计算几何在信息学奥赛中的应用

常州市第一中学　林厚从

一、引言

计算几何是几何学的一个重要分支，也是计算机科学的一个分支，研究解决几何问题的算法。在现代工程与数学、计算机图形学、机器人学、VLSI 设计、计算机辅助设计等学科领域中都有重要应用。

计算几何问题的输入一般是关于一组几何物体（如点、线）的描述；输出常常是有关这些物体的问题的回答，如直线是否相交，点围成的面积等问题。

最有名的例子就是凸包问题。

在最近几年的各项信息学竞赛中也出现了一些计算几何题，让人觉得耳目一新，但实际情况表明，信息学竞赛中的计算几何题得分率往往是最低的。究其原因是多样的，一是这类题目要求学生有深厚的计算几何知识，而这些知识平时应用很少，不仅中学生很少接触到，甚至普通的大学计算机专业学生也不会去研究；二是要求学生有很强的数学功底（如高等数学）和良好的空间思维能力，对计算几何中一些经典的算法熟练掌握，综合应用，精度要求也很高，编程量往往也很大；三是这类题目的每个测试点往往包含多组测试数据，造成时间不够；再者，选手遇到这类题目，有时会因为理解和思维上的困难而无法下手，导致得零分；有时又会因为考虑问题不够全面导致一些致命的错误。

但是，一旦你熟练掌握了计算几何中的一些基本问题和经典算法，那么，它解决问题往往是最高效的。因为：数学工具越高级，解决问题就越高效。

信息学竞赛中的计算几何题有一些不同于一般几何题的、很明显的特点。它们不会是证明题，依靠计算机进行的证明是极为罕见的。计算机擅长的是高速运算，所以这类题目中一般都有一个（或多个）解析几何中的坐标系，需要大量繁琐的、人力难以胜任的计算工作，这也许正是它们被称为计算几何的原因吧！计算几何题的几种常见问题类型有：

1、 计算求解题

解这一类题除了需要有扎实的解析几何的基础，还要全面地看待问题，仔细地分析题目中的特殊情况，比如求直线的斜率时，直线的斜率为无穷大；求两条直线的交点时，这两条直线平行等等。这些都是要靠平时学习时的积累。

2、 存在性问题

这一类问题可以用计算的方法来直接求解，如果求得了可行解，则说明是存在的，否则就是不存在的，但是模型的效率同模型的抽象化程度有关，模型的抽象化程度越高，它的效率也就越高。几何模型的的抽象化程度是非常低的，而且存在性问题一般在一个测试点上有好几组测试数据，几何模型的效率显然是远远不能满足要求的，这就需要对几何模型进行一定的变换，转换成高效率的模型。

3、 最佳值问题

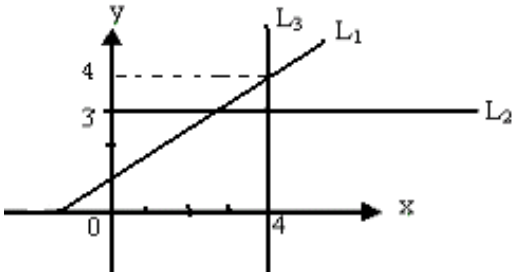
这类问题是计算几何题中比较难的问题，一般没有什么非常有效的算法能够求得最佳解，最常用的是用近似算法去逼近最佳解，近似算法的优劣也完全取决于得出的解与最优解的近似程度。

二、计算几何的基础——矢量

矢量分析是高等数学的一个分支，主要应用于物理学（如力学分析）。在一些计算几何问题中，矢量和矢量运算的一些独特的性质往往能发挥出十分突出的作用，使问题的求解过程变得简洁而高效。熟练掌握一些矢量分析的方法，并灵活地加以运用，就能轻松地解决许多看似复杂的计算几何题，或者会对我们解这类题目有很大帮助，甚至还有一些计算几何题是非用矢量方法不能解决的。

1、直线

①　直线的方程：



一般形式为：ax+by+c=0，或 y=kx+b。k 称为直线的斜率，b 称为截矩。

如上图中，直线 L_1 的方程为：x-y+1=0，L_2：y=3，L_3：x=4。

特例：若 a≠0，则一般方程可为：x+by+c=0；若 b≠0，则一般方程可为：ax+y+c=0。

②　直线的斜率：

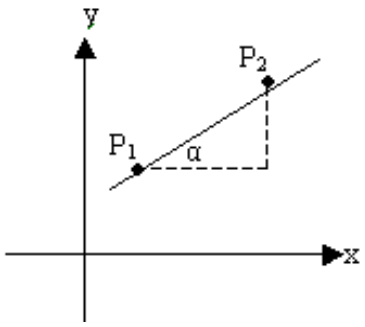
如下图：过两点 P_1，P_2 可以决定一条直线。斜率为：$k=\dfrac{y_2-y_1}{x_2-x_1}=\tan\alpha$。

如上图中，直线 L_1 的斜率为 1。

特别地：当 $y_1=y_2$ 时，斜率 k=0(如上图中的 L_2)，

当 $x_1=x_2$ 时，k 不存在(如上图中的 L_3)。

而当 x_1 与 x_2 无限接近时，斜率 k 趋于无穷大，这在编程时要特别小心。



③　两条直线垂直，则它们的斜率乘积等于-1。

2、线段

① 凸组合：两个不同的点 P1(x1,y1)、P2(x2,y2)的凸组合是满足下列条件的点 P3(x3,y3)：
对某个 0≤σ≤1，有 x3=σx1+(1-σ)x2,y3=σy1+(1-σ)y2
一般也可以写成：P3=σP1+(1-σ)P2
直观上看，P3 通过直线 P1P2，并且处于 P1 和 P2 之间（也包括 P1，P2 两点）的任意点。

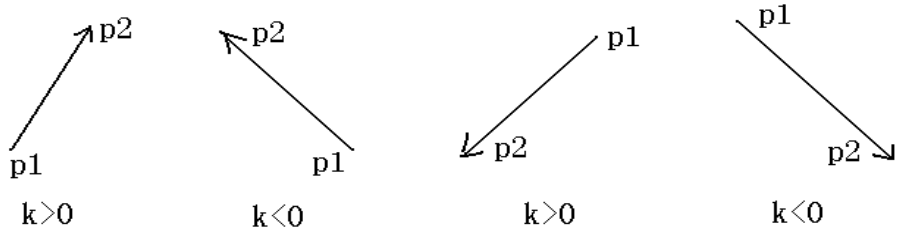
② 线段：线段 $\overline{P1P2}$ 是两个相异点 P1，P2 的凸组合的集合，其中 P1，P2 称为线段的端点

3、向量（矢量）的概念

① 矢量：有方向的线段，即 P1 和 P2 的顺序是有关系的，记为：$\overrightarrow{P1P2}$

如果 P1 是坐标原点，则 $\overrightarrow{P1P2}$ 又称为向量 P2，如下面的矢量示意图。

② 矢量的斜率：既然矢量是有方向的，那么矢量的斜率 k 就是有正负之分的，具体如下：



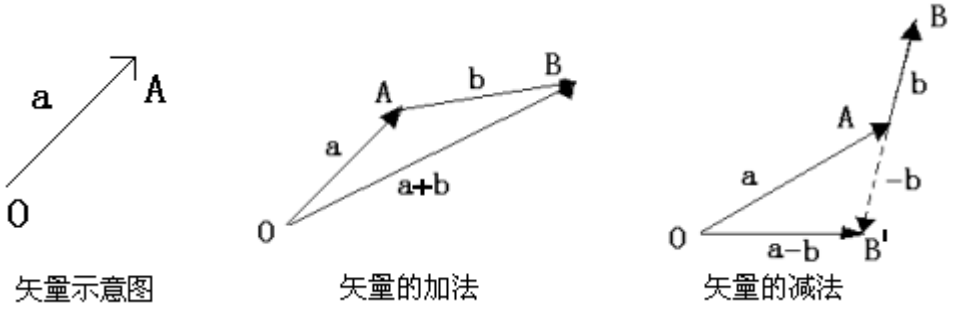
③ 设 $\overrightarrow{OA}$=a，则有向线段 $\overrightarrow{OA}$ 的长度叫做向量（矢量）a 的长度或模。记作|a|。

④ 夹角：两个非 0 矢量 a、b，在空间任取一点 *O*，作 $\overrightarrow{OA}$=a，$\overrightarrow{OB}$=b，则角∠AOB 叫做矢量 a 与 b 的夹角，记作<a，b>。若<a，b>=π/2，则称 a 与 b 互相垂直，记作 a⊥b。

4、矢量的加减法

以点 O 为起点、A 为端点作矢量 a，以点 A 为起点、B 为端点作矢量 b，则以点 O 为起点、B 为端点的矢量称为 a 与 b 的和 a+b，如下中图。

从 A 点作 $\overrightarrow{AB'}$，要求 $\overrightarrow{AB'}$ 的模等于|b|，方向与 b 相反，即 $\overrightarrow{AB'}$=-b，则以 O 为起点、B’为端点的矢量称为 a 与 b 的差 a-b，如下右图。



5、矢量的分解

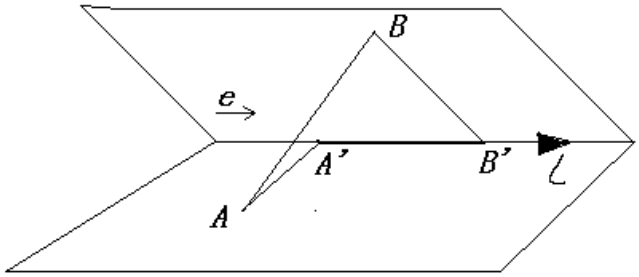
定理：如果空间三个矢量 a,b,c 不共面，那么对任一矢量 p，一定存在一个且仅一个有序实数组 x,y,z，使得：p=xa+yb+zc。

含义与物理上的合力和力的分解一样。

6、射影

已知向量 $\overrightarrow{AB}$ =a 和轴 *l*，e 是 *l* 上与 *l* 同方向的单位向量，作点 A 在 *l* 上的射影 A’，作点 B 在 *l* 上的射影 B’，则 $\overrightarrow{A'B'}$ 叫做向量 $\overrightarrow{AB}$ 在轴 *l* 上或在 e 方向上的（正）射影。

可以证明：$A'B'=|\overrightarrow{AB}|\cos\langle a,e\rangle$



7、矢量的数量积（点乘）

两个矢量的数量积是一个数，大小等于这两个矢量的模的乘积再乘以它们夹角的余弦。

$$a\cdot b=|a||b|\cos\langle a,b\rangle$$

用上面讲到的矢量的分解可以证明，数量积等于两个矢量的对应支量乘积之和。

$$a\cdot b=a_xb_x+a_yb_y+a_zb_z$$

数量积的性质：

① $a\cdot e=|a||e|\cos\langle a,e\rangle=|a|\cos\langle a,e\rangle$

② $a\perp b$ 等价于 $a\cdot b=0$，即 $a_xb_x+a_yb_y+a_zb_z=0$

③ 自乘：$|a|^2=a\cdot a$

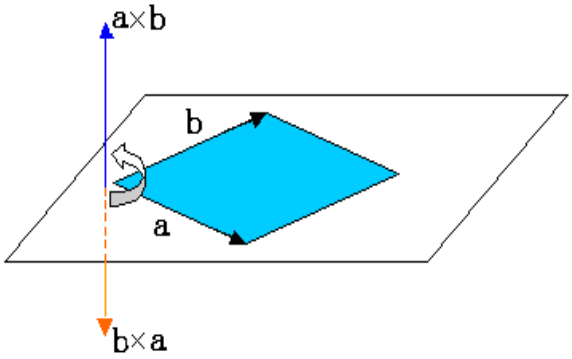
④ 结合律：$(\lambda\cdot a)\cdot b=\lambda(a\cdot b)$

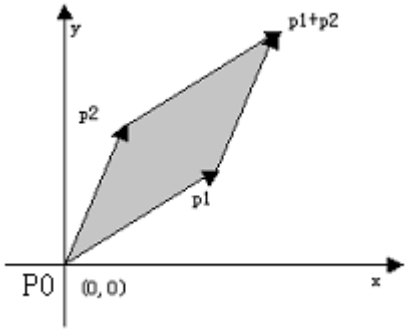
⑤ 交换律：$a\cdot b=b\cdot a$

⑥ 分配律：$a\cdot(b+c)=a\cdot b+a\cdot c$

8、矢量的矢量积（叉乘、叉积）

① 矢量积的一般含义：两个矢量 **a** 和 b 的矢量积是一个矢量，记作 a×b，其模等于由 **a** 和 **b** 作成的平行四边形的面积，方向与平行四边形所在平面垂直，当站在这个方向观察时，**a** 逆时针转过一个小于 π 的角到达 **b** 的方向。这个方向也可以用物理上的右手螺旋定则判断：右手四指弯向由 **A** 转到 **B** 的方向（转过的角小于 π），拇指指向的就是矢量积的方向。如下图（左）。





