智能地雷问题

摘要

智能地雷是一种新型地雷，能主动追踪并攻击目标，并且能够通过自主调节排布方位尽可能做到火力控制无死角。本文针对给出的智能地雷火力控制覆盖面积问题，提出了按照区域划分、构造模型、合理假设、循环遍历、程序求解、不足分析等步骤对此问题进行详细细致分析的方法，依据解决此问题特定的函数和对应的方法步骤，完成对这一问题的建模和求解。

对于问题 1，对给定的矩形区域进行划分，将地雷的攻击范围转化为圆覆盖问题。通过建立空间直角坐标系，将地雷抽象成一个点，地雷的攻击范围抽象为以点为圆心的圆，将地雷的数目和放置地点进行坐标化。分别通过圆内接三角形、四边形对给定区域进行划分，并使圆能够覆盖整个矩形区域，模拟地雷攻击范围的全覆盖。针对地雷的重新分布问题，通过改变圆内接多边形的形状对区域进行重新覆盖，减少地雷攻击范围重叠区域，提高地雷覆盖率，从而实现全覆盖。

对于问题 2，在地雷移动速度不变的情况下，要使地雷移动所用时间最少，即应满足在保证全覆盖的情况下，所有地雷在同时移动时移动距离最大的地雷在所有方案中移动距离最短，由此转化到点的移动距离及函数遍历问题。我们通过计算确定区域划分标准，然后利用循环程序对已经坐标化的数学模型进行对应遍历，遍历过程中运用检测函数对各顶点进行地雷火力覆盖检测，然后利用调整函数进行对应的调整，并改变覆盖圆的位置，如此反复进行，直到所有顶点均被智能地雷火力覆盖，亦即认为题设矩形区域完全被智能地雷火力控制范围覆盖，计算移动量最大的点在所有方案中的最小移动距离，完成对该问题的数学模型求解。

关键词：区域划分、以点代面、坐标化、循环遍历、函数检测调整

一、 问题重述

现在有一种被称为智能地雷的武器。智能地雷是“有腿地雷”，既能蹦，也能跳，还能飞，能够主动、准确地探测跟踪坦克、装甲战车，垂直攻击坦克的顶部或腹部。这些地雷之所以能够获得“智能”，就是因为它和探测技术、传感器技术、微处理器技术等高新技术结合在一起，所以展现出前所未有的活力。

在一块矩形区域内均匀分布着智能地雷，火力控制覆盖面积有部分重叠，按现在的地雷分布密度的 80%分布，火力控制仍能覆盖所有区域。当有部分地雷引爆后，火力控制出现死角时，地雷会自动调整位置重新分布尽可能做到不出现火力控制死角。

我们现在的问题是：

1.调整位置重新分布时，如何移动尽可能少的地雷（即不一定是全部移动，重新分布后不再是均匀的分布了）？

2.如何用最短的时间完成移动？

3.如果有时间，可以用计算机进行仿真。

二、 问题分析

2.1 问题 1 分析

首先给定的区域为矩形区域，需要多枚智能地雷方可做到火力控制区域完全覆盖给定面积，于是首先应对给定矩形区域进行对应的区域划分，做点一枚智能地雷的火力控制区域可以覆盖一块分割后的面积区域。

