    double a, b, c, c1, c2;
    double xA, yA, xB, yB, xC, yC;
    a = Dis(t.p[0], t.p[1]);
    b = Dis(t.p[1], t.p[2]);
    c = Dis(t.p[2], t.p[0]);
    //根据 S = a * b * c / R / 4;求半径 R
    tmp.r = (a*b*c)/(Area(t)*4.0);
    xA = t.p[0].x;
    yA = t.p[0].y;
    xB = t.p[1].x;
    yB = t.p[1].y;
    xC = t.p[2].x;
    yC = t.p[2].y;
    c1 = (xA*xA+yA*yA - xB*xB-yB*yB) / 2;
    c2 = (xA*xA+yA*yA - xC*xC-yC*yC) / 2;
    tmp.center.x = (c1*(yA - yC)-c2*(yA - yB)) / ((xA - xB)*(yA - yC)-(xA - xC)*(yA - yB));
    tmp.center.y = (c1*(xA - xC)-c2*(xA - xB)) / ((yA - yB)*(xA - xC)-(yA - yC)*(xA - xB));
    return tmp;
}
//确定最小包围圆
struct Circle MinCircle(int num, struct Traingle ct)
{
    struct Circle ret;
    if (num==0) ret.r = 0.0;
    else if (num==1)
    {
        ret.center = ct.p[0];
        ret.r = 0.0;
    }
    else if (num==2)
    {
        ret.center.x = (ct.p[0].x+ct.p[1].x)/2.0;
        ret.center.y = (ct.p[0].y+ct.p[1].y)/2.0;
        ret.r = Dis(ct.p[0], ct.p[1])/2.0;
    }
    else if(num==3) ret = CircumCircle(ct);
    return ret;
}
//递归实现增量算法
void Dfs(int x, int num, struct Traingle ct)
{
    int i, j;
    struct Point tmp;
    ans = MinCircle(num, ct);
    if (num==3) return;
    for (i=1; i<=x; i++)
        if (Dis(pt[i], ans.center)>ans.r)
        {
            ct.p[num]=pt[i];
            Dfs(i-1, num+1, ct);
            tmp=pt[i];
            for (j=i;j>=2;j--)
                pt[j]=pt[j-1];
            pt[1]=tmp;
        }
}
void Solve(int n)
{
    struct Traingle ct;
    Dfs(n, 0, ct);
}
int main (void)
{
    int n, i;
    while (scanf("%d", &n)!=EOF && n)
    {
        for (i=1;i<=n;i++)
            scanf("%lf %lf", &pt[i].x, &pt[i].y);
        Solve(n);
        printf("%.2lf %.2lf %.2lf\n", ans.center.x, ans.center.y, ans.r);
    }
    return 0;
}
```

5.求两个圆的交点

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
const double eps = 1e-8;
const double PI = acos(-1.0);

struct Point
{
    double x;
    double y;
};
typedef struct Point point;

struct Line
{
    double s, t;
};
typedef struct Line Line;

struct Circle
{
    Point center;
    double r;
    Line line[505];
    int cnt;
    bool covered;

}circle[105];

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

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;
}

void intersection_line_circle(point c,double r,point l1,point l2,point& p1,point& p2)
{
    point p=c;
    double t;
    p.x+=l1.y-l2.y;
    p.y+=l2.x-l1.x;
    p=intersection(p,c,l1,l2);
    t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(l1,l2);
    p1.x=p.x+(l2.x-l1.x)*t;
    p1.y=p.y+(l2.y-l1.y)*t;
    p2.x=p.x-(l2.x-l1.x)*t;
    p2.y=p.y-(l2.y-l1.y)*t;
}

//计算圆与圆的交点,保证圆与圆有交点,圆心不重合
void intersection_circle_circle(point c1,double r1,point c2,double r2,point& p1,point& p2)
{
    point u,v;
    double t;
    t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(c1,c2))/2;
    u.x=c1.x+(c2.x-c1.x)*t;
    u.y=c1.y+(c2.y-c1.y)*t;
    v.x=u.x+c1.y-c2.y;
    v.y=u.y-c1.x+c2.x;
    intersection_line_circle(c1,r1,u,v,p1,p2);
}
```

6.求三角形外接圆圆心

```
struct Point
{
    double x;
    double y;
}pt[1005];
struct Traingle
{
    struct Point p[3];
};
struct Circle
{
        struct Point center;
        double r;
}ans;
//计算两点距离
double Dis(struct Point p, struct Point q)
{
        double dx=p.x-q.x;
        double dy=p.y-q.y;
        return sqrt(dx*dx+dy*dy);
}
//计算三角形面积
double Area(struct Traingle ct)
{
        return
fabs((ct.p[1].x-ct.p[0].x)*(ct.p[2].y-ct.p[0].y)-(ct.p[2].x-ct.p[0].x)*(ct.p[1].y-ct.p[0].y))/2.0;
}
//求三角形的外接圆，返回圆心和半径(存在结构体"圆"中)
struct Circle CircumCircle(struct Traingle t)
{
        struct Circle tmp;
        double a, b, c, c1, c2;
        double xA, yA, xB, yB, xC, yC;
        a = Dis(t.p[0], t.p[1]);
        b = Dis(t.p[1], t.p[2]);
        c = Dis(t.p[2], t.p[0]);
        //根据 S = a * b * c / R / 4;求半径 R
        tmp.r = (a*b*c)/(Area(t)*4.0);
        xA = t.p[0].x;
        yA = t.p[0].y;
        xB = t.p[1].x;
        yB = t.p[1].y;
        xC = t.p[2].x;
        yC = t.p[2].y;
        c1 = (xA*xA+yA*yA - xB*xB-yB*yB) / 2;
        c2 = (xA*xA+yA*yA - xC*xC-yC*yC) / 2;
        tmp.center.x = (c1*(yA - yC)-c2*(yA - yB)) / ((xA - xB)*(yA - yC)-(xA - xC)*(yA - yB));
        tmp.center.y = (c1*(xA - xC)-c2*(xA - xB)) / ((yA - yB)*(xA - xC)-(yA - yC)*(xA - xB));
        return tmp;
}
```