② 我们给出叉积的等价而更有用的定义，把叉积定义为一个矩阵的行列式：

$$p_1 \times p_2 = \det\begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix}$$
$$= x_1 y_2 - x_2 y_1$$
$$= -p_2 \times p_1$$

如上右图，如果 $p_1 \times p_2$ 为正数，则相对原点(0,0)来说，p_1 在 p_2 的顺时针方向；如果 $p_1 \times p_2$ 为负数，则 p_1 在 p_2 的逆时针方向。如果 $p_1 \times p_2$=0，则 p_1 和 p_2 模相等且共线，方向相同或相反。

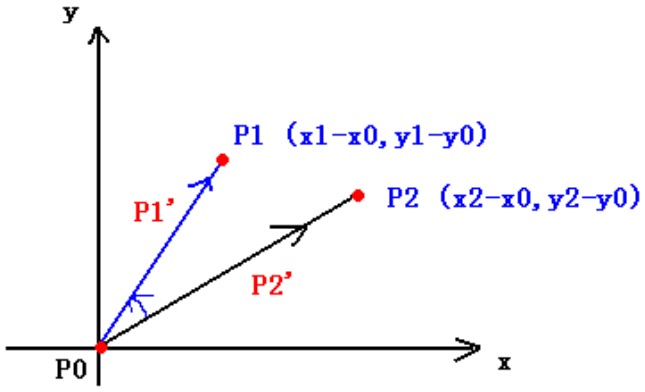
也可以用矢量的分解证明：

$$A \times B = \begin{vmatrix} i & j & k \\ Ax & Ay & Az \\ Bx & By & Bz \end{vmatrix} = |i| \begin{vmatrix} Ay & Az \\ By & Bz \end{vmatrix} - |j| \begin{vmatrix} Ax & Az \\ Bx & Bz \end{vmatrix} + |k| \begin{vmatrix} Ax & Ay \\ Bx & By \end{vmatrix}$$

即：
$$(A \times B)x = AyBz - AzBy$$
$$(A \times B)y = AzBx - AxBz$$
$$(A \times B)z = AxBy - AyBx$$

注：i, j, k 分别为 x, y, z 方向上的单位矢量

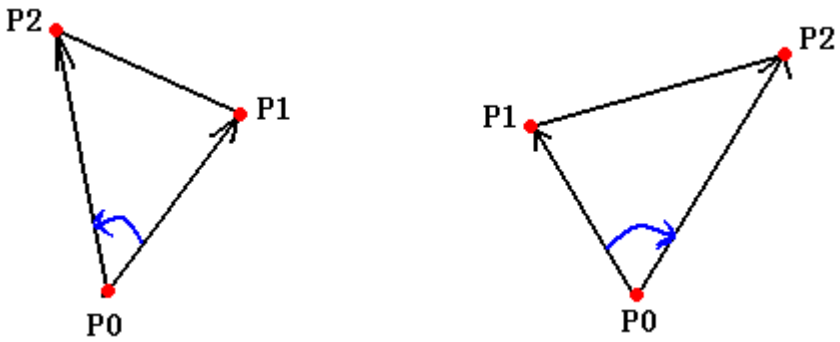
③ 现在探讨一个重要的问题：给定两个矢量：$\overrightarrow{P_0P_1}$ 和 $\overrightarrow{P_0P_2}$，对它们的公共端点 P0 来说，判断 $\overrightarrow{P_0P_1}$ 是否在 $\overrightarrow{P_0P2}$ 的顺时针方向。



方法：如上图，把 p_0 作为原点，得出向量 P1'=P1-P0 和 P2'=P2-P0，因此，这两个向量的叉积为：$(p_1-p_0)\times(p_2-p_0)=(x_1-x_0)(y_2-y_0)-(x_2-x_0)(y_1-y_0)$，如果该叉积为正，则 $\overrightarrow{P_0P_1}$ 在 $\overrightarrow{P_0P_2}$ 的顺时针方向，如果为负，则 $\overrightarrow{P_0P_1}$ 在 $\overrightarrow{P_0P_2}$ 的逆时针方向。如果等于 0，则 P0，P1，P2 三点共线。

④　讨论另一个重要问题：确定连续线段是向左转还是向右转，如下图，即两条连续线段 $\overrightarrow{P_0P_1}$ 和 $\overrightarrow{P1P_2}$ 在点 P1 是向左转还是向右转。也即∠$P_1P_0P_2$ 的转向。

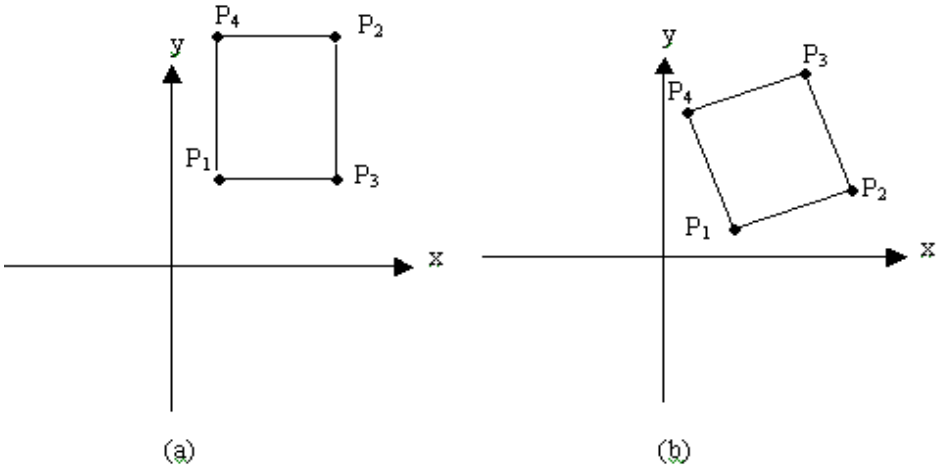
方法：叉积，同上。



⑤　矢量的叉积对于计算几何有着重要的意义，是很多算法的核心。用叉积可以判断从一个矢量到另一个矢量的旋转方向，可以求同时垂直于两个矢量的直线（矢量）方向，还能用来计算面积……在计算几何中大有用武之地，下面的例子中有更详尽的说明。

矢量的旋转：实际应用中，经常需要从一个矢量转到另一个矢量，这个过程称为矢量的旋转，旋转的方向由叉积判定。很多例子都用到矢量的旋转这个方法和相应特性。

9、坐标系中的矩形



若矩形边平行坐标轴，如(a)图，此时给出 2 个点的坐标：$P_1(x_1, y_1)$，$P_2(x_2, y_2)$，则其它两点坐

标为：$P_3(x_2, y_1)$， $P_4(x_1, y_2)$；

若矩形边不平行坐标轴，如(b)图，此时给出三点坐标：$P_1(x_1, y_1)$， $P_2(x_2, y_2)$， $P_3(x_3, y_3)$，则第 4 点坐标可以推得，利用切割法可以得到：

$$\begin{cases} x_4 + x_2 = x_1 + x_3 \\ y_4 + y_2 = y_1 + y_3 \end{cases} \Longrightarrow \begin{cases} x_4 = x_1 + x_3 - x_2 \\ y_4 = y_1 + y_3 - y_2 \end{cases}$$

10、圆

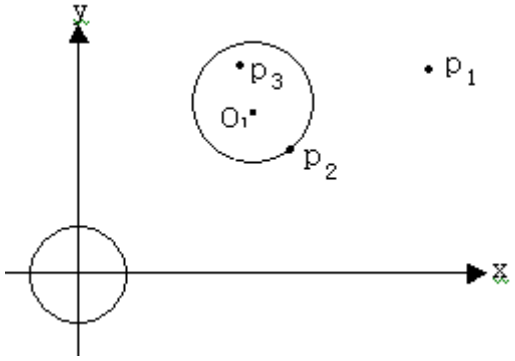
设圆心的坐标为(x_0, y_0) ，半径为 r。则圆的一般方程为：$(x-x_0)^2+(y-y_0)^2=r^2$

若 $O_1(x_0, y_0)$为原点（0，0），则方程为：$x^2+y^2=r^2$，点与圆的关系：若点为 $P(x_1, y_1)$，代入方程：

若：$(x_1-x_0)^2+(y_1-y_0)^2>r^2$ ，则 $P(x_1, y_1)$在圆外；

若：$(x_1-x_0)^2+(y_1-y_0)^2=r^2$ ，则 $P(x_1, y_1)$在圆上；

若：$(x_1-x_0)^2+(y_1-y_0)^2<r^2$ ，则 $P(x_1, y_1)$在圆内；



例 1、任给三条直线的方程(斜率表示法，a1,c1,a2,c2,a3,c3)，求它们所围成的三角形的面积。

算法分析

很容易，过程如下：

① 设 3 条直线的方程分别为：y = a1 * x + c1; y = a2 * x + c2; y = a3 * x + c3;

② 首先要判断这三条直线是否能构成三角形，若能则求出三个交点，再由交点求出三边的长度，再用海伦公式求出三角形的面积: s = sqrt(p*(p-a)*(p-b)*(p-c))　(p = (a+b+c)/2);

③ 构成三角形的条件是：三条直线的斜率互不相等，即 a1,a2,a3 互不相等；

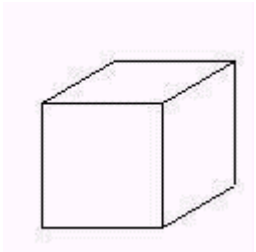
④ 求两条直线的交点坐标，方法是求两个直线方程所列成的二元一次方程组的解，如:y = a * x + b, y = c * x + d，交点坐标为 ((d-b)/(a-c),a*(d-b)/(a-c)+b);

⑤ 由两个点的坐标求两点之间距离只要用“两点间直线距离的公式”即可，如（a,b)、(c,d）两点间的距离为 s=sqrt(sqr(a-c)+sqr(b-d));

例 2、蚂蚁搬家

问题描述：

右图所示为一单位正方体（即边长为 1 的正方体），有一只蚂蚁要从上表面上的一点出发爬往下表面的某一点，规定蚂蚁只能沿正方体表面爬行，要求编程求出一条从起点到终点的最短距离。起点和终点坐标从键盘输入，设定上下两个表面的坐标原点均为正方形的中心，且两个表面的坐标系的 X 轴和 Y 轴方向一致，输出时保留三位小数。



输入格式：4 个实数（-0.5 到 0.5 之间），分别表示出发点和目的点的坐标。如：

Input x1,y1,x2,y2：0.26　0.50　0.50　0.18

输出格式：

The shortest distance is: 1.146

例 3、判断点与多边形中的位置

问题描述：

已知点 P 的 X、Y 坐标和一个 N 边形 $A_1A_2\cdots A_n$。判断点 P 在 N 边形的内部,外部或边上。

输入：

第 1 行输入 N；

第 2 行至第 N+1 行输入 N 边形各点的 X、Y 坐标；

第 N+2 行输入 P 点 X、Y 坐标；

输出：

如果在内部，输出"NEIBU"

如果在外部，输出"WAIBU"

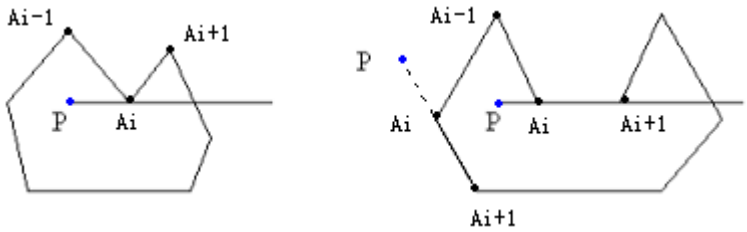
如果在边上，输出"BIAN"

算法分析：

判断点 P 是否在多边形的边上比较容易，可以逐一对每条边进行检查，判断 P 是否在多边形的边上。关键问题在于解决点 P 在多边形的内部还是外部。

首先过点 P 做一条足够长的射线，如果这条射线与多边形的交点个数为奇数，那么点 P 在多边形的内部，如果交点为偶数（包括 0 个），那么点 P 在多边形的外部。

当然还要对一些特殊情况进行处理。如以下 2 种情况：



左图中点 P 与多边形其中一点相交，右图中点 P 与多边形的一边重合。对这 2 种特殊情况往往有以下 2 种处理方法：

方法 1、 再任意取一条射线，判断点 P 与多边形的交点个数，直到不出现特殊情况为止。

方法 2、 对 2 种情况另作处理：

（1） 交点在Ai点上：

a、 如果Ai-1、 Ai+1在射线的同侧，则***这点不算***。

b、 如果Ai-1、 Ai+1在射线的两侧，则这点要算。

（2） 和多边形一条边Ai、Ai+1重合：

再反向作一条射线，找交点情况。

在实际情况中，出现特殊情况的可能性很小，因此用第一种方法比较简便，可以使程序比较简洁。

从以上分析，得出下列算法：（下列采用第 1 种方法）——随机化

（1） 输入数据。

（2） 逐一判断点 P 是否在多边形的边上，如果是，则输出"bian"，退出。

（3） 随机产生一点 Q，作出射线 PQ。直到射线 PQ 不与多边形的点和任一边不相交为止。

（4） 计算射线 PQ 与多边形边的交点数，如果交点数为奇数，则点 P 在多边形的内部(in)，否则点 P 在多边形的外部(out)。

程序：略。

例 4、无线电测向

问题描述：

一艘有天线定位装置的船能通过接收当地灯塔信号来确定自己的位置。每个灯塔固定在已知点上并发出特有的信号。当船检测到信号，它可通过旋转天线直到信号达到最大强度。这样就可确定自身与该灯塔的位置关系。只要接收到两个灯塔的信息，就有可能确定船当前的位置。