对于分割后的各部分区域，仍然是以面为单位，不便于建立数学模型和讨论求解，于是在合理假设的前提条件下，利用分割后小矩形区域四个顶点代替整个

小矩形区域，即只要四个顶点被智能地雷火力完全覆盖，便认为这四个顶点组成的矩形区域被智能地雷火力完全覆盖，将二维问题转化为一维问题。

为建立完善的数学模型，对给定矩形区域建立直角坐标系，并将讨论区域、顶点位置以及智能地雷数目和放置地点进行坐标化，从而建立起合理、完善的数学模型。

在建立起的数学模型的基础上，运用循环遍历、函数检测和调整等方式完成对该问题的求解。

2.2 问题 2 分析

在问题 1 的基础上，通过分析计算，求得在保证完全覆盖的情况下移动距离最大的点在所有方案中的最小移动距离，来表示要完成目标移动所需的最小时间。

三、模型假设

我们针对不同的问题，提出下列假设：

1、每一枚地雷的爆炸对其他智能地雷没有影响；

2、智能地雷火力控制范围覆盖检测函数依据具体问题而确定；

3、在一枚地雷火力控制范围内的矩形区域，是否被火力覆盖可归结为此区域的四个顶点是否被地雷火力覆盖；

4、给定矩形区域和初始智能地雷数目合理；

5、覆盖圆的圆心位置坐标可以精确定位，覆盖圆的半径大小相同；

四、定义与符号说明

A：矩形区域的长度

B：矩形区域的宽度

r：智能地雷爆炸半径

a：每个小矩形的长度

b：每个小矩形的宽度

i：长度单位（即 x 轴单位）

j：宽度单位（即 y 轴单位）

m：长度取值范围，即 0≤i≤m（m=A/a）

n：宽度取值范围，即 0≤j≤n（n=B/b）

K：开始时智能地雷个数

k：第 k 个职能地雷

X[K]：存储所有地雷 x 坐标值的一维数组

Y[K]：存储所有地雷 y 坐标值的一维数组

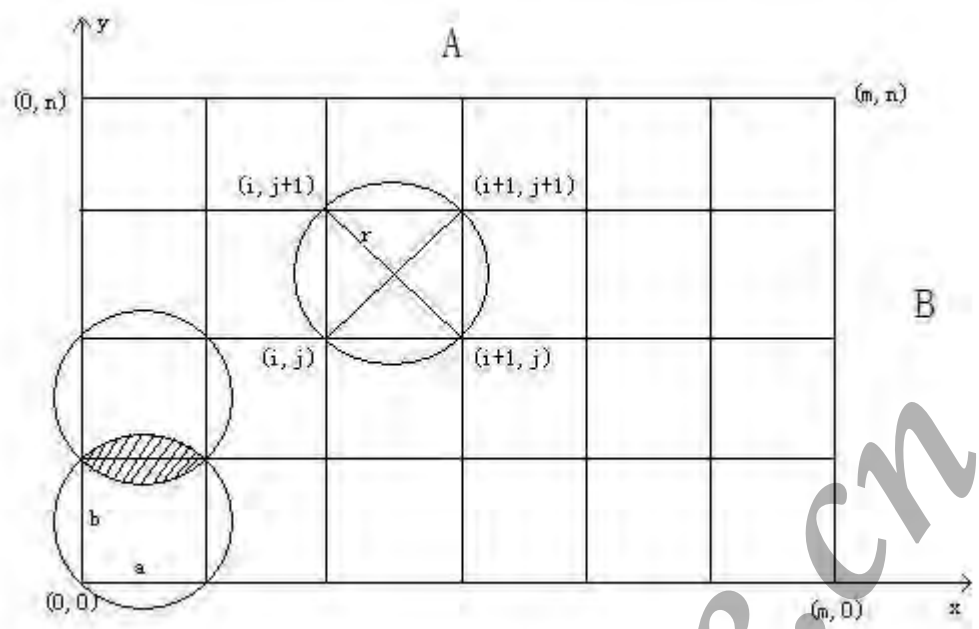
R_j^i：坐标为(i，j)的点

五、模型的建立与求解

5.1 区域划分

根据题目要求，首先依据如下原则对矩形区域进行划分，划分成若干形状相同的小矩形。

其中划分原则为：将矩形区域的长度 A 和宽度 B 分别按照 a 和 b 的单位跨度进行分割，并且同时满足关系式$(2r)^2 = a^2 + b^2$；划分后区域如图 1 所示：



注：图中每个小矩形的中心（对角线的交点）为此矩形的控制中心，圆形区域为控制中心位置地雷的火力覆盖面积，阴影部分为火力交叉区域。

5.2 模型分析与假设：

由图可见，整片矩形区域的火力覆盖问题可以简化为每一个小矩形火力覆盖问题的总和，即只要满足智能地雷的火力控制面积覆盖了每一个小矩形，亦即满足题目要求，智能地雷火力控制覆盖整片矩形区域、没有火力控制死角。