7.求凸包

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define INF 999999999.9
#define PI acos(-1.0)
struct Point
{
    double x, y, dis;
}pt[1005], stack[1005], p0;
int top, tot;
//计算几何距离
double Dis(double x1, double y1, double x2, double y2)
{
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
//极角比较， 返回-1: p0p1 在 p0p2 的右侧，返回 0:p0,p1,p2 共线
int Cmp_PolarAngel(struct Point p1, struct Point p2, struct Point pb)
{
    double delta=(p1.x-pb.x)*(p2.y-pb.y)-(p2.x-pb.x)*(p1.y-pb.y);
    if (delta<0.0) return 1;
    else if (delta==0.0) return 0;
    else return -1;
}
// 判断向量 p2p3 是否对 p1p2 构成左旋
bool Is_LeftTurn(struct Point p3, struct Point p2, struct Point p1)
{
    int type=Cmp_PolarAngel(p3, p1, p2);
    if (type<0) return true;
    return false;
}
//先按极角排，再按距离由小到大排
int Cmp(const void*p1, const void*p2)
{
    struct Point*a1=(struct Point*)p1;
    struct Point*a2=(struct Point*)p2;
    int type=Cmp_PolarAngel(*a1, *a2, p0);
    if (type<0) return -1;
    else if (type==0)
    {
        if (a1->dis<a2->dis) return -1;
        else if (a1->dis==a2->dis) return 0;
        else return 1;
    }
    else return 1;
}
//求凸包
void Solve(int n)
{
    int i, k;
    p0.x=p0.y=INF;
    for (i=0;i<n;i++)
    {
        scanf("%lf %lf",&pt[i].x, &pt[i].y);
        if (pt[i].y < p0.y)
        {
            p0.y=pt[i].y;
            p0.x=pt[i].x;
            k=i;
        }
        else if (pt[i].y==p0.y)
        {
            if (pt[i].x<p0.x)
            {
                p0.x=pt[i].x;
                k=i;
            }
        }
    }
    pt[k]=pt[0];
    pt[0]=p0;
    for (i=1;i<n;i++)
        pt[i].dis=Dis(pt[i].x,pt[i].y, p0.x,p0.y);
    qsort(pt+1, n-1, sizeof(struct Point), Cmp);
    //去掉极角相同的点
    tot=1;
    for (i=2;i<n;i++)
        if (Cmp_PolarAngel(pt[i], pt[i-1], p0))
            pt[tot++]=pt[i-1];
    pt[tot++]=pt[n-1];
    //求凸包
    top=1;
    stack[0]=pt[0];
    stack[1]=pt[1];
    for (i=2;i<tot;i++)
    {
        while (top>=1 && Is_LeftTurn(pt[i], stack[top], stack[top-1])==false)
            top--;
        stack[++top]=pt[i];
    }
}
int main (void)
{
    int n;
    while (scanf("%d",&n)==2)
    {
        Solve(n);
    }
    return 0;
}
```

8.凸包卡壳旋转求出所有对踵点、最远点对

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define INF 999999999.9
#define PI acos(-1.0)
struct Point
{
    double x, y, dis;
}pt[6005], stack[6005], p0;
int top, tot;
//计算几何距离
double Dis(double x1, double y1, double x2, double y2)
{
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
//极角比较， 返回-1: p0p1 在 p0p2 的右侧，返回 0:p0,p1,p2 共线
int Cmp_PolarAngel(struct Point p1, struct Point p2, struct Point pb)
{
    double delta=(p1.x-pb.x)*(p2.y-pb.y)-(p2.x-pb.x)*(p1.y-pb.y);
    if (delta<0.0) return 1;
    else if (delta==0.0) return 0;
    else return -1;
}
// 判断向量 p2p3 是否对 p1p2 构成左旋
```

```
bool Is_LeftTurn(struct Point p3, struct Point p2, struct Point p1)
{
    int type=Cmp_PolarAngel(p3, p1, p2);
    if (type<0) return true;
    return false;
}
//先按极角排，再按距离由小到大排
int Cmp(const void*p1, const void*p2)
{
    struct Point*a1=(struct Point*)p1;
    struct Point*a2=(struct Point*)p2;
    int type=Cmp_PolarAngel(*a1, *a2, p0);
    if (type<0) return -1;
    else if (type==0)
    {
        if (a1->dis<a2->dis) return -1;
        else if (a1->dis==a2->dis) return 0;
        else return 1;
    }
    else return 1;
}
//求凸包
void Hull(int n)
{
    int i, k;
    p0.x=p0.y=INF;
    for (i=0;i<n;i++)
    {
        scanf("%lf %lf",&pt[i].x, &pt[i].y);
        if (pt[i].y < p0.y)
        {
            p0.y=pt[i].y;
            p0.x=pt[i].x;
            k=i;
        }
        else if (pt[i].y==p0.y)
        {
            if (pt[i].x<p0.x)
            {
                p0.x=pt[i].x;
                k=i;
            }
        }
    }
    pt[k]=pt[0];
    pt[0]=p0;
    for (i=1;i<n;i++)
        pt[i].dis=Dis(pt[i].x,pt[i].y, p0.x,p0.y);
    qsort(pt+1, n-1, sizeof(struct Point), Cmp);
    //去掉极角相同的点
    tot=1;
    for (i=2;i<n;i++)
        if (Cmp_PolarAngel(pt[i], pt[i-1], p0))
            pt[tot++]=pt[i-1];
    pt[tot++]=pt[n-1];
    //求凸包
    top=1;
    stack[0]=pt[0];
    stack[1]=pt[1];
    for (i=2;i<tot;i++)
    {
        while (top>=1 && Is_LeftTurn(pt[i], stack[top], stack[top-1])==false)
            top--;
        stack[++top]=pt[i];
    }
}
//计算叉积
double CrossProduct(struct Point p1, struct Point p2, struct Point p3)
{
    return (p1.x-p3.x)*(p2.y-p3.y)-(p2.x-p3.x)*(p1.y-p3.y);
}
//卡壳旋转，求出凸多边形所有对踵点
void Rotate(struct Point*ch, int n)
{
    int i, p=1;
    double t1, t2, ans=0.0, dif;
    ch[n]=ch[0];
    for (i=0;i<n;i++)
    {
        //如果下一个点与当前边构成的三角形的面积更大，则说明此时不构成对踵点
        while                    (fabs(CrossProduct(ch[i],ch[i+1],ch[p+1]))                    >
fabs(CrossProduct(ch[i],ch[i+1],ch[p])))
            p=(p+1)%n;
        dif=fabs(CrossProduct(ch[i],ch[i+1],ch[p+1])) - fabs(CrossProduct(ch[i],ch[i+1],ch[p]));
        //如果当前点和下一个点分别构成的三角形面积相等，则说明两条边即为平行线，
对角线两端都可能是对踵点
        if (dif==0.0)
        {
            t1=Dis(ch[p].x, ch[p].y, ch[i].x, ch[i].y);
            t2=Dis(ch[p+1].x, ch[p+1].y, ch[i+1].x, ch[i+1].y);
            if (t1>ans)ans=t1;
            if (t2>ans)ans=t2;
        }
        //说明 p，i 是对踵点
        else if (dif<0.0)
        {
            t1=Dis(ch[p].x, ch[p].y, ch[i].x, ch[i].y);
            if (t1>ans)ans=t1;
        }
    }
    printf("%.2lf\n",ans);
}
int main (void)
{
    int n;
    while (scanf("%d",&n)==1)
    {
        Hull(n);
        Rotate(stack, top+1);
    }
    return 0;
}
```