编程任务：通过一对灯塔信息来确定船的位置。

灯塔和船的位置被确定在一个直角坐标系内。X 轴正向指向东，Y 轴正向指向北。船的航行路线从正北开始按顺时针用度表示。北是 0°，东是 90°，南是 180°，西是 270°。灯塔与船的位置关系用相对于船的航行方向顺时针用度表示。

数据输入：

输入数据由文件名为 INPUT.TXT 的文本文件提供。文件的第一行是一个整数，表示灯塔的数目 N(N<=30)。以下 N 行,每行表示一个灯塔,为灯塔名称（名称是 20 个以下的字母），X 坐标和 Y

坐标。它们都用空格隔开。

灯塔信息下面是船的信息包括三行，一行是船的方向，其余两行是所接收到的灯塔信号。

具体如下：

输入数据	数据的含义
方向	船的航行方向；
名称 1　角度 1	第一个灯塔信息的名称，灯塔的方位；
名称 2　角度 2	第二个灯塔信息的名称，灯塔的方位。

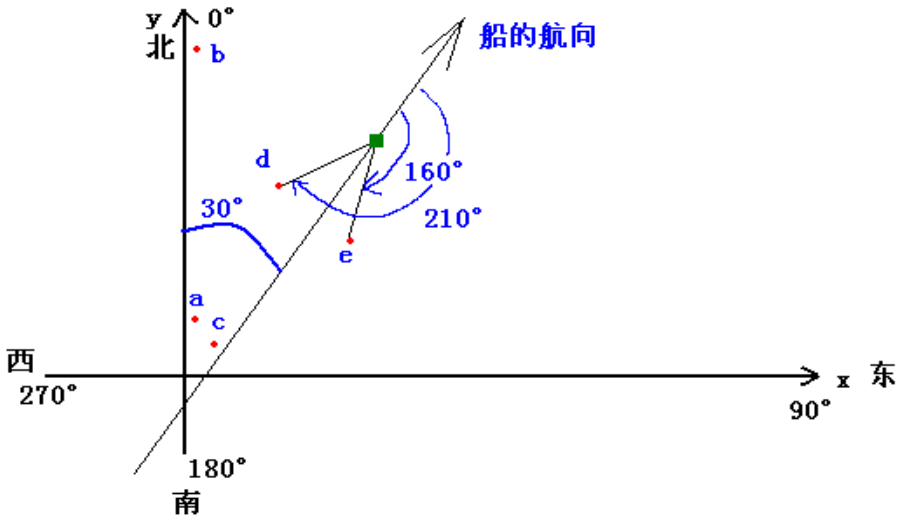
灯塔的方位为船与灯塔所在的直线与船的航行方向的夹角（从船的航行方向开始顺时针，角度 1=角度 1+180）。2 个数据用空格隔开。

数据输出：

将船的位置（精确到 2 位小数）。输出到 OUTPUT.TXT 文件中。如果无法确定船的位置，应输出"NO ANSWER"（不能使用小写）。

输入样例：

5
a 1 5
b 1 1000
c 2 4
d 51 60
e 153 79
30
e 160
d 210



输出样例：

160.83 123.41

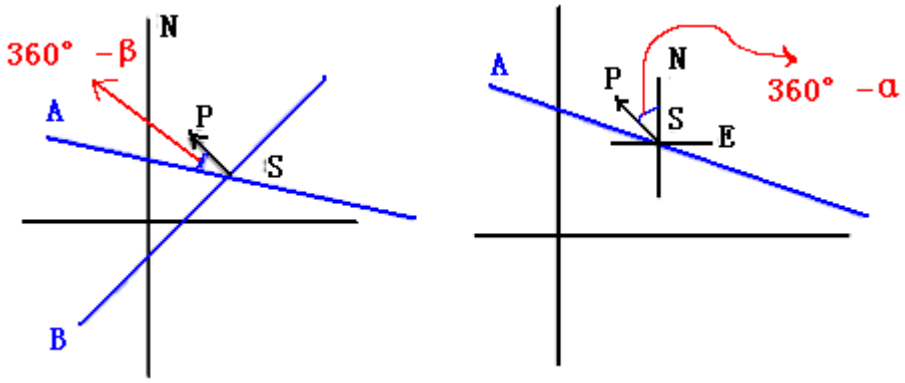
算法分析：

我们在解题前一共得到了这些信息：各灯塔所在位置的坐标、船的航行方向、船航行方向和船与其中两个灯塔所在直线的夹角。我们能否先求出船与灯塔所在直线的倾斜角，由于我们又知道该灯塔的坐标，于是就可以得到该直线的点斜式方程；再通过解直线方程组，得到的两直线交点坐标即为当前船所在的位置。

于是本题的难点就在于如何求船与灯塔所在直线的倾斜角。

如下图，我们假设船位于 S，接收到来自灯塔 A，B 的信号，而船的航行方向是有向线段 SP。可想而知，直线的倾斜角一定与船的航向以及该直线与船航向的夹角有关。以直线 AS 为例，由于船的航向 α 是从正北开始按顺时针用度表示，所以图中的∠NSP=360°－α；因为直线 AS 与船的航

行方向的夹角 β 也是从船的航行方向开始顺时针表示，所以，∠ASP=360°－β。那么，直线 AS 的倾斜角该如何表示呢？从图中看，∠AS E 即为直线 AS 的倾斜角，而∠ASE=（∠ASP +∠NSP)+ 90° =［（360°－α）+（360°－ β）］+ 90° ，对以上等式进行整理，直线 AS 的倾斜角= 90°－（α＋β ）+ 720°。 对于其它情况，结果是否还是这样呢？我们再对其它情况（ 例如有向线段 S P 在不同的象限 ，直线与航向的夹角也各不相同）进行类似的做法，发现存在一个普遍的结论：**直线倾斜角= 90°－（ α＋β）＋ K ＊ 360°，其中 K∈Z 。**



到此，问题已经解决一半了，接下来，我们该做的只是列出直线方程，并且解出它们的交点。

我们知道，直线斜率等于该直线倾斜角的正切值，由于 Tan 是周期为 180° 的周期函数，所以直线斜率 K=T a n[90°－（α＋β）]。设该直线对应灯塔坐标为（ X1，Y1 ），该直线方程就可以表示成 ：Y= K （X － X1)+Y1。因为我们知道两个灯塔的信息，用同样的方法我们也可以得到另外一个直线方程。联立方程，解出交点坐标就不困难了。

然而在解题时，我们还要注意一些问题：

1、在 TURBO PASCAL 中，没有三角函数 T a n ，但可以用 S i n 和 C o s 计算出来；

2、在 TURBO PASCAL 中，三角函数的定义域是用弧度表示的，因此在计算斜率之前，我们必须将角度转成弧度；

3、当直线斜率不存在（ 即直线倾斜角等于 90°） 时，我们知道，当角度无限趋近于 90° 时，T a n 的取值就无限趋近于正无穷或负无穷，因此我们可以取一个很大或很小的数（ 如 1E20 等 ）来表示此时的 T a n 值，这不会影响到计算结果。

也许，解这道题的方法不止这一种，但无疑地，以上的解法是最便捷的。得到这种解法的关键就在于对题目所给出的信息的正确运用以及对出题者本意的准确把握。我们在审题时应该注意到本题给出的是两条不定直线和一些关于角度的信息，而待求点即为这两条直线的交点。如果再仔细地考虑一下，我们会发现确定这两条直线是达到目标的必经途径。于是解题的框架就大致地出来了。

三、计算几何的基本算法

任何复杂的算法都是由许多简单的算法组合而成的，计算几何题也同样如此，这些基本算法是求解计算几何题的基础，任何对基本算法的不熟悉，都可能导致解题的失败，所以熟悉这些算法是非常重要的。下面我们先来看看一些最基本的计算几何算法。

```
Const zero=1e-6;
type pointtype=record
                    x,y:extended;
               end;
```

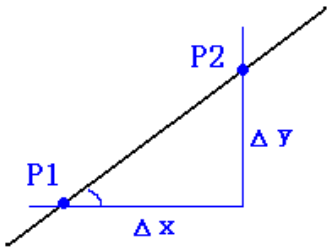
1、 两点间的线段长度，P1(x1,y1)，P2(x2,y2)

公式：$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

```
function linelength(x1,y1,x2,y2:extended):extended;
 begin
    linelength:=sqrt(sqr(x1-x2)+sqr(y1-y2));
 end;
```

2、 已知 2 点 P_1（X_1,Y_1），P_2（X_2,Y_2），求直线 P_1P_2 的斜率

```
function k(x1,y1,x2,y2: extended): extended;
  begin
    if abs(x1-x2)<=zero  then writeln( 'not exist!' )
                         else k:=(y2-y1)/(x2-x1);
  end;
```



3、 求过 P1(x1,y1)，P2(x2,y2)的直线方程 ax+by+c=0

有以下的 3 种情况：

① $x_1=x_2$，不存在 k，此时方程为：$x=x_1$

② $y_1=y_2$，k=0，此时方程为：$y=y_1$

③ 一般地，若 $x_1\neq x_2$ 且 $y_1\neq y_2$，由方程一般形式 x+by+c=0 代入：

$$\begin{cases} x_1+by_1+c=0 \\ x_2+by_2+c=0 \end{cases} \Longrightarrow \text{从中解出} \begin{cases} b=(x_2-x_1)/(y_2-y_1) \\ c=-(x_2-x_1)y_1/(y_2-y_1)-x_1 \end{cases}$$

或者通过公式：$\dfrac{y-y_1}{y_2-y_1}=\dfrac{x-x_1}{x_2-x_1}$

```
procedure getline(x1,y1,x2,y2:extended;var a,b,c:extended);
   begin
     if abs(x1-x2)<=zero
       then begin a:=1;b:=0;c:=-x1 end
       else if (y1-y2)<=zero
              then begin  a:=0;b:=1;c:=-y1 end
```

```
            else begin  a:=1;                {或 a:=y1-y2}
                        b:=(x2-x1)/(y2-y1);{b:=x1-x2}
                        c:=(x1-x2)*y1/(y2-y1)-x1;{c:=x1*y2+x2*y1}
                 end;
    end;
```

4、 已知两条不平行的直线 P_1P_2 ，P_3P_4，求交点 P5

首先判断这两条直线是否平行或重合，再解下面的二元一次方程：$\begin{cases} x+b_1y+c_1=0 \\ x+b_2y+c_2=0 \end{cases}$

利用上面的 3 讲到的结论，过程如下：

```
procedure GetJiao(p1,p2,p3,p4:pointtype; var p5:pointtype);
  var a1,b1,c1,a2,b2,c2,d:extended;
  begin
    getline(p1.x,p1.y,p2.x,p2.y,a1,b1,c1); {求直线方程}
    getline(p3.x,p3.y,p4.x,p4.y,a2,b2,c2);
    if (a1=0) and (a2=0) or (b1=0) and (b2=0) then exit; {两个特例}
                                {表示两条直线分别平行于 X 轴或 Y 轴}
    if (b1<>0) and (b2<>0) and (a1/b1=a2/b2) then exit;
                                {表示两条直线斜率相等，即平行}
       a1:=p1.y-p2.y; b1:=p2.x-p1.x; c1:=p1.x*p2.y-p2.x*p1.y;
       a2:=p3.y-p4.y; b2:=p4.x-p3.x; c2:=p3.x*p4.y-p4.x*p3.y;
       d:=a1*b2-a2*b1;
       p5.x:=(b1*c2-b2*c1)/d;    {交点坐标}
       p5.y:=(c1*a2-c2*a1)/d;
  end;
```

5、 判断两条线段 P_1P_2 ，P_3P_4 是否相交

方法 1：只要在 4 的基础上，加上一个判断，看交点是否在线段 P_1P_2 ，P_3P_4 上，即：

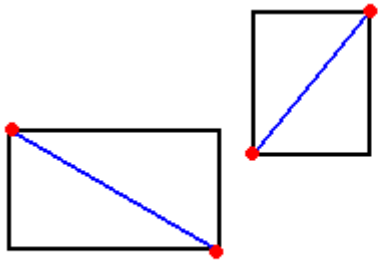
```
function judgejiao(var p1,p2,p3,p4):Boolean;
  var p:pointtype;
  begin
    getjiao(p1,p2,p3,p4,p);    {求交点}
    if min(p1.x,p2.x)<=p.x<=max(p1.x,p2.x) and min(p1.y,p2.y)<=p.y<=max(p1.y,p2.y)
      and min(p3.x,p4.x)<=p.x<=max(p3.x,p4.x) and min(p3.y,p4.y)<=p.y<=max(p3.y,p4.y)
    then  jiao:=true   {判断在线段内}
    else  jiao:=false;
  end;
```