而根据矩形的划分原则和图示可得，每块小矩形区域是否被智能地雷完全火力覆盖可以归结为此矩形区域的四个顶点是否被地雷火力完全覆盖，即：要验证题设矩形区域是否被智能地雷完全火力覆盖，只需要验证划分后小矩形的所有顶点是否被完全火力覆盖即可。

于是可是使用两个 for 循环完成对这些顶点的遍历，以检验哪些顶点没有被地雷火力覆盖，并作出相应的调整。

注：for(i=0;i<=m;i++)

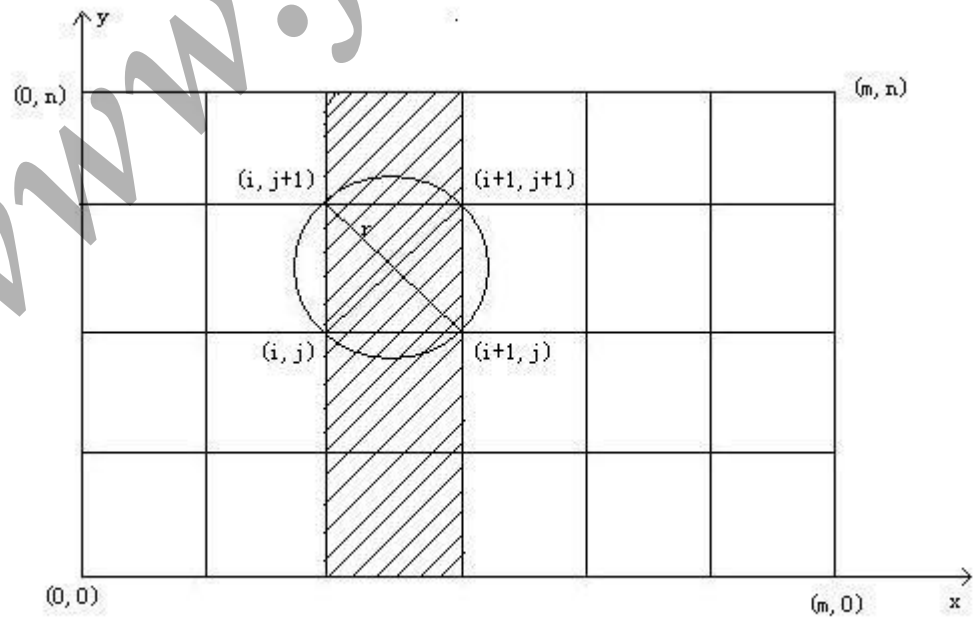
{for(j=0;j<=n;j++){……}

……}

5.3 问题解答：

首先利用给定数组 X[K]、Y[K]分别存储现有智能地雷的 x、y 坐标值，并在每爆炸一颗地雷后将此地雷对应坐标数据在 X、Y 数组中删掉，变为 X[K-1]、Y[K-1]。

利用两个 for 循环遍历所有顶点坐标(i，j)其中 0≤i≤m、0≤j≤n，利用函数 check(i，j)检验坐标（i，j)点处是否被地雷火力覆盖，如果被火力覆盖则函数返回值 true，否则返回 false；然后检验一个小矩形区域的四个顶点是否均被火力覆盖，如果某小矩形区域的四个顶点均被地雷火力覆盖、即 check(i，j)&&check(i+1，j)&&check(i，j+1)&&check(i+1，j+1)=true，则此区域不用调整，否则，如果 check(i，j)&&check(i+1，j)&&check(i，j+1)&&check(i+1，j+1)=false、即以(i+1/2，j+1/2)为中心的矩形区域没有完全覆盖，又由假设知 A≥B，则需要对横坐标在 i 与 i+1 之间（即 i≤X[k]≤i+1，如下图——图 2 中阴影部分所示）的地雷进行调整。