9.凸包+旋转卡壳求平面面积最大三角

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define INF 99999999999.9
#define PI acos(-1.0)
struct Point
{
    double x, y, dis;
}pt[50005], stack[50005], p0;
int top, tot;
double Dis(double x1, double y1, double x2, double y2)
{
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
```

```
int Cmp_PolarAngel(struct Point p1, struct Point p2, struct Point pb)
{
    double delta=(p1.x-pb.x)*(p2.y-pb.y)-(p2.x-pb.x)*(p1.y-pb.y);
    if (delta<0.0) return 1;
    else if (delta==0.0) return 0;
    else return -1;
}
bool Is_LeftTurn(struct Point p3, struct Point p2, struct Point p1)
{
    int type=Cmp_PolarAngel(p3, p1, p2);
    if (type<0) return true;
    return false;
}
int Cmp(const void*p1, const void*p2)
{
    struct Point*a1=(struct Point*)p1;
    struct Point*a2=(struct Point*)p2;
    int type=Cmp_PolarAngel(*a1, *a2, p0);
    if (type<0) return -1;
    else if (type==0)
    {
        if (a1->dis<a2->dis) return -1;
        else if (a1->dis==a2->dis) return 0;
        else return 1;
    }
    else return 1;
}
void Hull(int n)
{
    int i, k;
    p0.x=p0.y=INF;
    for (i=0;i<n;i++)
    {
        scanf("%lf %lf",&pt[i].x, &pt[i].y);
        if (pt[i].y < p0.y)
        {
            p0.y=pt[i].y;
            p0.x=pt[i].x;
            k=i;
        }
        else if (pt[i].y==p0.y)
        {
            if (pt[i].x<p0.x)
            {
                p0.x=pt[i].x;
                k=i;
            }
        }
    }
    pt[k]=pt[0];
    pt[0]=p0;
    for (i=1;i<n;i++)
        pt[i].dis=Dis(pt[i].x,pt[i].y, p0.x,p0.y);
    qsort(pt+1, n-1, sizeof(struct Point), Cmp);
    tot=1;
    for (i=2;i<n;i++)
        if (Cmp_PolarAngel(pt[i], pt[i-1], p0))
            pt[tot++]=pt[i-1];
    pt[tot++]=pt[n-1];
    top=1;
    stack[0]=pt[0];
    stack[1]=pt[1];
    for (i=2;i<tot;i++)
    {
        while (top>=1 && Is_LeftTurn(pt[i], stack[top], stack[top-1])==false)
            top--;
        stack[++top]=pt[i];
    }
}
double TArea(struct Point p1, struct Point p2, struct Point p3)
{
    return fabs((p1.x-p3.x)*(p2.y-p3.y)-(p2.x-p3.x)*(p1.y-p3.y));
}
void Rotate(struct Point*ch, int n)
{
    if (n<3)
    {
        printf("0.00\n");
        return;
    }
    int i, j, k;
    double ans=0.0, tmp;
    ch[n]=ch[0];
    for (i=0;i<n;i++)
    {
        j=(i+1)%n;
        k=(j+1)%n;
        while ((j!=k) && (k!=i))
```