问题是：运用了除法，使得效率很低，并且容易产生误差，当线段几乎平行时，该算法对实际计算机除法运算的精确度非常敏感。所以，希望在这种题目中只用加、减、乘、比较运算，

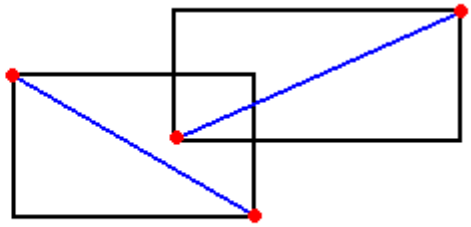
不要用除法，也不要用三角函数。

方法 2：用叉积去做，分两步：

第 1 步：快速排斥试验，如果分别以 P_1P_2 ，P_3P_4 为对角线做矩形，而这两个矩形不相交，则这两条线段肯定不相交，如下左图；即使两个矩形相交，这两条线段也不一定相交，如下右图，这时再用第 2 步判断；



矩形不相交



矩形相交，但线段不跨立

表示成语句，即两个矩形相交当且仅当下列式子为真：

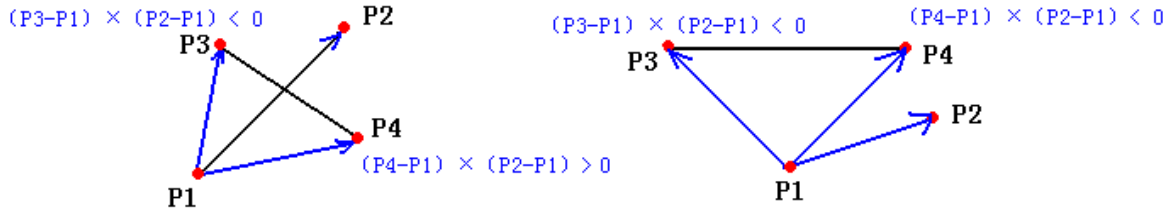
（max(x1,x2)≥min(x3,x4)）∧（max(x3,x4)≥min(x1,x2)）∧（max(y1,y2)≥min(y3,y4)）∧（max(y3,y4)≥min(y1,y2)）

两个矩形相交必须在两个方向上都相交，式子的前半部分判断在 x 方向上是否相交，后半部分判断在 y 方向上是否相交。

第 2 步：确定每条线段是否“跨立”另一条线段所在的直线。

跨立：如果点 P1 处于直线 P3P4 的一边，而 P2 处于该直线的另一边，则我们说线段 $\overline{p1p2}$ 跨立直线 P3P4，如果 P1 或 P2 在直线 P3P4 上，也算跨立。

两条线段相交当且仅当它们能够通过第 1 步的快速排斥试验，并且每一条线段都跨立另一条线段所在的直线。

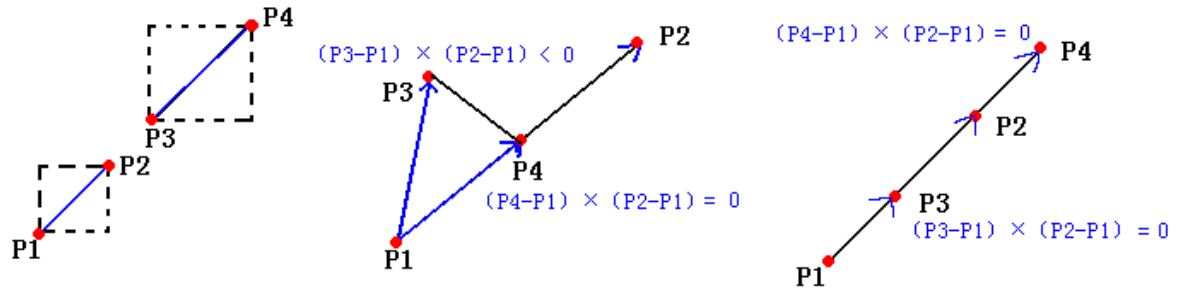


跨立试验

具体第 2 步的实现，只要用叉积去做就可以了，即只要判断矢量 $\overline{p1p3}$ 和 $\overline{p1p4}$ 是否在 $\overline{p1p2}$ 的两边相对的位置上，如果这样，则线段 $\overline{p1p2}$ 跨立直线 P3P4。也即检查叉积（P3-P1）×（P2-P1）与（P4-P1）×（P2-P1）的符号是否相同，相同则不跨立，线段也就不相交，否则相交。

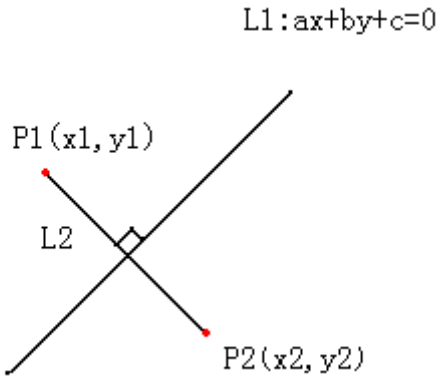
当然也有一些特殊情况需要处理，如任何一个叉积为 0，则 P3 或 P4 在直线 P1P2 上，又因为通过了快速排斥试验，所以下图左边的情况是不可能出现的，只会出现右边的两种情况。当然，

还会出现一条或两条线段的长度为 0，如果两条线段的长度都是 0，则只要通过快速排斥试验就能确定；如果仅有一条线段的长度为 0，如 $\overline{p3p4}$ 的长度为 0，则线段相交当且仅当叉积（P3-P1）×（P2-P1）。



几种跨立时的特例

6、 已知直线 ax+by+c，求点 P1(x1,y1)关于此直线的对称点 P2(x2,y2)

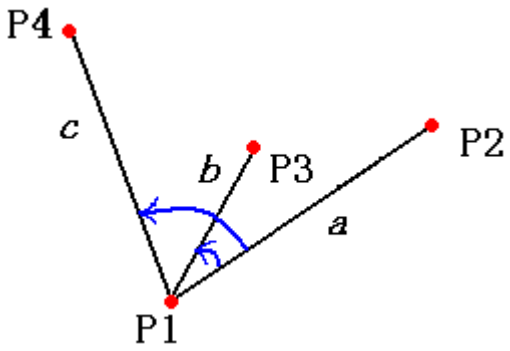


首先线段 P_1P_2 的中点在 L1 上，所以得出方程 1：a*(x1+x2)/2+b(y1+y2)/2+c=0

对于 L1，y=-ax/b-c/b，所以 k1=-a/b，对于 L2，k2=(y2-y1)/(x2-x1) ，再利用 L1 和 L2 的斜率乘积等于-1，得出方程 2：a(y2-y1)-b(x2-x1)=0。，对于方程 1 和方程 2 利用消元法，解方程组即可求出 x2 和 y2。过程如下：

```
procedure dcd(a,b,c,x1,y1:extended;var x2,y2:extended);
  begin
    x2:=((b*b-a*a)*x1-2*a*b*y1-2*a*c)/(a*a+b*b);
    y2:=((a*a-b*b)*y1-2*a*b*x1-2*b*c)/(a*a+b*b);
  end;
```

7、 判断两点 P_3 和 P_4 是否在直线 P_1P_2 的异侧



用叉乘和矢量的旋转，如果 a×b 和 a×c 的方向一致，则 P_3 和 P_4 是在 a 的同侧，否则在异侧。即：

t1=（a×c）$z = a_x c_y - a_y c_x$

$= (p2.x - p1.x)(p4.y - p1.y) - (p2.y - p1.y)(p4.x - p1.x)$

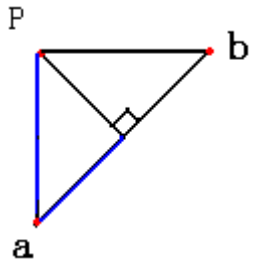
t2=（a×b）$z = a_x b_y - a_y b_x$

$= (p2.x - p1.x)(p3.y - p1.y) - (p2.y - p1.y)(p3.x - p1.x)$

如果 t1*t2>0，则在同侧，否则在异侧。

```
function Span(a,b,c,d:pointtype):boolean;{ 异侧返回 true，同侧返回 false}
   var x1,y1,x2,y2,t1,t2:extended;
   begin
     x1:=b.x-a.x;y1:=b.y-a.y;x2:=c.x-a.x;y2:=c.y-a.y;
     t1:=x1*y2-x2*y1;
     x2:=d.x-a.x;y2:=d.y-a.y;
     t2:=x1*y2-x2*y1;
     if t1*t2>0 then Span:=false else span:=true;
   end;
```

8、 求点 p 到直线 ab 的距离



公式： $d = \dfrac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$

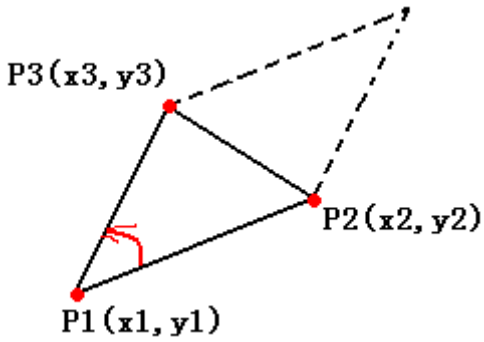
```
function Distance(a,b,p:pointtype):extended;
   var x1,y1,x2,y2,d:extended;
   begin
     x1:=b.x-a.x;y1:=b.y-a.y;
     x2:=p.x-a.x;y2:=p.y-a.y;
     distance:=abs(x1*y2-x2*y1)/sqrt(sqr(x1)+sqr(y1));
   end;
```

9、 已知 3 点 P_1（X_1, Y_1），P_2（X_2, Y_2），P_3（X_3, Y_3），求三角形 $P_1P_2P_3$ 的面积

方法 1：用海伦公式 $\sqrt{p(p-a)(p-b)(p-c)}$，其中，p=(a+b+c)/2，程序如下：

```
function area1(x1,y1,x2,y2,x3,y3:extended):extended;
  var p,a,b,c:extended;
  begin
     a:=linelength(x1,y1,x2,y2);{调用前面的函数 1}
     b:=linelength (x2,y2,x3,y3);
     c:=linelength (x3,y3,x1,y1);
     p:=0.5*(a+b+c);
     area1:=sqrt(p*(p-a)*(p-b)*(p-c));
  end;
```

方法 2：矢量的叉积



因为：$|\overrightarrow{P1P2} \times \overrightarrow{P1P3}| = S_{\text{平行四边形}}$

所以：$S_{\text{三角形}} = |\overrightarrow{P1P2} \times \overrightarrow{P1P3}| \ / \ 2$

$$= \begin{vmatrix} i & j & k \\ x2-x1 & y2-y1 & 0 \\ x3-x1 & y3-y1 & 0 \end{vmatrix} \ / \ 2$$

$$= (|i| \begin{vmatrix} y2-y1 & 0 \\ y3-y1 & 0 \end{vmatrix} - |j| \begin{vmatrix} x2-x1 & 0 \\ x3-x1 & 0 \end{vmatrix} + |k| \begin{vmatrix} x2-x1 & y2-y1 \\ x3-x1 & y3-y1 \end{vmatrix}) \ / \ 2$$

= （0-0+（x2-x1）（y3-y1）-（x3-x1）（y2-y1））/ 2

= (x1y2+x2y3+x3y1-x1y3-x2y1-x3y2)/2

由于是面积，所以还要加上个绝对值。函数如下：

```
function sarea(x1,y1,x2,y2,x3,y3:real):real;
  begin
```

```
    sarea:=abs(x1*y2+x2*y3+x3*y1-x1*y3-x2*y1-x3*y2)/2;
  end;
```