利用函数 distance(p，q)计算第 p 个地雷与第 q 个地雷的距离，找出在上图所示阴影区域中距离点(i+1/2，j+1/2)最近的地雷点 k，并对其做出相应的调整——运用移动函数 move(k，$R_{j+1/2}^{\ i+1/2}$)将地雷 k 移动至点(i+1/2，j+1/2)处，然后再利用 for 循环和 check 函数检验阴影矩形区域所包含的所有顶点是否还有未被智能地雷火力覆盖的点，如果有、则按照上述步骤继续进行调整；否则完成此次检查和调整，程序继续向下进行，直到(m+1)*(n+1)个顶点均被地雷火力覆盖为止。

5.4 条件限定：

由区域划分原则和图示可知，将题设给定矩形区域划分为 m*n 个小矩形区域，于是显然可得以此种方式安排智能地雷，至少需要 m*n 个智能地雷（且这 m*n 个地雷分布在每个小矩形区域的中心位置）才能使火力完全覆盖给定区域，做到在该区域内完全没有火力控制死角。也即是说当智能地雷数目少于 m*n 个时，不论如何调整均无法做到地雷火力控制完全覆盖给定矩形区域。

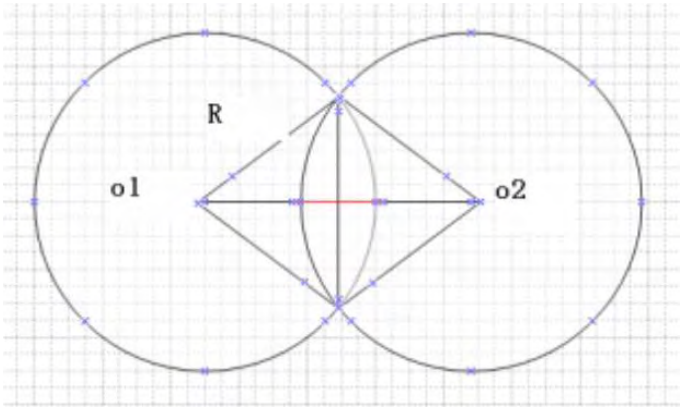
5.5 模型推广及样例

上述讨论的是圆内接矩形划分的覆盖问题，对于圆内接正三角形及正六边形的覆盖问题求解同上。

定理：若干半径为 R 的圆完全覆盖正方形区域的充分条件是这些圆的内接正多边形完全覆盖了正方形区域。

证明：假设正 n 边形 A_i 覆盖了正方形区域中的 B_i，A_i 所有 A_i（i=1,2,3……M），覆盖的区域包含了正方形区域 B，即所有 B_i 覆盖的区域包含了正方形区域 B。因为对于由 A_i 形成的外接圆 O_i，必包含区域 B_i,所以，所有 O_i（i=1,2,3……M）覆盖的区域必包含了正方形区域 B。

对于两个相交的圆：



相交面积 $s_1 = 2[\frac{\theta}{180}\pi R^2 - \frac{1}{2}R^2 \sin\theta]$

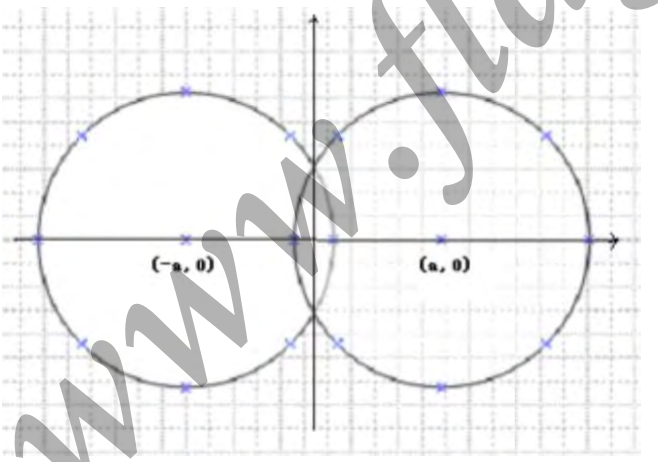
令 $k = \frac{s_1}{s_2} = \frac{\frac{\theta\pi R^2}{180} - R^2\sin\theta}{\pi R^2} = \frac{\theta}{180} - \frac{\sin\theta}{\pi}$