```
		{
			while (TArea(ch[i],ch[j],ch[k+1])>TArea(ch[i],ch[j],ch[k]))
				k=(k+1)%n;
			tmp=TArea(ch[i],ch[j], ch[k]);
			if (tmp>ans) ans=tmp;
			j=(j+1)%n;
		}
	}
	printf("%.2lf\n",ans/2.0);
}
int main (void)
{
	int n;
	while (scanf("%d",&n)==1)
	{
		if (n==-1)break;
		Hull(n);
		Rotate(stack, top+1);
	}
	return 0;
}
```

10.Pick 定理

// Pick 定理求整点多边形内部整点数目
// (1) 给定顶点座标均是整点（或正方形格点）的简单多边形，皮克定理说明了其面积 A 和内部格点数目 i、边上格点数目 b 的关系：A = i + b/2 - 1；
// (2) 在两点（x1，y1），（x2，y2）连线之间的整点个数（包含一个端点）为：gcd（|x1－x2|，|y1－y2|）；
// (3) 求三角形面积用叉乘

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

long long x[3], y[3], area, b;
long long My_Abs(long long t)
{
	if (t<0) return -t;
	return t;
}
long long Gcd(long long x, long long y)
```

```
{
    if (y==0) return x;
    long long mod=x%y;
    while (mod)
    {
        x=y;
        y=mod;
        mod=x%y;
    }
    return y;
}
int main (void)
{
    int i;
    while (1)
    {
        for (i = 0;i < 3;i ++)
            scanf("%lld %lld", &x[i], &y[i]);
        if(x[0]==0&&y[0]==0&&x[1]==0&&y[1]==0&&x[2]==0&&y[2]==0) break;
        area = (x[1]-x[0])*(y[2]-y[0])-(x[2]-x[0])*(y[1]-y[0]);
        area = My_Abs(area);
        b=0;
        b=Gcd(My_Abs(x[1]-x[0]), My_Abs(y[1]-y[0])) + Gcd(My_Abs(x[2]-x[0]), My_Abs(y[2]-y[0])) + Gcd(My_Abs(x[1]-x[2]), My_Abs(y[1]-y[2]));
        printf("%lld\n", (area-b+2)/2);
    }
    return 0;
}
```

11.求多边形面积和重心

```
#include <stdio.h>
#include <math.h>
int x[1000003], y[1000003];
double A, tx, ty, tmp;
int main (void)
{
    int cases, n, i;
    scanf ("%d", &cases);
    while (cases --)
    {
        scanf ("%d", &n);
        A = 0.0;
        x[0] = y[0] = 0;
        for (i = 1; i <= n; i ++)
        {
            scanf ("%d %d", &x[i], &y[i]);
            A += (x[i-1]*y[i] - x[i]*y[i-1]);
        }
        A += x[n]*y[1] - x[1]*y[n];
        A = A / 2.0;
        tx = ty = 0.0;
        for (i = 1; i < n; i ++)
        {
            tmp = x[i]*y[i+1] - x[i+1]*y[i];
            tx += (x[i]+x[i+1]) * tmp;
            ty += (y[i]+y[i+1]) * tmp;
        }
        tmp = x[n]*y[1] - x[1]*y[n];
        tx += (x[n]+x[1])*tmp;
        ty += (y[n]+y[1])*tmp;
        printf ("%.2lf %.2lf\n", tx/(6.0*A), ty/(6.0*A));
    }
    return 0;
}
```

12.判断一个简单多边形是否有核

```
#include <stdio.h>
#include <string.h>
const int INF = (1<<30);
struct Point
{
    int x, y;
}pt[150];
typedef struct Point Point;
bool turn_right[150];
int det(Point s1, Point t1, Point s2, Point t2)
{
    int d1x = t1.x-s1.x;
    int d1y = t1.y-s1.y;

    int d2x = t2.x-s2.x;
    int d2y = t2.y-s2.y;

    return d1x*d2y - d2x*d1y;
```

```
}
void Swap(int &a, int &b)
{
        if (a>b)
        {
                int t=a;
                a=b;
                b=t;
        }
}
int main (void)
{
        int n, i, cross, maxx, minx, maxy, miny, maxn, minn, countn=0;
        while (scanf("%d", &n)==1&&n)
        {
                maxx=maxy=-INF;
                minx=miny=INF;
                //点按顺时针给出
                for (i=1; i<=n; i++)
                {
                        scanf("%d %d", &pt[i].x, &pt[i].y);
                        if (maxx<pt[i].x) maxx=pt[i].x;
                        if (maxy<pt[i].y) maxy=pt[i].y;
                        if (minx>pt[i].x) minx=pt[i].x;
                        if (miny>pt[i].y) miny=pt[i].y;
                }
                pt[n+1]=pt[1];
                pt[n+2]=pt[2];
                pt[n+3]=pt[3];
                pt[n+4]=pt[4];
                //求每条线段的转向
                for (i=1; i<=n+1; i ++)
                {
                        cross = det(pt[i],pt[i+1], pt[i+1], pt[i+2]);
                        if (cross<0)
                                turn_right[i+1]=true;
                        else turn_right[i+1]=false;
                }
                //两条边连续右转的为凸处，只有此时才可影响“核”肯恩存在的范围
                for (i=2; i<= n+1; i++)
                        if (turn_right[i] && turn_right[i+1])
                        {
                                if (pt[i].x==pt[i+1].x)
                                {
```

```
                    minn=pt[i].y;
                    maxn=pt[i+1].y;
                    Swap(minn, maxn);
                    if (minn>miny) miny=minn;
                    if (maxn<maxy) maxy=maxn;
                }
                else
                {
                    minn=pt[i].x;
                    maxn=pt[i+1].x;
                    Swap(minn, maxn);
                    if (minn>minx) minx=minn;
                    if (maxn<maxx) maxx=maxn;
                }
            }
        if (minx<=maxx && miny<=maxy)
            printf("Floor #%d\nSurveillance is possible.\n\n", ++countn);
        else printf("Floor #%d\nSurveillance is impossible.\n\n", ++countn);
    }
    return 0;
}
```

13.模拟退火

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Lim 0.999999
#define EPS 1e-2
#define PI acos(-1.0)
double Temp, maxx, minx, maxy, miny, lx, ly, dif;
int nt, ns, nc;
struct Target
{
    double x, y;
}T[105];
struct Solution
{
    double x, y;
    double f;
}S[25], P, A;
double Dis(double x1, double y1, double x2, double y2)
```