注：其实，后面求 N 边形的面积也就是从这儿推广得到的。

四、经典算法

有了以上的基本算法还远远不够，因为光靠竞赛时的临时思考，组合算法从时间上来说是来不及的，这就需要熟悉一些经典的计算几何算法，在竞赛中直接使用，比如凸包问题就是。

1、 求平面凸包

问题描述 1：求覆盖平面上 n 个点的最小的凸多边形。也可以这样描述：

给定一个连接的多边形，可能是凸多边形，也有可能是凹多边形。现在，你的任务就是编程求这个多边形的最小凸包。如果它本身是凸多边形，那么最小凸包就是它本身。

数据范围：多边形顶点坐标 X，Y 是非负整数，不超过 512。

输入：

共有 K 组数据，每组测试数据的点都是按逆时针顺序输入的，没有 3 个点共线。

每组测试数据的第 1 行是 N，表示有 N 个点。以下 N 行，每行两个整数 X，Y。

输出：

输出格式与输入格式一样，第一行是 K，表示共有 K 组输出。以下 K 组数据：

每组的第一行为 M，表示该凸包上有 M 个顶点，以下 M 行每行两个整数 X，Y，表示凸包顶点的坐标。也按逆时针方向输出。

样例输入：

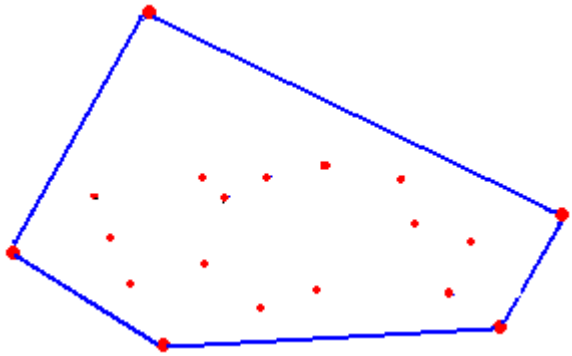
```
1
14
30 30
50 60
60 20
70 45
86 39
112 60
200 113
250 50
300 200
130 240
76 150
47 76
```

36 40
33 35

样例输出：
1
8
60 20
250 50
300 200
130 240
76 150
47 76
30 30
60 20

问题分析：

形象地描述一下，就是平面上有 n 根柱子，把一条封闭的弹性绳套上这些柱子，绳子绷紧后形成的多边形就是我们所求的凸包，如右图。



算法 1：一种容易想到的算法如下——不妨称之为斜率逼近法。

(1) 如上图，在所有的点中找出一个 $y=\min\{y_1, y_2, \cdots\cdots, y_n\}$ 的点，记为 P_1；

(2) 从 P_1 出发，首先在 $k>0$ 同时 $(x_2>x_1, y_2>y_1)$ 中找出最小的 k 的点 P_2

若没有，则再在 $k<0$ 内同时 $(x_2<x_1, y_2>y_1)$ 中找出最小的 k 的点 P_2，

若还没有，则再在 $k>0$ 同时 $(x_2<x_1, y_2<y_1)$ 中找出最小的 k 的点 P_2，

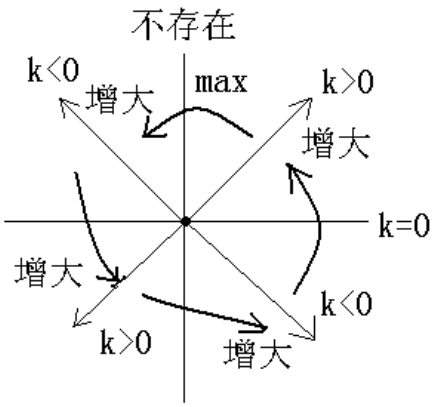
若还没有，则再在 $k<0$ 内同时 $(x_2>x_1, y_2<y_1)$ 中找出最小的 k 的点 P_2；

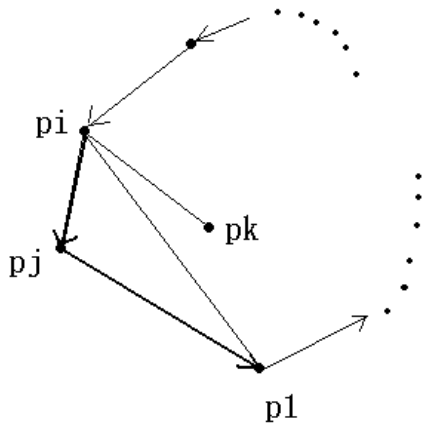
即：按下左图示意的顺序找 p2

注：如果在找 P2 时，存在多个点同时满足一个要求，则取距离 p1 最远的一个点作 P2。

(3) 从 P_2 出发，用第 2 步同样的方法找到点 P_3；

（4）依次类推，直到 $P_j=P_1$ 为止（如下右图，可以证明，一定能保证找到最后一个满足要求的 P_j，使得 $P_j=P_1$），形成的 $P_1P_2P_3\cdots\cdots P_j$ 即为所求的凸包。



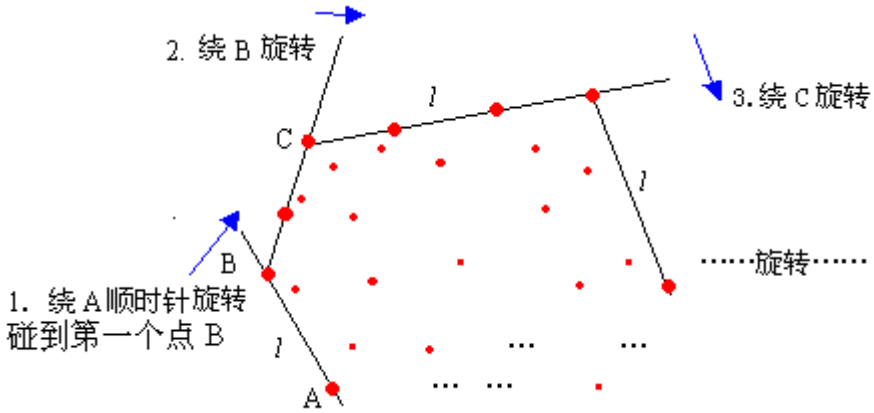


小结：

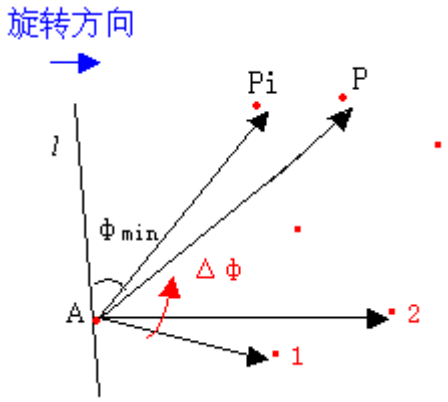
时间复杂度为 O（mn），m 表示凸包上的点数，时间主要花在求斜率上，如存在一些 $\overrightarrow{P_{i-1}P_i}$ 的斜率趋于无穷大，则如何解决呢？

算法 2：数学构造法——Jarvis 算法

形成凸包的数学构造法是这样描述的：如下图，找一条直线 *l* 过其中一点（记为 A）并且所有其他点都在 *l* 的同侧（显然这样的直线一定可以找到），则 A 必为凸包上的一点。让 *l* 以 A 为轴点向一个方向（如：顺时针方向）不断旋转，直至 *l* 碰到除 A 以外的第一个点（记为 B）。如果同时碰到多于一个点，则取与 A 点距离最大的。再以 B 为轴点，向相同的方向旋转 *l*，重复上述过程，直至 *l* 回到 A 点。开始时找出的 A 点和旋转过程中找出的 B、C 等点就构成了凸包的顶点序列。



我们无法在计算机上实现直线的旋转，因而不能直接套用数学构造法，但是可以通过对点的扫描达到同样的目的。其实，绕着 A 旋转 *l* 找到的是与 *l* 夹角最小的一条边，也是必然在凸包上的边。



以上图为例，轴点为 A，*l* 向顺时针方向旋转，目标点放在点变量 P 中。P 的初始值可以是除 A 以外任意一点。然后对每一个点 Pi，根据矢量的叉积和定向旋转，如果（$\overrightarrow{AP} \times \overrightarrow{AP_i}$）z>0，即 $\overrightarrow{AP}$ 转到 $\overrightarrow{AP_i}$ 为逆时针，$\overrightarrow{AP_i}$ 与 *l* 的夹角小于 AP 与 *l* 的夹角；或者（（$\overrightarrow{AP} \times \overrightarrow{AP_i}$）z=0）and（|APi| > |AP|），即 A，P，P_i 三点共线，P_i 离 A 点更远，则 P：=P_i。这样，找到一个逆时针方向的 Δ Φ，AP 就转过去。把所有的点都扫描一遍之后，AP 就转到了与 *l* 夹角最小的边上，也就是凸包上的一条边了，该过程记为 PROC1。

Jarvis 算法的核心部分就是上面的 PROC1，主要流程如下：

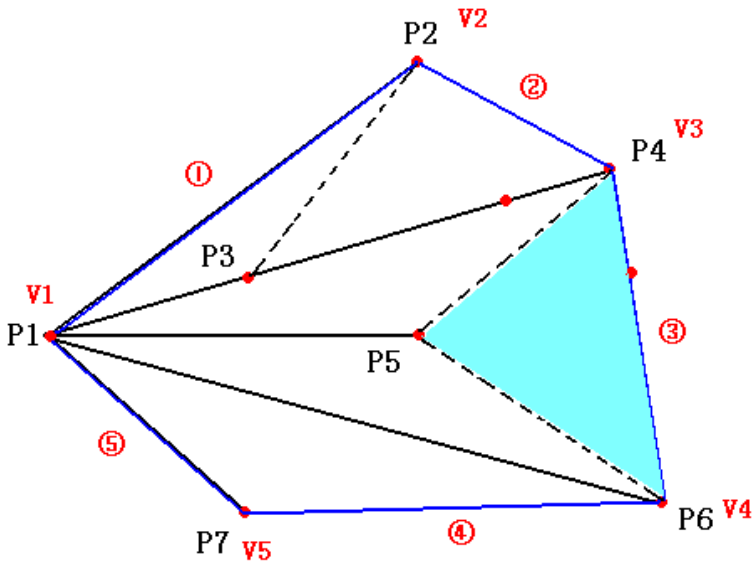
```
V0 := P1~Pn 中与某一点距离最大的点 {必然在凸包上的点}
Repeat
  PROC1
Until  目标点变量 P=V0。{重复次数等于凸包上点的个数}
```

小结：

矢量定向旋转的引入使算法变得十分简洁，编程复杂度很低。如果设凸包上点的个数为 m，则 Jarvis 算法的时间复杂度同样为 O（nm）。

算法 3：Graham 算法

如果 Jarvis 算法的时间效率不够高，我们还可以选择另一种更高效的算法——Graham 算法。这个算法在找凸包的边之前先作如下的预处理：首先，找一个凸包上的点（这一步与 Jarvis 算法相同），把这个点放到第一个点的位置 P_1。然后把 P_2~P_n 按照 P_1P_i 的方向排序，排序时元素的前后顺序由旋转方向（方向相同时由长度）决定，可以用矢量积判定。比如当（（$\overrightarrow{P_1P_i} \times \overrightarrow{P_1P_j}$）z>0）or（（$\overrightarrow{P_1P_i} \times \overrightarrow{P_1P_j}$）z=0）and（$|P_1P_i| > |P_1P_j|$）时，$P_i$ 在 P_j 之后（按顺时针方向排序）。预处理完成后，P 序列的点如果顺次连起来，可以连成一条不自交的封闭曲线，于是凸包顶点序列 V 就包含于 P 序列中了（V 序列的元素顺序与在 P 序列中的顺序相同）。剩下的问题就是：P 序列中哪些点才是凸包上的点呢？即应该跳过哪些点呢？



得出 Graham 算法的框架如下：

```
V1 := P1;   j := 1;
For I := 2 to n do {这里的 P 序列是按顺时针排序的，按顺序扫描每个点}
 begin
   While (I<n)and((V1Pj × V1Pi+1)z=0) do  I := I + 1;    {注 1}
   While (j>1)and(Vj-1Vj × VjPi)z>=0) do  J := j - 1;   {注 2}
   J := j + 1;  Vj := Pi      {Pi 进入 V 序列}
 End;
```

注 1：

当 $\overrightarrow{V_1P_j}$ 和 $\overrightarrow{V_1P_{i+1}}$ 方向相同时，P_i（与 V_1 距离较小）一定不是凸包的顶点，应跳过，如上图中的 P3。

注 2：

当 $\overrightarrow{V_{j-1}V_j}$ 到 $\overrightarrow{V_jP_i}$ 为逆时针或 0 角度旋转时，V_{j-1}，V_j，P_i 呈凹形，V_j 一定不是凸包的顶点，退出 V 序列，如上图中的 P5。

参考程序：program\Graham.pas

Graham 算法也可以通过设置一个关于侯选点的堆栈 S 来解决凸包问题，设输入点集 Q 中的每个点都被压入堆栈一次，不是凸包中的点最终将被弹出堆栈。当算法终止时，堆栈 S 中仅包含凸包中的顶点，其从下往上的顺序为各点在边界上出现的逆时针方向排列的顺序。

下列过程要求|Q|>=3,它调用函数 TOP(S)返回处于堆栈 S 顶部的点,并调用函数 NEXT-TO - TOP（S）返回处于堆栈顶部下面的那个点。

PROCEDURE Graham-Scan(Q)；

BEGIN

设 PO 是 Q 中 Y 坐标最小的点，如果有多个这样的点则取最左边的点作为 PO；

设<P1,P2,……,Pm>是 Q 中剩余的点，对其按逆时针方向相对 PO 的极角进行排序，如果有数个点有相同的极角，则去掉其余的点，只留下一个与 PO 距离最远的那个点；