可得 k 值与圆的半径 R 无关

将 $\sin\theta$ 中的 θ 化为数量 $\sin\frac{\theta\pi}{180}$

$$\frac{dk}{d\theta} = \frac{d}{d\theta}(\frac{\theta}{180} - \frac{\sin\frac{\theta\pi}{180}}{\pi}) = \frac{1}{180}(1 - \cos\frac{\theta\pi}{180}) > 0$$

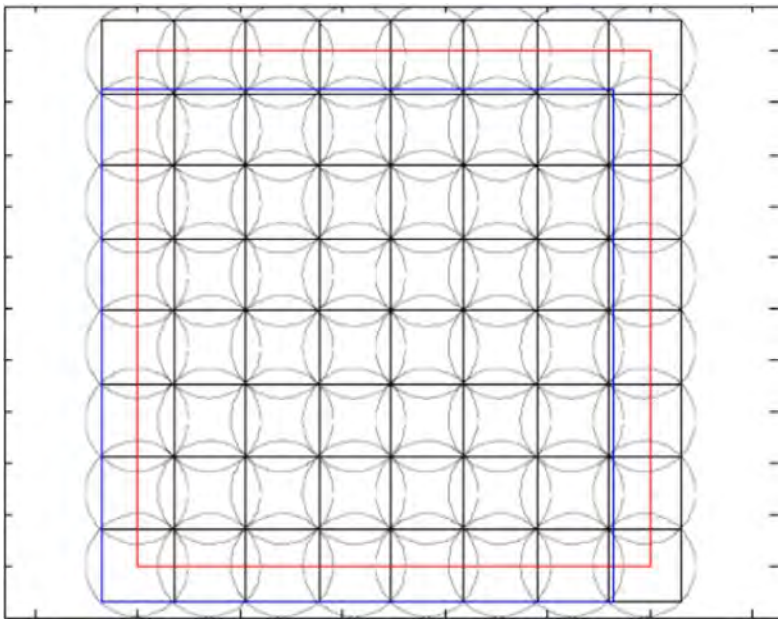
所以随着 θ 增大，k 的值是单调增加的并且增长速度满足正弦形式



随着 θ 的增大，k 的值是单调增加的并且增长速度满足正弦形式。
所以可以得到采用圆内接六边形的方式覆盖圆的利用率大于采用圆内接正方形覆盖。

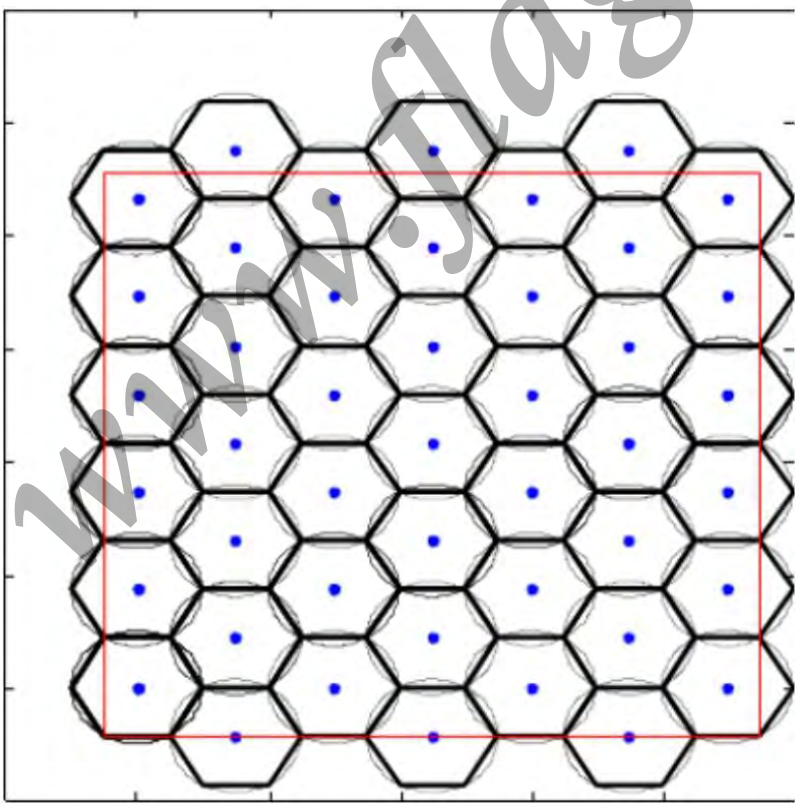
假设 A=1600，B=1600，r=$100\sqrt{2}$

使用内接正方形按如图所示方式覆盖时：



其内角为 90°，边长为 200,延两个方向上的贡献皆为 200，个数为 $\frac{1600}{200}=8$ 个，总数要 64 个;

当使用内接正六边形按如图所示方式覆盖时:



其内角为 120°，边长为 $100\sqrt{2}$，水平方向奇数行贡献 $100\sqrt{6}$，偶数行首位两个六

边形有效贡献为$50\sqrt{6}$，垂直方向贡献$150\sqrt{2}$（其中处于边界的一个正六边形仅贡献$100\sqrt{2}$。）

所以垂直方向需要正六边形个数为$\frac{1600}{100\sqrt{6}}$+1=7，水平方向共 7 行，奇数行每行需要 6 个，偶数行每行需要$\frac{1600-100\sqrt{2}}{150\sqrt{2}}$+1=7，所以需要正六边形的个数为 45 个。

即若原来分布的地雷数为 64 个，爆破后按内接正六边形覆盖只需要 45 个即可实现全覆盖。因为 64*80%=51 大于 45 个，所以该模拟实验结果符合题意。

针对问题 2，只需要每次地雷爆破后，邻近的最近的雷按照内接正六边形排列即可实现全覆盖，并通过 5.3 的循环算法求得移动最远的地雷在所有方案中的最小值，求得最优解。

六、模型的评价与推广

本文运用分割的方法对题目给定的矩形区域进行了处理，并通过检验划分后小矩形区域四个顶点的火力覆盖情况来代表对应小矩形区域的地雷火力覆盖情况，同时运用数组和 for 循环对分隔后的各个顶点进行遍历，并利用函数进行相应的检测和调整，已达到随时调整智能地雷分布以使得火力控制覆盖整个矩形区域、没有火力控制死角的目的。