```
{
        return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
void Seed(void)
{
        int i, j;
        for (i=0;i<ns;i++)
        {
                S[i].x=minx+((double)(rand()%1000+1)/1000.0)*lx;
                S[i].y=miny+((double)(rand()%1000+1)/1000.0)*ly;
                S[i].f=0.0;
                for (j=0;j<nt;j++)
                        S[i].f=S[i].f+Dis(S[i].x,S[i].y, T[j].x, T[j].y);
        }
}
void Trans(void)
{
        int i, j, k;
        double theta;
        for (i=0;i<ns;i++)
        {
                P=S[i];
                for (j=0;j<nc;j++)
                {
                        theta=(((double)(rand()%1000+1))/1000.0)*2.0*PI;
                        A.x=P.x+Temp*cos(theta);
                        A.y=P.y+Temp*sin(theta);
                        if (A.x<minx||A.x>maxx||A.y<miny||A.y>maxy)
                                continue;
                        A.f=0.0;
                        for (k=0;k<nt;k++)
                                A.f=A.f+Dis(A.x,A.y,T[k].x,T[k].y);
                        dif=A.f-S[i].f;
                        if (dif<0.0)S[i]=A;
                        else
                        {
                                dif=exp(-dif/Temp);
                                if (dif>Lim) S[i]=A;
                        }
                }
        }
}
int main (void)
{
    int i, k;
    while (scanf("%d",&nt)==1&&nt)
    {
        maxx=maxy=0;
        minx=miny=(1<<20);
        for (i=0;i<nt;i++)
        {
            scanf("%lf %lf",&T[i].x,&T[i].y);
            if (maxx<T[i].x)maxx=T[i].x;
            if (minx>T[i].x)minx=T[i].x;
            if (maxy<T[i].y)maxy=T[i].y;
            if (miny>T[i].y)miny=T[i].y;
        }
        lx=maxx-minx;
        ly=maxy-miny;
        Temp=sqrt(lx*lx+ly*ly)/3.0;
        ns=5, nc=10;
        Seed();
        while (Temp>EPS)
        {
            Trans();
            Temp=Temp*0.40;
        }
        k=0;
        for (i=1;i<ns;i++)
            if (S[k].f>S[i].f)
                k=i;
        printf ("%.0lf\n", S[k].f);
    }
    return 0;
}
```

14.六边形坐标系

```
//第一种六边形坐标系
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>
double Dis(double x1, double y1, double x2, double y2)
{
    double dx=x1-x2;
    double dy=y1-y2;
    return sqrt(dx*dx+dy*dy);
}
void Get_KL(double L, double x, double y, int &k, int &l, double &cd)
{
    k=floor((2.0*x)/(3.0*L));
    l=floor((2.0*y)/(sqrt(3.0)*L));
    double d1, d2, x1, y1, x2, y2;
    if ((k+l)&1)
    {
        x1=k*L*1.5;
        y1=(l+1.0)*L*sqrt(3.0)*0.5;
        x2=(k+1.0)*L*1.5;
        y2=l*L*sqrt(3.0)*0.5;
        d1=Dis(x1,y1, x,y);
        d2=Dis(x2,y2, x,y);
        if (d1>d2)
        {
            k++;
            cd=d2;
        }
        else
        {
            l++;
            cd=d1;
        }
    }
    else
    {
        x1=k*L*1.5;
        y1=l*L*sqrt(3.0)*0.5;
        x2=(k+1.0)*L*1.5;
        y2=(l+1.0)*L*sqrt(3.0)*0.5;
        d1=Dis(x1,y1, x,y);
        d2=Dis(x2,y2, x,y);
        if (d1>d2)
        {
            k++,l++;
            cd=d2;
        }
        else cd=d1;
    }
}
int My_Abs(int x)
{
        if (x<0) return -x;
        return x;
}
int main (void)
{
        double L, x1, y1, x2, y2, ans, cd1, cd2;
        int k1, l1, k2, l2;
        while (scanf("%lf %lf %lf %lf %lf",&L,&x1,&y1,&x2,&y2)==5)
        {
                if (L==0.0&&x1==0.0&&y1==0.0&&x2==0.0&&y2==0.0) break;
                Get_KL(L, x1, y1, k1, l1, cd1);
                Get_KL(L, x2, y2, k2, l2, cd2);
                if (k1==k2&&l1==l2) printf("%.3lf\n", Dis(x1,y1, x2,y2));
                else
                {
                        ans=cd1+cd2;
                        if (My_Abs(k1-k2) > My_Abs(l1-l2))
                                ans=ans+sqrt(3.0)*L*My_Abs(k1-k2);
                        else
ans=ans+sqrt(3.0)*L*My_Abs(k1-k2)+sqrt(3.0)*L*(double)(My_Abs(l1-l2)-My_Abs(k1-k2))/2.0
;
                        printf("%.3lf\n", ans);
                }
        }
        return 0;
}
```

//第二种六边形坐标系

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
struct A
{
        int x, y, num;
}a[10001];
const int dec[6][2] = {{-1,1},{-1,0},{0,-1},{1,-1},{1,0},{0,1}};
bool adj(int x1, int y1, int x2, int y2)
{
        if (x1 == x2 && abs(y1-y2) == 1) return true;
        if (y1 == y2 && abs(x1-x2) == 1) return true;
        if (x1 == x2 + 1 && y1 == y2 -1) return true;
        if (x1 == x2 - 1 && y1 == y2 +1) return true;
        return false;
```

```
}
bool flag[10001];
int main (void)
{
    int i, j, k, x, u, v, cut, minn, cnt[6];
    memset(cnt, 0, sizeof(cnt));
    a[1].num = 1, cnt[1] = 1;
    a[1].x = a[1].y = 0;
    for (i = 2; i < 10001; i ++)
    {
        k = (int)((3.0+sqrt(12.0*i - 3.0))/6.0+0.0000001);
        if (i == 3*(k-1)*(k-1)+3*(k-1)+1) k    --;
        j = i - (3*(k-1)*(k-1)+3*(k-1)+1);
        // 当前的六边形是第 k 层的第 j 个六边形
        if (j == 1) a[i].x = a[i-1].x, a[i].y = a[i-1].y + 1;
        else
        {
            x = (j-1) / k;
            a[i].x = a[i-1].x + dec[x][0], a[i].y = a[i-1].y + dec[x][1];
        }
        memset(flag, false, sizeof(flag));
        x = 12*k-6, cut = 0;
        for (u = i-1, v = 0; u>=1&&v<x; u --, v ++)
            if (adj(a[u].x, a[u].y, a[i].x, a[i].y))
            {
                cut ++;
                flag[a[u].num] = true;
                if (cut == 3) break;
            }
        minn = 10001;
        for (u = 1; u < 6; u ++)
            if ((!flag[u])&&minn > cnt[u])
            {
                minn = cnt[u];
                x = u;
            }
        a[i].num = x;
        cnt[x] ++;
    }
    scanf ("%d", &x);
    while (x --)
    {
        scanf ("%d", &i);
        printf ("%d\n", a[i].num);
```

```
	}
	return 0;
}
```