TOP[S]：=0；PUSH（PO，S)；PUSH（P1，S)；PUSH（P2，S)；

FOR I：=3 TO M DO

begin

WHILE 由点 NEXT-TOP-TOP（S)，TOP（S）和 Pi 所形成的角形成一次非左转　DO

POP（S)；

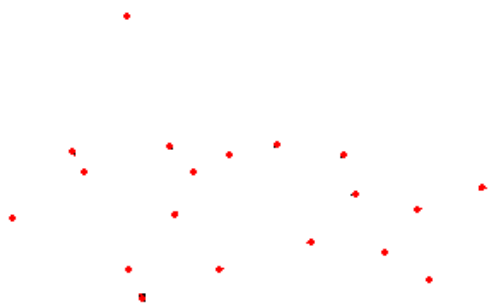
PUSH（S，Pi)；

End;

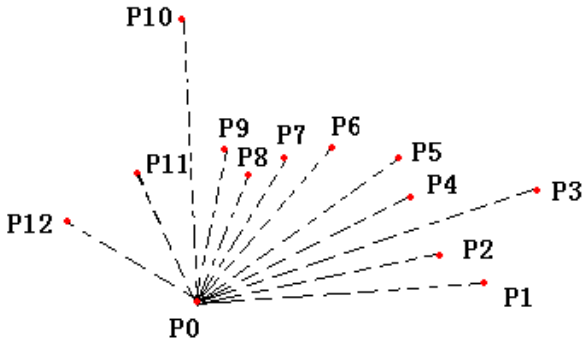
TETRUN(S)；

END;

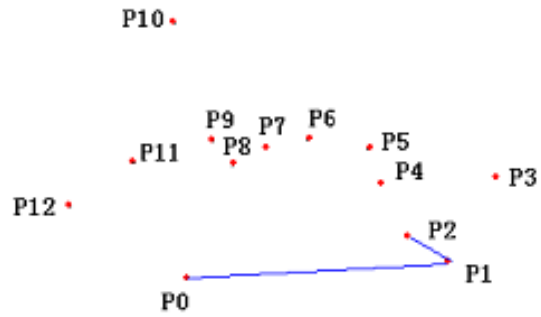
举例如下：



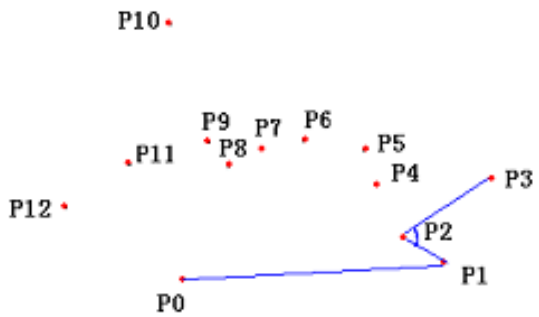
输入18个点



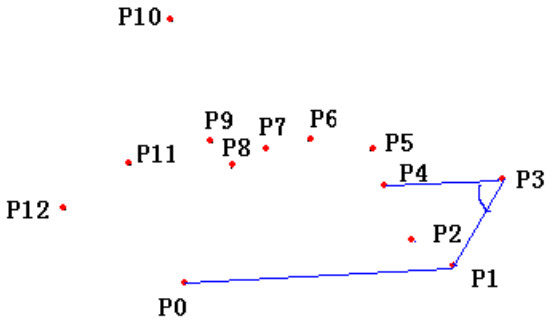
找到y值最小的点作为PO，如果有多个，则选择最左边的那个；
按逆时针方向相对于PO的极角进行排序，依次编上号P1—Pm，m=12
如果有多个极角相同，则去掉其余的点，只留下与PO最远的那个点；



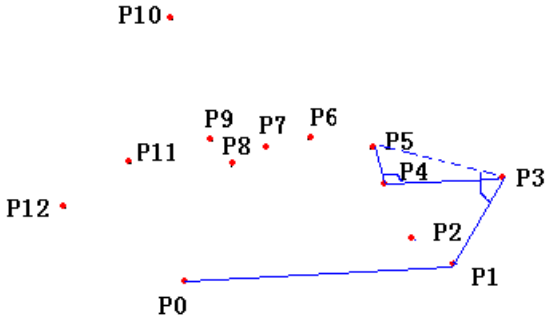
PO,P1,P2进栈



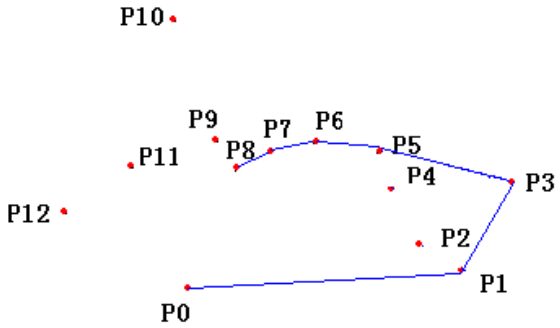
∠P1P2P3向左转，所以P2出栈
而∠P0P1P3是向右转，所以P1不用出栈；
同时P3进栈，变成下图



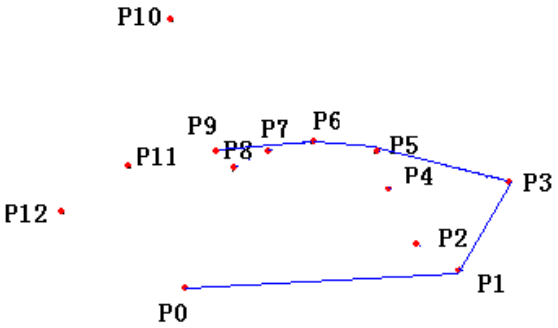
∠P1P3P4是向右转，所以P4进栈



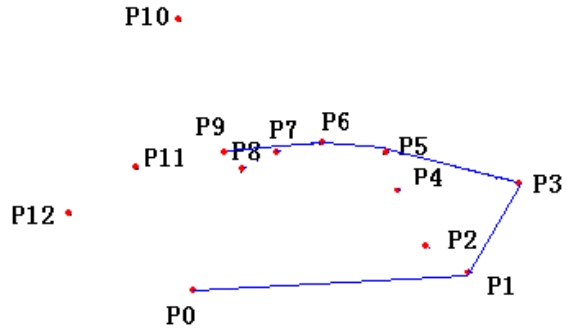
∠P3P4P5是想左转，所以P4出栈，
而∠P1P3P5是向右转，所以P5进栈



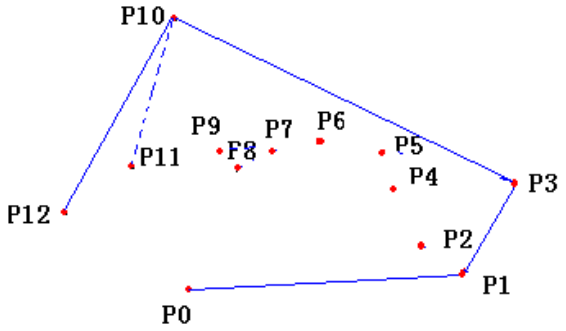
同理，会出现P6、P7、P8进栈



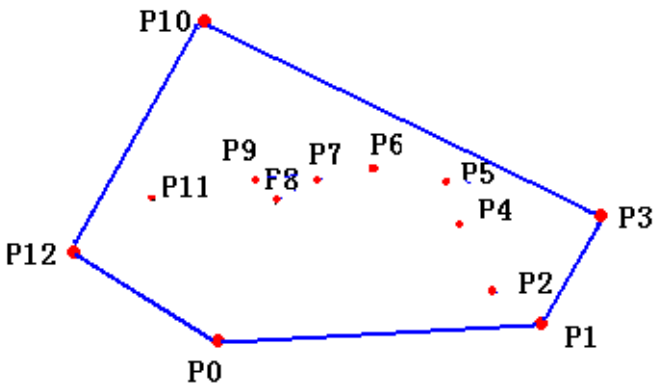
遇到P9，会现P8、P7出栈，而P9进栈



遇到P10，所以P9、P6、P5依次出栈
而P10进栈



遇到P11，P11进栈
但遇到P12后，P11又出栈，而P12进栈



遇到i=m(12)，结束。凸包就形成了。

小结：

这个算法框架的时间复杂度为 O（n）。Graham 算法的时间复杂度主要在快速排序上，所以总的时间复杂度为 O（nlogn）。由于有排序的预处理，找边时就省去了盲目的扫描，虽然排序过程要花费时间，总的时间效率还是提高了许多。

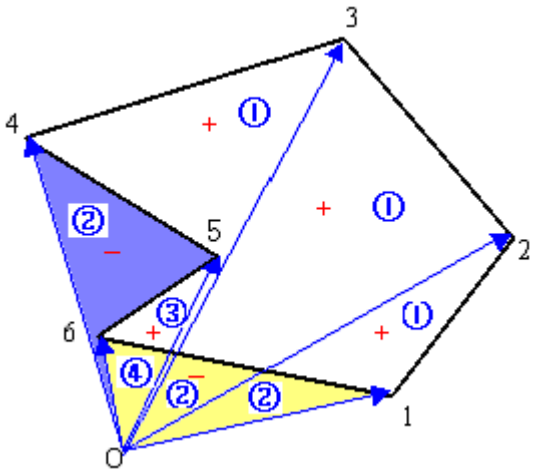
Jarvis 算法和 Graham 算法都充分利用了矢量的叉乘（矢量积）对判断矢量旋转方向的作用，都具有简洁、高效的特点。这是矢量的定向旋转在计算几何中的重要应用。

2、 求凸多边形的面积

问题描述：按一定的顺序（顺、逆时针）给出一个多边形的各顶点坐标，求该多边形的面积。

问题分析：

我们知道任何一个 n 边形都可以分割为(n—2)个三角形，找出这些三角形，把面积相加是一种解决问题的办法。但是这样免不了要用繁琐的算法，编程复杂度和时间复杂度很差。



如果引入“A 和 B 的矢量积是一个矢量，其模等于由 A 和 B 作成的平行四边形的面积……”，情况就不同了。设顶点序列为 P_1，P_2…P_n，则图形的面积：

$$A=\frac{1}{2}\left|\sum_{i=1}^{n}\overrightarrow{OP_i}\times\overrightarrow{OP_{i+1}}\right|\left(P_{n+1}=P_1\right)$$

这个公式对无论什么形状的多边形都成立，如上图。它同样是把图形分割为三角形，但分割的方法与上面的方法大相径庭。上图已经清楚地显示了这一点。按这个方法分出的三角形中，有的面积是负的。由矢量积的定义，矢量 A 逆时针转过一个小于 π 的角到 B，则 A×B 的模等于 A、B 形成的三角形面积的 2 倍（平行四边形面积），方向同 z 轴；如果旋转方向为顺时针，则矢量积的方向与 z 轴相反。因此，按上述公式，矢量的端点沿着多边形转一圈，多边形内的区域被矢量扫过奇数次，面积被记入总和，形外的区域被扫过偶数次，两个相反方向的矢量积相互抵消，于是就得到了多边形的面积。上图圆圈内的数字为扫描过的次数，两个阴影部分为减去的面积。

主要程序段如下：

```
ans:=0;
for i:=1 to n do ans:=ans+(p[i].x+p[i+1].x)*(p[i].y-p[i+1].y);
if ans<0 then ans:=-ans;
ans:=ans/2;
```

3、 求两个凸多边形的交集面积

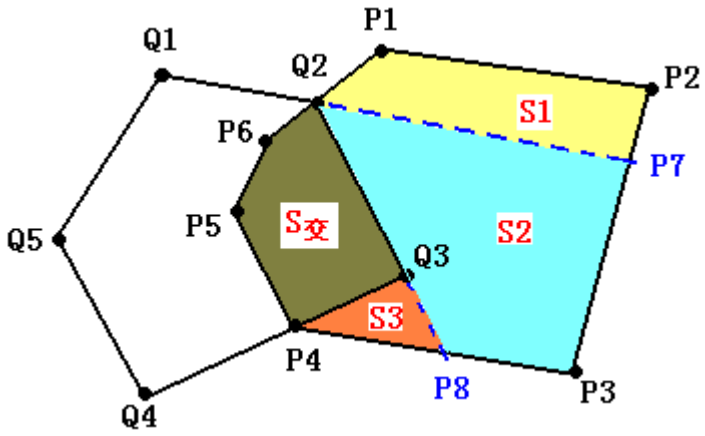
问题描述：已知两个凸多边形，求他们的交集的面积

输入：输入文件共 n+m+2 行

第一行为 n，后面的 n 行为 n 个点的 x,y 坐标，每行两个实数；

接下来一行为 m，再后面的 m 行为 m 个点的 x,y 坐标，每行两个实数；

输出：这个 n 边形和 m 边形的交集面积，精确到小数点后 3 位。



算法分析：

用其中一个凸多边形 Q 的每条边(Qi,Qi+1)对另一个凸多边形 P 进行切割，每次切割保留 P 中与 Q 中顶点处在其边(Qi,Qi+1)同侧的顶点以及所有线段(Pj-1,Pj)与直线(Qi,Qi+1)的交点。