七、附录（部分程序的伪代码）

7.1 check 函数：

check(i，j) //检测点为(i，j)点，第一个输入参数为待检测点的横坐标单位值，第二个输入参数为待检测点的纵坐标单位值，返回值为 true 或 flase：当被检测点被智能地雷火力控制范围所覆盖，则返回 true；否则返

回 false。

7.2 distance 函数：

distance(p，q) //计算点 p 与点 q 之间的距离，输入参数为 p 点和 q 点的横纵坐

标值，返回值为 p、q 点之间的距离值

公式为：distance(p，q)=$\sqrt{(x_p-x_q)^2+(y_p-y_q)^2}$

7.3 move 函数：

move(k，R_j^i) //将指定编号 k 的地雷移动到指定位置 R_j^i，输入参数一为指定移动的地雷编号，并同时调用该编号地雷的坐标值，输入参数二为指定的位置，同时调用该位置的坐标值；函数执行过程中，首先完成对两组坐标的调用，然后将被移动地雷的坐标值改为指定位置的对应坐标值，以此完成地雷的对应移动。

7.4 程序循环检测调整部分：

```
for(i=0;i<=m;i++)
    {for(j=0;j<=n;)
        {if(check(i，j)&&check(i+1，j)&&check(i，j+1)&&check(i+1，j+1)=false)
                //这四个点未被地雷火力完全覆盖，即至少一个未被地雷火力覆盖
            {int p,q,min=0;
             for(k=1;k<=K;k++)
               {if(X[k]>=i&&X[k]<=i+1){q=distance(k，R_{j+1/2}^{i+1/2});}  //计算距离
                if(q<min){min=q;p=k;} //找出距离最小的地雷编号和位置，并记录
               }
             move(p，R_{j+1/2}^{i+1/2}) //调整距离最近的地雷到指定位置(i+1/2, j+1/2)点
```

```
            j=0;//将 j 置零后再次进行循环，即对图 2 中阴影部分进行调整后检验

          }

        else j++;  //j 值继续增加，进行循环遍历

        }

    }
```

