# 计算几何资料

一、引言

计算机的出现使得很多原本十分繁琐的工作得以大幅度简化，但是也有一些在人们直观看来很容易的问题却需要拿出一套并不简单的通用解决方案，比如几何问题。作为计算机科学的一个分支，计算几何主要研究解决几何问题的算法。在现代工程和数学领域，计算几何在图形学、机器人技术、超大规模集成电路设计和统计等诸多领域有着十分重要的应用。在本文中，我们将对计算几何常用的基本算法做一个全面的介绍，希望对您了解并应用计算几何的知识解决问题起到帮助。

二、目录

本文整理的计算几何基本概念和常用算法包括如下内容：

1. 矢量的概念
2. 矢量加减法
3. 矢量叉积
4. 折线段的拐向判断
5. 判断点是否在线段上
6. 判断两线段是否相交
7. 判断线段和直线是否相交
8. 判断矩形是否包含点
9. 判断线段、折线、多边形是否在矩形中
10. 判断矩形是否在矩形中
11. 判断圆是否在矩形中
12. 判断点是否在多边形中
13. 判断线段是否在多边形内
14. 判断折线是否在多边形内
15. 判断多边形是否在多边形内
16. 判断矩形是否在多边形内
17. 判断圆是否在多边形内
18. 判断点是否在圆内
19. 判断线段、折线、矩形、多边形是否在圆内
20. 判断圆是否在圆内
21. 计算点到线段的最近点
22. 计算点到折线、矩形、多边形的最近点
23. 计算点到圆的最近距离及交点坐标
24. 计算两条共线的线段的交点
25. 计算线段或直线与线段的交点
26. 求线段或直线与折线、矩形、多边形的交点
27. 求线段或直线与圆的交点
28. 凸包的概念
29. 凸包的求法

三、算法介绍

1.矢量的概念：

如果一条线段的端点是有次序之分的，我们把这种线段成为有向线段(directed segment)。如果有向线段 p1p2 的起点 p1 在坐标原点，我们可以把它称为矢量(vector)p2。

2.矢量加减法：

设二维矢量 P = ( x1, y1 )，Q = ( x2 , y2 )，则矢量加法定义为： P + Q = ( x1 + x2 , y1 + y2 )，同样的，矢量减法定义为： P - Q = ( x1 - x2 , y1 - y2 )。显然有性质 P + Q = Q + P，P - Q = - ( Q - P )。

3. 矢量叉积：

计算矢量叉积是与直线和线段相关算法的核心部分。设矢量 P = ( x1, y1 )，Q = ( x2, y2 )，则矢量叉积定义为由(0,0)、p1、p2 和 p1+p2 所组成的平行四边形的带符号的面积，即：P × Q = x1*y2 - x2*y1，其结果是一个标量。显然有性质 P × Q = - ( Q × P ) 和 P × ( - Q ) = - ( P × Q )。一般在不加说明的情况下，本文下述算法中所有的点都看作矢量，两点的加减法就是矢量相加减，而点的乘法则看作矢量叉积。

叉积的一个非常重要性质是可以通过它的符号判断两矢量相互之间的顺逆时针关系：

若 P × Q > 0，则 P 在 Q 的顺时针方向。

若 P × Q < 0，则 P 在 Q 的逆时针方向。

若 P × Q = 0，则 P 与 Q 共线，但可能同向也可能反向。

4. 折线段的拐向判断：

折线段的拐向判断方法可以直接由矢量叉积的性质推出。对于有公共端点的线段 p0p1 和 p1p2，通过计算(p2 - p0) × (p1 - p0)的符号便可以确定折线段的拐向：