15.用一个给定半径的圆覆盖最多的点

```
//同半径圆的圆弧表示
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define PI acos(-1.0)
struct Point
{
	double x, y;
}pt[2005];
double dis[2005][2005];
struct List
{
	double a;
	bool flag;
	int id;
}list[8005];
int cnt;
double Dis(int i, int j)
{
	double dx=pt[i].x-pt[j].x;
	double dy=pt[i].y-pt[j].y;
	return sqrt(dx*dx+dy*dy);
}
int Cmp(const void*p1, const void*p2)
{
	struct List*a1=(struct List*)p1;
	struct List*a2=(struct List*)p2;
	if (a1->a<a2->a)return -1;
	else if (a1->a==a2->a) return a1->id-a2->id;
	else return 1;
}
int main (void)
{
	int n, i, j, ans, num;
	double r, theta, delta, a1, a2;
	while (scanf("%d %lf",&n,&r)==2)
```

```
{
    if (n==0&&r==0.0) break;
    r=r+0.001;
    r=r*2.0;
    for (i=1;i<=n;i++)
        scanf("%lf %lf", &pt[i].x, &pt[i].y);
    for (i=1;i<n;i++)
        for (j=i+1;j<=n;j++)
        {
            dis[i][j]=Dis(i, j);
            dis[j][i]=dis[i][j];
        }
    ans=0;
    for (i=1;i<=n;i++)
    {
        cnt=0;
        for (j=1;j<=n;j++)
            if ((j!=i)&&(dis[i][j]<=r))
            {
                theta=atan2(pt[j].y-pt[i].y, pt[j].x-pt[i].x);
                if (theta<0.0) theta=theta+2.0*PI;
                delta=acos(dis[i][j]/r);
                a1=theta-delta;
                a2=theta+delta;
                list[++cnt].a=a1;
                list[cnt].flag=true;
                list[cnt].id=cnt;
                list[++cnt].a=a2;
                list[cnt].flag=false;
                list[cnt].id=cnt;
            }
        qsort(list+1,cnt,sizeof(struct List),Cmp);
        num=0;
        for (j=1;j<=cnt;j++)
            if (list[j].flag)
            {
                num++;
                if (num>ans) ans=num;
            }
            else num--;
    }
    printf("It is possible to cover %d points.\n", ans+1);
}
return 0;
```

```
}
```

16.不等大的圆的圆弧表示

```
intersection_circle_circle(circle[i].center, circle[i].r, circle[j].center, circle[j].r, p1, p2);
                    a1= atan2(p1.y-circle[j].center.y, p1.x-circle[j].center.x);
                    if (a1<0.0) a1=a1+2.0*PI;
                    a2= atan2(p2.y-circle[j].center.y, p2.x-circle[j].center.x);
                    if (a2<0.0) a2=a2+2.0*PI;

                    if (a1>a2)
                    {
                        tmp=a1;
                        a1=a2;
                        a2=tmp;
                    }
                    mid=(a1+a2)/2.0;
                    xtest = circle[j].center.x +circle[j].r*cos(mid);
                    ytest = circle[j].center.y +circle[j].r*sin(mid);

                    if (!point_in_circle(xtest, ytest, i))
                    {
                        circle[j].cnt++;
                        circle[j].line[circle[j].cnt].s=0;
                        circle[j].line[circle[j].cnt].t=a1;
                        circle[j].cnt++;
                        circle[j].line[circle[j].cnt].s=a2;
                        circle[j].line[circle[j].cnt].t=2.0*PI;
                    }
                    else
                    {
                        circle[j].cnt++;
                        circle[j].line[circle[j].cnt].s=a1;
                        circle[j].line[circle[j].cnt].t=a2;
                    }
```