$S_{交}=S_P-S_1-S_2-S_3$

这样问题就转换为：求若干个凸多边形的面积了。

参考程序：program\area.pas

4、 求 N 个点中的最接近点对

算法分析：略，参见《新高级本》一书 P69；

参考程序：program\MinDistance_Among_Points.pas

五、应用举例

例 1、监视摄像仪（CTSC1998）

问题描述：

一个著名的仓库管理公司 SERKOI 请你为其安装一套闭路监视系统，由于 SERKOI 财力有限，每个房间只能安装一台摄像机，不过其镜头可以向任何方向转换。

请你写一个程序，对于给定的房间示意图，判断是否有可能在这个房间中的某一位置安置一台摄像机，使其能监视到房间的每一个角落。

输入格式：

第一行是一个整数 n（4<=n<=100），表示房间的示意图是一个 n 边形。以下 n 行，每一行给出一个顶点的坐标（xn,yn）。

输出格式：

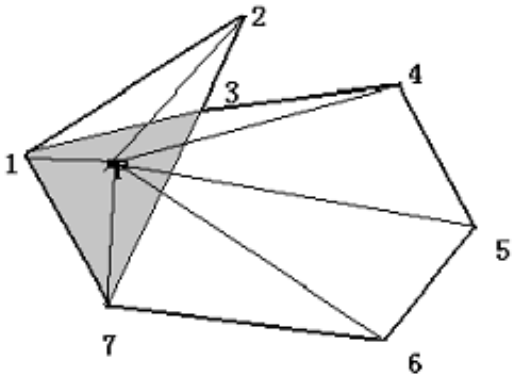
如果能找出安放摄像机的点则输出'OK！'，否则输出'IMPOSSIBLE！'。

问题分析：

复述一下题意：有一个多边形房间，判断能否在房间的某一点安装一个监视器，从而能监视到整个房间。即：顺时针给出一个多边形房间的 n 个顶点的坐标 V_i（X_i, Y_i），判断能否在某一点看到房间的全部。

题目中有两个很重要的条件：房间内部是相通的，是一个任意多边形；光线是直线传播的，因此，在监视范围内，不能有障碍物。如以下情况都是不符合要求的：房间内部不连通，光线被阻挡。

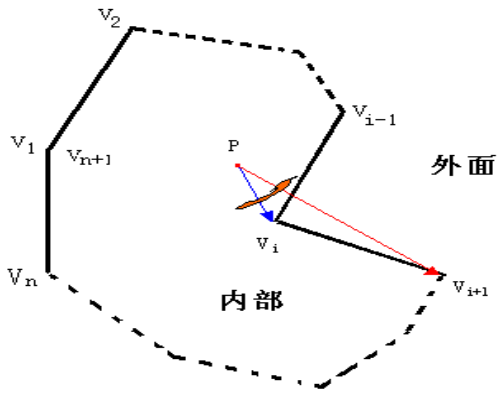
也就是说，要求判定给定的多边形中是否存在这样一点 p，它与多边形内任意一点的连线都不与多边形的边相交。如果 p 与多边形的各个顶点的连线都不多边形的边相交，则它与形内任意点的连线也不与多边形的边相交，点 p 就可以安装摄像机。[p 与各顶点的连线所分出的三角形中的点也都不与边相交]。即点 p 应该在任意一条边的右侧或在边上。如下图就是符合要求的。



[算法 1]

1. 初始化，求 n 条边所在的直线方程；
2. For I:=1 to n do
   For j:=1 to n do
   If (I<>j) and (linei 与 linej 相交) not (交点在多边形的任意一边的左侧)
   Then 存在点 p；
3. 否则，不存在点 p。

本题也可以用矢量方法做，而且容易的多。由于顶点是按顺时针方向给出的，对图形内某一点 P 和一条边 V_iV_{i+1}，如果从 $\overrightarrow{PV_i}$ 转到 $\overrightarrow{PV_{i+1}}$ 为逆时针旋转，则从 P 点必看不到边 V_iV_{i+1}。我们可以得出这样的结论：从 P 点可以看到整个房间当且仅当对任意一条边 V_iV_{i+1}（$V_{n+1}=V_1$），$\overrightarrow{PV_i}$ 转到 $\overrightarrow{PV_{i+1}}$ 为顺时针旋转或没有旋转，即（$\overrightarrow{PV_i} \times \overrightarrow{PV_{i+1}}$）z<=0。这里的矢量都在 xy 平面内，所以它们的矢量积垂直于 xy 平面，在计算 A×B 时，我们只需计算 z 方向的分量 AxBy-AyBx。解本题的关键——矢量旋转方向的判断，只需一个 AxBy-AyBx 就实现了，于是问题就解决了。



[算法 2]

先把所有墙壁所在的直线两两之间的交点求出；(所有可能的边界点)

Answer := false;

对求出的每一个交点 Vi

{　Answer := Answer or (Vi 可看到房间的全部)　}

要是改用解析几何方法，就必须求许多直线、线段的相交情况，显然无法设计出这样简单、高效的算法了。矢量方法的优越性由此可见一斑。

例 2、矩形覆盖

问题描述：

有一正 N 边形，给定它的三个顶点(not colinear)。求一面积最小的且边均平行于 x,y 轴的

矩形，它能完全盖住这个正 N 边形。输出这一矩形的最小面积。

输入：

第一行是 N，N<=1e7。接下来三行每行是一个顶点的坐标（实数）。

输出：

最小面积，一行一个实数，精确到小数点后 3 位。

注意：

每个测试点有多组数据组成，以一个 0 表示结束。

样例输入：

```
4
10.00000 0.00000
0.00000 -10.00000
-10.00000 0.00000
6
22.23086 0.42320
-4.87328 11.92822
1.76914 27.57680
23
156.71567 -13.63236
139.03195 -22.04236
137.96925 -11.70517
0
```

样例输出：

```
400.000
1056.172
397.673
```

例 3、双人游戏(game.pas,c,cpp,exe)

问题描述：

游戏者 B 在一张 100*100 的纸上确定了一个目标点，游戏者 A 一开始在点(0,0)上，每次游戏者 A 从一个点到另一个点，如果新的点离目标点近了，那么游戏者 B 说“Hotter”，如果新的点离目标远了，那么游戏者 B 说“Colder”，如果距离不变，那么游戏者 B 说“Same”。

输入文件包括很多行，每行包含游戏者 A 这一步到达的点(x,y)和游戏者 B 说的话，对每次游戏者 B 说话，判断目标点可能的位置的面积，精确到小数点后 2 位。

问题分析：

这是一道纯粹的计算求解题，首先证明可能的位置的图形一定是个凸多边形。

因为每次对游戏者 B 的回答，就可以确定可能的位置在出发点和到达点中垂线的哪一边或就是中垂线，每次的可能图形都是凸多边形。所以这个图形是许多个凸多边形的交集，所以这个图形是凸多边形。

接下来就是解题了，先令多边形为一个四边形，(0,0),(0,100), (100,100),(100,0)，然后对每次

游戏者 B 的回答，用这条中垂线将多边形分成 2 部分，取可能的那部分，即可。

不过这样并不是完全正确的，必须考虑到特殊情况，比如游戏者 A 到达的点和这步前的点完全相同，这时就不存在中垂线了，这些都是解题中要注意的重点。

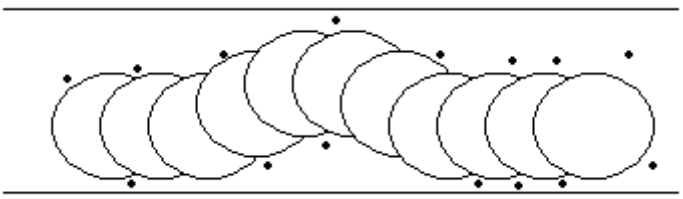
在这道题中就用到了很多计算几何的知识，包括求线段之间的交点，判断点是否在线段的两边，证明最终图形是凸多边形等等。

参考程序：program\game.pas;

例 4、人穿柱子游戏(fat.pas,c,cpp,exe)

问题描述：

在一个无限长的条形路上，有 n(n<=200)个柱子，体积不计，有一个人想从左边走到右边，人近似看成一个半径为 R 的圆（如下图），问能否实现？



分析：

拿到这道题最基本的做法是对从最左边的柱子到最右边的柱子中，每一个竖列进行扫描，计算可走到的范围，如果到最右边的柱子所在的列都有可走到的范围，则有解，否则无解。可是如果最左和最右的 2 个柱子相距非常远，那么这样计算的时间复杂度无疑是非常高的，所以我们应该对这个几何模型进行转化。

首先在这个图形中，不动的是柱子（近似看成点），动的是人（近似看成一个圆），这样处理比较麻烦，所以我们应该先把动的转换成点，圆转换成圆心是最容易想到的，对圆心来说，和柱子的距离不能<R，所以可以把每个柱子转换为以其为圆心，半径为 R 的圆，人转换成他的圆心，这样就使得计算可走到范围容易多了。

不过转换成这个模型后，问题还没有得到根本的解决，必须进一步的转换。

前 2 个算法有一个公共的特点，就是计算的都是圆外的部分，而计算圆内的部分的连通性显然比计算圆外部分来得简单，所以我们现在的目标就是把圆外部分换成圆内部分。

因为左右方向上是无穷长的，所以如果左右部分在圆外相通的话，那么上下两条直线在圆内部分就是不相通的，反之如果左右部分在圆外不相通的话，那么上下两条直线在圆内部分就是相通的，

所以我们可以将对每一个竖列的扫描转换成对每一个横行的扫描，而且又是在圆内操作，效率大大提高了。

但是前面的转换，对模型的抽象化程度却一点也没有改进，如何在这方面进行改进无疑是最关键的。

分析圆的特性，任意 2 个属于同一个圆的点必定是相通的，这就启发我们利用圆的特性，把难以处理的区域转换成几个具有代表性的点，使得能够完全表示出区域连通的特性来，到了这里，应该很容易就可以看出，取每个圆的圆心是最好不过了，因为每个圆的大小完全相同，不存在包

含，相切（如果内切，就是重合了，如果外切，就是中间不连通的）等等复杂的关系，只有相交和相离的关系，而且如果 2 个圆之间相交的话，那么这 2 个圆就是相通的，可以在这 2 个圆的圆心之间连一条边，增加一个源点，与上边有交点的圆和源点连一条边，增加一个汇点，与下边有交点的圆和汇点连一条边，这样就把一道几何题完全转换成了一道图论题，只要判断源点和汇点之间是否有路就可以了，这是一道非常经典的图论题，解法就不说了。

从上面这道题的解题过程中，可以得出这样的结论：解这一类的几何题必须充分了解几何变换，挖掘题目中隐含的线索，对题目的本质进行充分的探索，才能做好几何题。

参考程序：program\fat.pas;

例 5、奶牛的位置(enemy.pas,c,cpp,exe)

问题描述：

一个农夫在一个 x*y 的矩形田地上放牧 n 头奶牛（n<=25），它们互相之间都非常仇视，所以都希望能够离其它奶牛尽量的远，它们有它们自己的标准，就是离其它奶牛的距离的倒数和越小越好，因为农夫想让它们尽量高兴，所以他必须找到一种使所有的奶牛之间的距离的倒数和尽量小，不过学历不高的农夫觉得自己很难做到，请你来为他找到这种方案，他将按照方案的解和最优解的差距来决定付给你的酬劳的多少（他知道最优解还叫你求什么呢？;））

输入 x,y,n，输出你的解的 n 头奶牛的位置。

问题分析：

显然这是一道求几何最值的问题，而且显然是应该用近似算法来做，那么决定解决问题质量的就是近似算法的优劣。

最简单的想法就是：既然 n 最多也只不过是 25，就来个“没有功劳，也有苦劳”吧，对在正方形上手算得到的比较好的解，按照比例放到矩形里去，比如 n=4 时，正方形上的解是四个角，而一般（x,y 之间相差不大）的矩形，这也就是最优解了。

但是这种算法在 x,y 之间差距比较大的时候，就充分显示出它的不足了，和最优解的差距非常大，几乎 1 分也得不到了，所以这种算法并不是一种非常好的方法，只能够混混而已。

在求最优解的题目中贪心法是很难有用武之地，但是贪心法是一种非常常用的近似算法，所以解这道题目，可以用贪心法一试。

第一个点取(0,0)，以后每个点都取可以使当前的倒数和最大的点，当然这里要用到逐步求精法，这样可以得到一个比较接近最优解的方案，经过分析，也很难找到可以使贪心法得到离最优解较远的数据，所以这道题中贪心法完全是一种有效的算法。