7.5 最小圆覆盖代码

```
#include<stdio.h>
#include<math.h>
struct   TPoint
{
         double x,y;
};
TPoint a[1005],d;
double r;

double   distance(TPoint   p1,   TPoint   p2)   //两点间距离
{
    return (sqrt((p1.x-p2.x)*(p1.x -p2.x)+(p1.y-p2.y)*(p1.y-
p2.y)));
}
double multiply(TPoint   p1,   TPoint   p2,   TPoint   p0)
{
    return   ((p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y));
}
void MiniDiscWith2Point(TPoint   p,TPoint   q,int   n)
{
    d.x=(p.x+q.x)/2.0;
    d.y=(p.y+q.y)/2.0;
    r=distance(p,q)/2;
    int k;
    double c1,c2,t1,t2,t3;
    for(k=1;k<=n;k++)
    {
        if(distance(d,a[k])<=r)continue;
        if(multiply(p,q,a[k])!=0.0)
        {
            c1=(p.x*p.x+p.y*p.y-q.x*q.x-q.y*q.y)/2.0;
            c2=(p.x*p.x+p.y*p.y-a[k].x*a[k].x-a[k].y*a[k].y)/2.0;
```

```
            d.x=(c1*(p.y-a[k].y)-c2*(p.y-q.y))/((p.x-q.x)*(p.y-
a[k].y)-(p.x-a[k].x)*(p.y-q.y));
            d.y=(c1*(p.x-a[k].x)-c2*(p.x-q.x))/((p.y-q.y)*(p.x-
a[k].x)-(p.y-a[k].y)*(p.x-q.x));
            r=distance(d,a[k]);
        }
        else
        {
            t1=distance(p,q);
            t2=distance(q,a[k]);
            t3=distance(p,a[k]);
            if(t1>=t2&&t1>=t3)

{d.x=(p.x+q.x)/2.0;d.y=(p.y+q.y)/2.0;r=distance(p,q)/2.0;}
            else if(t2>=t1&&t2>=t3)

{d.x=(a[k].x+q.x)/2.0;d.y=(a[k].y+q.y)/2.0;r=distance(a[k],q)/2.0;}
            else

{d.x=(a[k].x+p.x)/2.0;d.y=(a[k].y+p.y)/2.0;r=distance(a[k],p)/2.0;}
        }
    }
}

void MiniDiscWithPoint(TPoint  pi,int  n)
{
    d.x=(pi.x+a[1].x)/2.0;
    d.y=(pi.y+a[1].y)/2.0;
    r=distance(pi,a[1])/2.0;
    int j;
    for(j=2;j<=n;j++)
    {
    if(distance(d,a[j])<=r)continue;
        else
        {
            MiniDiscWith2Point(pi,a[j],j-1);
        }
    }
}
int main()
{
    int i,n;
    while(scanf("%d",&n)&&n)
    {
```

```
        for(i=1;i<=n;i++)
        {
            scanf("%lf %lf",&a[i].x,&a[i].y);
        }
        if(n==1)
        { printf("%.2lf %.2lf 0.00\n",a[1].x,a[1].y);continue;}
        r=distance(a[1],a[2])/2.0;
        d.x=(a[1].x+a[2].x)/2.0;
        d.y=(a[1].y+a[2].y)/2.0;
        for(i=3;i<=n;i++)
        {
            if(distance(d,a[i])<=r)continue;
            else
            MiniDiscWithPoint(a[i],i-1);
        }
        printf("%.2lf %.2lf %.2lf\n",d.x,d.y,r);
    }
    return 0;
}
```