17.矩形面积并

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

```
#include<math.h>
struct Node
{
    int l, r, cnt;
    double cover;
}node[80005];
struct Point
{
    double x;
    double y1, y2;
    int id_y1, id_y2, id_x;
    bool flag;
}pt[20005];
double y[20005];
int total, cnty;
int cmp1(const void*p1, const void*p2)
{
    double*a1=(double*)p1;
    double*a2=(double*)p2;
    if (*a1<*a2) return -1;
    else if (*a1==*a2) return 0;
    else return 1;
}
int cmp2(const void*p1, const void*p2)
{
    struct Point*a1=(struct Point*)p1;
    struct Point*a2=(struct Point*)p2;
    if (a1->x<a2->x) return -1;
    else if (a1->x==a2->x)
    {
        if (a1->id_x<a2->id_x) return -1;
        else if (a1->id_x==a2->id_x) return 0;
        else return 1;
    }
    else return 1;
}
int find(double target)
{
    int head=1, tail=cnty, mid;
    while (head<=tail)
    {
        mid=(head+tail)>>1;
        if (y[mid]==target) return mid;
        else if (y[mid]<target) head=mid+1;
```

```
            else tail=mid-1;
        }
        return 0;
}
void Build(int l, int r, int s)
{
        node[s].l=l;
        node[s].r=r;
        node[s].cnt=0;
        node[s].cover=0.0;
        if (l+1<r)
        {
                int mid=(l+r)>>1;
                Build(l,mid,s<<1);
                Build(mid,r,(s<<1)+1);
        }
}
void Update(int s)
{
        if (node[s].cnt>0)
                node[s].cover=y[node[s].r]-y[node[s].l];
        else if(node[s].l+1==node[s].r)
                node[s].cover=0.0;
        else node[s].cover=node[s<<1].cover+node[(s<<1)+1].cover;
}
void Insert(int l, int r, int s)
{
        if (l<=node[s].l&&node[s].r<=r)
        {
                node[s].cnt++;
                Update(s);
                return;
        }
        if (node[s].l+1<node[s].r)
        {
                int mid=(node[s].l+node[s].r)>>1;
                if (l<mid) Insert(l,r,s<<1);
                if (r>mid) Insert(l,r,(s<<1)+1);
                Update(s);
        }
}
void Delete(int l, int r, int s)
{
        if (l<=node[s].l&&node[s].r<=r)
```

```
    {
        if (node[s].cnt>0)
            node[s].cnt--;
        Update(s);
        return;
    }
    if (node[s].l+1<node[s].r)
    {
        int mid=(node[s].l+node[s].r)>>1;
        if (l<mid) Delete(l,r,s<<1);
        if (r>mid) Delete(l,r,(s<<1)+1);
        Update(s);
    }
}
int main (void)
{
    int n, i, j, countn=0;
    double ans;
    while (scanf("%d", &n)==1 && n)
    {
        cnty=total=0;
        for (i=1;i<=n;i++)
        {
            total++;
            scanf("%lf %lf", &pt[total].x, &pt[total].y1);
            pt[total].flag=true;
            pt[total].id_x=total;
            y[++cnty]=pt[total].y1;

            total++;
            scanf("%lf %lf", &pt[total].x, &pt[total].y2);
            pt[total].flag=false;
            pt[total].id_x=total;
            y[++cnty]=pt[total].y2;

            pt[total].y1=pt[total-1].y1;
            pt[total-1].y2=pt[total].y2;
        }
        qsort(y+1, cnty, sizeof(double), cmp1);
        j=cnty;
        cnty=1;
        for (i=2;i<=j;i++)
            if (y[i]!=y[i-1])
                y[++cnty]=y[i];
        for (i=1;i<=total;i++)
        {
            pt[i].id_y1=find(pt[i].y1);
            pt[i].id_y2=find(pt[i].y2);
        }
        qsort(pt+1, total, sizeof(struct Point), cmp2);

        ans=0.0;
        Build(1,cnty,1);
        Insert(pt[1].id_y1, pt[1].id_y2, 1);
        for (i=2;i<=total;i++)
        {
            ans=ans+(pt[i].x-pt[i-1].x)*node[1].cover;
            if (pt[i].flag) Insert(pt[i].id_y1, pt[i].id_y2, 1);
            else Delete(pt[i].id_y1, pt[i].id_y2, 1);
        }
        printf("%.0lf\n", ans+1e-10);
    }
    return 0;
}
```

18.矩形的周长并

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct Point
{
    int x, y;
}plist[10001];
struct Line
{
    int x, b, e, flag;
}llist[10001];
struct Item
{
    int y, id, idx;
}ilist[10001];
struct Node
{
    int l, r, c, m, line;
    bool lf, rf;
```

```
}node[40005];
int ys[10001];
int cmp1(const void*p1, const void*p2)
{
        struct Item *a1 = (struct Item*)p1;
        struct Item *a2 = (struct Item*)p2;
        return a1->y - a2->y;
}
int cmp2(const void*p1, const void*p2)
{
        struct Item *a1 = (struct Item*)p1;
        struct Item *a2 = (struct Item*)p2;
        return a1->id - a2->id;
}
int cmp3(const void*p1, const void*p2)
{
        struct Line *a1 = (struct Line*)p1;
        struct Line *a2 = (struct Line*)p2;
        return a1->x - a2->x;
}
void getm(int s)
{
        if (node[s].c > 0)
        {
                node[s].m = ys[node[s].r-1] - ys[node[s].l-1];
                node[s].line = 1;
                node[s].rf = node[s].lf = true;
        }
        else if (node[s].r - node[s].l <= 1)
        {
                node[s].m = node[s].line = 0;
                node[s].rf = node[s].lf = false;
        }
        else
        {
                node[s].m = node[s<<1].m + node[(s<<1)+1].m;
                node[s].line = node[s<<1].line + node[(s<<1)+1].line;
                if (node[s<<1].rf && node[(s<<1)+1].lf) node[s].line --;
                node[s].lf = node[s<<1].lf;
                node[s].rf = node[(s<<1)+1].rf;
        }
}
void build(int l, int r, int s)
{
        node[s].l = l;
        node[s].r = r;
        node[s].c = node[s].m = node[s].line;
        if (node[s].r - node[s].l > 1)
        {
                int mid = (node[s].l + node[s].r)>>1;
                build(l,mid,s<<1);
                build(mid,r,(s<<1)+1);
        }
}
void insert(int l, int r, int s)
{
        if (l <= node[s].l && node[s].r <= r)
        {
                node[s].c ++;
                getm(s);
        }
        if (node[s].r - node[s].l > 1)
        {
                int mid = (node[s].l + node[s].r)>>1;
                if (l < mid) insert(l, r, s<<1);
                if (mid < r) insert(l, r, (s<<1)+1);
                getm(s);
        }
}
void delet(int l, int r, int s)
{
        if (l <= node[s].l && node[s].r <= r)
        {
                node[s].c --;
                getm(s);
        }
        if (node[s].r - node[s].l > 1)
        {
                int mid = (node[s].l + node[s].r)>>1;
                if (l < mid) delet(l, r, s<<1);
                if (mid < r) delet(l, r, (s<<1)+1);
                getm(s);
        }
}
int main (void)
{
        int n, i, j, l, r, x1, y1, x2, y2, tot, p, ans;
        while (scanf ("%d", &n) == 1 && n)
```