不过贪心法终究和最优解有一定距离，并不能得到所有的分数，所以需要找到另一种更加有效的算法，由于这道题没有固定的最佳解法，所以，对任何固定算法应该都有使其得不到满分的数据，所以我们又想到了非固定算法：随机化算法。这个算法在这里我就不多介绍了，留给大家自己去思考。

参考程序：贪心法,见 program\enemy.pas;

例 6、铁路连线问题

问题描述：

皮亚琴查是意大利一个美丽的城市，有着丰富的旅游资源。该地的旅游景点又被分为 N 个自然景观和 N 个人文景观，为了方便游客，当地政府决定在每两个自然景观和人文景观之间增设一条铁路，也就是增加 N 条铁路。为了降低工程难度，他们要求铁路之间不能相交，现在你的手边有每个自然景观和人文景观的坐标。市长 PIPPO 要求你的程序给出一个可行的铁路连线方案。

输入：

第 1 行为一个数字 N（0<N<10000）；

第 2 N+1 行为 N 个自然景观的坐标，两个数字之间用空格隔开；

第 N+2 2N+1 行为 N 个人文景观的坐标，两个数字之间也用空格隔开；

输入数据保证有解。

输出：

输出数据包括 N 行，为一种可行解的方案。

每行包括两个数据，即两个连线的景观的编号。

0≤矩形数目<5000；

坐标数值为整数，范围是[-10000，10000]。

问题分析：

本题是一道简单的几何算法和分治策略相结合的题目。

我们先用与 Graham 扫描法相类似的方法对于题目给出的点进行顺时针或逆时针排序，然后找出一点 y 坐标值最低的点记为 V1，按顺序扫描这些点，累加所扫描到的人文景观和自然景观的个数，（注意：V1 也要统计在内），直到扫描到的一点 Vj，使人文景观和自然景观的个数相等，恰好可以与 V1 相搭配，我们就在 V1,Vj 之间建立一条铁路连先，这样原本排好序的集合（V1,V2……VN）就可以化为两个更小的集合（V2……Vj-1）和（Vj+1……Vn）,可以递归的求解。

例 7、音乐厅布置（Hall.pas/c/cpp）

【背景】

建成于 1891 年的纽约卡内基音乐厅（Carnegie Hall）音响效果绝佳，是世界最著名的音乐厅之一，它有很大的舞台，在这个音乐季，他们正准备演出莫扎特（Mozart）的歌剧《后宫诱逃（Die Entfuehrung aus dem Serail)》。剧中某些场景需要搭建一个土耳其皇宫，正如我们所熟悉的阿拉伯建筑风格，"皇宫"由一些高度、直径都相同的圆柱支撑，其中一些柱子和外墙相接，另一些在皇宫内部，所有外墙上的柱子向皇宫内的一侧的地面上都放有灯（假设灯放在柱子的圆心这个点上），灯光可以照射到皇宫内的任何部分，除非被东西遮挡。"皇宫"外围是灰白色的"石壁"，就是两根柱子中间的部分，并且总在两根柱子圆心的连线上。墙壁都有一定的透光度，使得内部灯光散射出来，晶莹剔透。因为半透明，墙壁不太厚，相对于柱子的直径，我们忽略墙壁的厚度，如右图所示，灰色部分表示皇宫内部，蓝色部分表示柱子内部。皇宫的横截面总是凸多边形，上方有和横截面几何形状一模一样且厚度密度都均匀的平顶（图中线段围住的部分，即以各

个外墙上的柱子的圆心为顶点的凸多边形)。演出时场景布置当然要富丽堂皇、尽善尽美，因此他们请天才的你来解决他们的一切问题。

【问题】

1、 舞台上不能放置太重的东西，柱子、墙壁都是用塑料等轻的材料做的，不能承重。而皇宫的平顶面积大，还是挺重的，不能直接压在这些材料上，只能通过舞台顶部的架子悬挂。当然，悬挂用的绳子是越细越少越好。他们提出一个大胆的想法，让你只用一根非常坚固的绳子，悬挂起整个屋顶。你的任务就是，找到平顶上一个恰当的点，使得从这个点可以把平顶平稳的吊起来。

2、 布景时，布景师们在顶部的架子上走动，时常不小心碰到一些工具，掉下去砸坏了道具。因此，他们观察了几个经常放工具的点，并告诉你这些点在舞台上投影的位置，请你判断一下这些东西竖直掉下会对"皇宫"造成什么影响。某物品从某个点落下，如果恰好砸在柱子上（内），输出 C；砸到墙壁上，输出 L；砸到皇宫内空地，输出 I；落到皇宫外，输出 O；如果砸到物体的交界处，只需按照 CLIO 的顺序输出第一个。

3、 演出时，录音师为演出制作环绕立体声录音。为了排除墙壁隔音的影响，他们在"皇宫"里的地面上也放置了几个麦克风。但为了不影响灯光照射，麦克风必须放在灯光照射不到的地方。出于声学的考虑，录音师建议了几个适合放置麦克风的位置，请你从光学的角度，分析一下其中那些位置可以放置，哪些不可以。如某个麦克风可以放置，那么输出 1，否则输出 0。

我们用一个平面直角坐标系表示舞台地面，给你的数据包括：柱子总数、直径，"皇宫"高度，平顶厚度，每根柱子中心的坐标，以及问题 2、问题 3 要判断点的坐标。

如果你能完美解决了音乐厅面临的三个难题，演出后，你将被授予**"最佳协作奖"**。

【输入】

第 1 行有四个整数，是柱子总数 p、直径 d、"皇宫"高度 h1 和平顶厚度 h2，数据中间都用一个空格隔开。

以下 p 行，每行有两个实数 xi,yi，数据间用一个空格隔开，是第 i 根柱子的中心坐标。

第 p+2 行是一个整数 n，表示问题 2 要判定的顶点个数。

以下 n 行，每行有两个实数 xi,yi，是问题 2 要判定的第 i 个点的坐标，数据间用一个空格隔开。

第 p+n+3 行是一个整数 m，表示任务 3 要判定的顶点个数。

以下 m 行，每行有两个实数 xi,yi，数据间用一个空格隔开，是任务 5 要判定的第 i 个点的坐标，并保证这些点都在"皇宫"的可用空间内。

【输出】

第 1 行包含两个实数 x,y，表示可以悬挂平顶的点的坐标，数据间用一个空格间隔，保留三位小数。

第 2 行包括 n 个字母，是 CLIO 之一，表示问题 2 的结果，数据间用一个空格间隔。

第 3 行，输出 m 个数，是 0 或者 1，表示问题 3 的结构，数字间用一个空格间隔。

【样例输入】

7 1 110 10

```
-3 3
3 2
-2 -2
3 -1
0 0
1 1
-1 1
4
-4 1
-2 1
0 0
0 2.5
2
1 -1
0.25 0.75
```

【样例输出】

```
0.015 0.545
0 I C L
0 0
```

【算法分析】

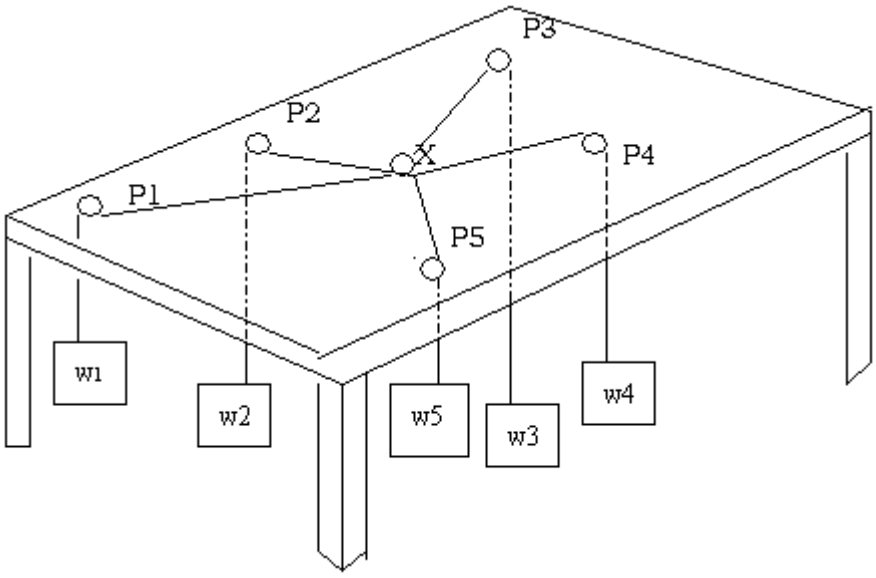
这一题从算法上来说，使比较简单的，只要仔细看清题目的要求，编程细心且耐心，就不难解决问题。

首先，求凸包是整个问题的基础，因为题目没有给出哪些柱子是在外，哪些在内，而这将是解决问题必须的条件。求凸包可以用卷包裹法或 Graham,我想大家对这应该是很熟悉的了。

我们得到凸包之后，很明显可以看出来，第一小题就是求凸包的重心。由于房顶的**密度厚度均匀**，所以房顶的厚度就是一个不需要考虑的多余条件。求凸多边形重心的一个简单方法就是我们顺次把凸包分割成为 n-3 个三角心，根据三角形中心的计算公式：$x=(x_a+x_b+x_c)/3$, $y=(y_a+y_b+y_c)/3$,就可以得到每个三角形的重心。然后计算三角形面积（如用海轮公式）。这样我们就可以计算两个图形的重心（x1,y1），（x2,y2），如已知他们的面积（正比于质量）S1，S2，有：λ=S2/S1，再利用定比分点公式：x=(x1+λ*x2)/(1+λ), y=(y1+λ*y2)/(1+λ),求出这两个图形整体的重心（x,y）。

例 8、平衡点（jsoi2004 省队选拔赛最后一题）

问题描述:



如图：有 n 个重物，每个重物系在一条足够长的绳子上。每条绳子自上而下穿过桌面上的洞，然后系在一起。图中 X 处就是公共的绳结。假设绳子是完全弹性的（不会造成能量损失），桌子足够高（因而重物不会垂到地上），且忽略所有的摩擦。

问：绳结 X 最终平衡于何处？

注意：桌面上的洞都比绳结 X 小得多，所以即使某个重物特别重，绳结 X 也不可能穿过桌面上的洞掉下来，最多是卡在某个洞口处。

输入：

你的程序将从文件读入数据（input8.DAT）。文件的第一行为一个正整数 n（1≤n≤1000），表示重物和洞的数目。接下来的 n 行，每行是 3 个整数：X_i Y_i W_i，分别表示第 i 个洞的坐标以及第 i 个重物的重量。**(-10000≤x,y≤10000, 0<w≤1000)**

输出：

你的程序必须在屏幕上输出两个浮点数（保留小数点后三位），分别表示处于最终平衡状态时绳结 X 的横坐标和纵坐标。

样例输入：

```
3
0  0  1
0  2  1
1  1  1
```

样例输出:

```
0.577    1.000
```

例 9、巨人和鬼

问题描述：

有 n 个巨人正与 n 个鬼进行战斗。每个巨人的武器是一个质子包，他（她）可以用一串质子流射中鬼而把鬼消灭。质子流沿直线行进，在击中鬼时就终止。巨人们决定采取下列战略：他们各寻找一个鬼以形成 n 个巨人-鬼对，然后每个巨人同时向他（她）选取的鬼射出一串质子流。我们知道，让质子流相互交叉是很危险的，因此巨人选择的配对方式应该使质子流都不会交叉。

假定每个巨人和每个鬼的位置都是平面上一个固定的点，并且没有三个共线。

问题 1：

论证：存在一条通过一个巨人和一个鬼的直线使直线一边的巨人数和同一边的鬼数相等，说明如何在 O（nlgn）的时间内找到这样一条直线。

问题 2：

编写一个 O（n^2lgn）的算法，使其按不会有质子流交叉的条件把巨人和鬼配对输出。

六、小结与思考

几何题是信息学竞赛的重要组成部分，这一类问题与其他类型的题目相比，对算法和数据结构知识的考察相对少一些，而对选手的数学功底，尤其是解析几何等数学工具的掌握水平提出了较高的要求。

信息学问题有一个普遍的规律（其他学科往往也是如此）：数学工具越高级，解决问题就越高效。矢量分析是高等数学中的一个分支，是一种高级的数学工具。其中的两个式子虽看似简单，

$$\mathbf{A}\cdot\mathbf{B}=AxBx+AyBy+AzBz$$

$$\mathbf{A}\times\mathbf{B}=\begin{vmatrix} i & j & k \\ Ax & Ay & Az \\ Bx & By & Bz \end{vmatrix}$$

却包含着极为丰富的内容。由它们可以导出许多有用的结论，在计算几何问题中可以得到极为广泛的应用。几乎在每一道信息学竞赛的几何题中，都可以找到发挥它独特作用的地方。甚至有一些题，只要建立运用矢量的模型，问题就几乎解决了。

综上所述，学好矢量分析，充分地理解并能够熟练地运用一些矢量方法与技巧，会给我们解决计算几何问题带来许多方便。