```
{
        for (i = 0; i < n; i ++)
        {
                scanf ("%d %d %d %d", &x1, &y1, &x2, &y2);
                l = 2*i;
                r = l + 1;

                plist[l].x = x1;
                plist[l].y = y1;
                plist[r].x = x2;
                plist[r].y = y2;

                ilist[l].y = y1;
                ilist[l].id = l;
                ilist[r].y = y2;
                ilist[r].id = r;
        }
        tot = 2*n;
        qsort(ilist, tot, sizeof(struct Item), cmp1);
        ys[0] = ilist[0].y;
        ilist[0].idx = 0;
        j = 0;
        for (i = 1; i < tot; i ++)
        {
                if (ilist[i].y != ilist[i-1].y)
                {
                        j ++;
                        ys[j] = ilist[i].y;
                }
                ilist[i].idx = j;
        }
        p = j + 1;
        qsort(ilist, tot, sizeof(struct Item), cmp2);
        for (i = 0; i < n; i ++)
        {
                l = 2*i;
                r = l + 1;
                llist[l].x = plist[l].x;
                llist[l].b = ilist[l].idx;
                llist[l].e = ilist[r].idx;
                llist[l].flag = 1;

                llist[r].x = plist[r].x;
                llist[r].b = ilist[l].idx;
                llist[r].e = ilist[r].idx;
                llist[r].flag = 0;
            }
            qsort(llist, tot, sizeof(struct Line), cmp3);
            build(1,p,1);
            insert(llist[0].b+1, llist[0].e+1,1);
            int now_m = node[1].m, now_line = node[1].line;
            ans = now_m;
            for (i = 1; i < tot; i ++)
            {
                if (llist[i].flag) insert(llist[i].b+1, llist[i].e+1, 1);
                else delet(llist[i].b+1, llist[i].e+1, 1);
                ans += (abs(node[1].m - now_m) + 2*(llist[i].x - llist[i-1].x)*now_line);
                now_m = node[1].m;
                now_line = node[1].line;
            }
            printf ("%d\n", ans);
        }
        return 0;
}
```

19.最近圆对

```
#include<iostream>
#include<stdlib.h>
#include<string.h>
#include<set>
#include <math.h>
using namespace std;
set <int>tree;
set <int>::iterator iter;
struct Point
{
        double x;
        int id, flag;
}p1[100001], p2[100001];
int tot1, tot2;
struct Q
{
        double x,y, r;
}q[50001];
int cmp(const void*p1, const void*p2)
{
        struct Point*a1=(struct Point*)p1;
        struct Point*a2=(struct Point*)p2;
        if (a1->x<a2->x) return -1;
        else if (a1->x==a2->x) return a2->flag-a1->flag;
        else return 1;
}
int cmp1(const void*p1, const void*p2)
{
        struct Q*a1=(struct Q*)p1;
        struct Q*a2=(struct Q*)p2;
        if (a1->y<a2->y)return -1;
        else if (a1->y==a2->y)return 0;
        else return 1;
}
double dis(double x1, double y1, double x2, double y2)
{
        return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
bool judge(int i, int j, double d)
{
        if (dis(q[i].x, q[i].y, q[j].x, q[j].y)<=q[i].r+q[j].r+2.0*d)
            return true;
        return false;
}
bool insert(int v,double d)
{
        iter = tree.insert(v).first;
        if (iter != tree.begin())
        {
            if (judge(v, *--iter,d))
            {
                return true;
            }
            ++iter;
        }
        if (++iter != tree.end())
        {
            if (judge(v, *iter,d))
            {
                return true;
            }
        }
        return false;
}
```

```
bool remove(int v,double d)
{
    iter = tree.find(v);

    if (iter != tree.begin() && iter != --tree.end())
    {
        int a = *--iter;
        ++iter;
        int b = *++iter;
        if (judge(a, b,d))
        {
            return true;
        }
    }
    tree.erase(v);
    return false;
}
bool check(double d)
{
    int i=1, j=1;
    while (i<=tot1&&j<=tot2)
    {
        if (p1[i].x-d<=p2[j].x+d)
        {
            if (insert(p1[i++].id, d))
                return true;
        }
        else
        {
            if (remove(p2[j++].id, d))
                return true;
        }

    }
    while (i<=tot1)
    {
        if (insert(p1[i++].id, d))
            return true;
    }
    while (j<=tot2)
    {
        if (remove(p2[j++].id, d))
            return true;
    }
```

```
        return false;
}
int main (void)
{
        int cases, n, i;
        scanf("%d",&cases);
        while (cases--)
        {
                scanf("%d",&n);
                tot1=tot2=0;
                for (i=1;i<=n;i++)
                        scanf("%lf %lf %lf",&q[i].x,&q[i].y, &q[i].r);
                qsort(q+1,n,sizeof(struct Q),cmp1);
                for (i=1;i<=n;i++)
                {
                        tot1++;
                        p1[tot1].x=q[i].x-q[i].r;
                        p1[tot1].id=i;
                        p1[tot1].flag=1;

                        tot2++;
                        p2[tot2].x=q[i].x+q[i].r;
                        p2[tot2].id=i;
                        p2[tot2].flag=-1;
                }
                qsort(p1+1,tot1,sizeof(struct Point),cmp);
                qsort(p2+1,tot2,sizeof(struct Point),cmp);

                double head=0.0, tail=dis(q[1].x,q[1].y,q[2].x,q[2].y)+1.0, mid;
                while (tail-head>1e-8)
                {
                        tree.clear();
                        mid=(head+tail)/2.0;
                        if (check(mid))
                        {
                                tail=mid;
                        }
                        else head=mid;
                }
                printf ("%.6lf\n",2.0*head);
        }
        return 0;
}
```

20.求两个圆的面积交

```
double area_of_overlap(point c1, double r1, point c2, double r2)
{
    double a = distance(c1, c2), b = r1, c = r2;
    double cta1 = acos((a * a + b * b - c * c) / 2 / (a * b)),
           cta2 = acos((a * a + c * c - b * b) / 2 / (a * c));
    double s1 = r1*r1*cta1 - r1*r1*sin(cta1)*(a * a + b * b - c * c) / 2 / (a * b);
    double s2 = r2*r2*cta2 - r2*r2*sin(cta2)*(a * a + c * c - b * b) / 2 / (a * c);
    return s1 + s2;
}
```