若(p2 - p0) × (p1 - p0) > 0,则 p0p1 在 p1 点拐向右侧后得到 p1p2。

若(p2 - p0) × (p1 - p0) < 0,则 p0p1 在 p1 点拐向左侧后得到 p1p2。

若(p2 - p0) × (p1 - p0) = 0,则 p0、p1、p2 三点共线。

具体情况可参考下图：

5. 判断点是否在线段上：

设点为 Q，线段为 P1P2 ，判断点 Q 在该线段上的依据是：( Q - P1 ) × ( P2 - P1 ) = 0 且 Q 在以 P1，P2 为对角顶点的矩形内。前者保证 Q 点在直线 P1P2 上，后者是保证 Q 点不在线段 P1P2 的延长线或反向延长线上，对于这一步骤的判断可以用以下过程实现：

```
ON-SEGMENT(pi,pj,pk)
if min(xi,xj) <= xk <= max(xi,xj) and min(yi,yj) <= yk <= max(yi,yj)
then return true;
else return false;
```

特别要注意的是，由于需要考虑水平线段和垂直线段两种特殊情况，min(xi,xj)<=xk<=max(xi,xj)和 min(yi,yj)<=yk<=max(yi,yj)两个条件必须同时满足才能返回真值。

6. 判断两线段是否相交：

我们分两步确定两条线段是否相交：

(1)快速排斥试验

设以线段 P1P2 为对角线的矩形为 R， 设以线段 Q1Q2 为对角线的矩形为 T，如果 R 和 T 不相交，显然两线段不会相交。

(2)跨立试验

如果两线段相交，则两线段必然相互跨立对方。若 P1P2 跨立 Q1Q2 ，则矢量 ( P1 - Q1 ) 和( P2 - Q1 )位于矢量( Q2 - Q1 ) 的两侧，即( P1 - Q1 ) × ( Q2 - Q1 ) * ( P2 - Q1 ) × ( Q2 - Q1 ) < 0。上式可改写成( P1 - Q1 ) × ( Q2 - Q1 ) * ( Q2 - Q1 ) × ( P2 - Q1 ) > 0。当 ( P1 - Q1 ) × ( Q2 - Q1 ) = 0 时，说明 ( P1 - Q1 ) 和 ( Q2 - Q1 )共线，但是因为已经通过快速排斥试验，所以 P1 一定在线段 Q1Q2 上；同理，( Q2 - Q1 ) ×(P2 - Q1 ) = 0 说明 P2 一定在线段 Q1Q2 上。所以判断 P1P2 跨立 Q1Q2 的依据是：( P1 - Q1 ) × ( Q2 - Q1 ) * ( Q2 - Q1 ) × ( P2 - Q1 ) >= 0。同理判断 Q1Q2 跨立 P1P2 的依据是：( Q1 - P1 ) × ( P2 - P1 ) * ( P2 - P1 ) × ( Q2 - P1 ) >= 0。具体情况如下图所示：

在相同的原理下，对此算法的具体的实现细节可能会与此有所不同，除了这种过程外，大家也可以参考《算法导论》上的实现。

7. 判断线段和直线是否相交：

有了上面的基础，这个算法就很容易了。如果线段 P1P2 和直线 Q1Q2 相交，则 P1P2 跨立 Q1Q2，即：( P1 - Q1 ) × ( Q2 - Q1 ) * ( Q2 - Q1 ) × ( P2 - Q1 ) >= 0。

8. 判断矩形是否包含点：

只要判断该点的横坐标和纵坐标是否夹在矩形的左右边和上下边之间。

9. 判断线段、折线、多边形是否在矩形中：

因为矩形是个凸集，所以只要判断所有端点是否都在矩形中就可以了。

10. 判断矩形是否在矩形中：

只要比较左右边界和上下边界就可以了。

11. 判断圆是否在矩形中：

很容易证明，圆在矩形中的充要条件是：圆心在矩形中且圆的半径小于等于圆心到矩形四边的距离的最小值。

12. 判断点是否在多边形中：

判断点 P 是否在多边形中是计算几何中一个非常基本但是十分重要的算法。以点 P 为端点，向左方作射线 L，由于多边形是有界的，所以射线 L 的左端一定在多边形外，考虑沿着 L 从无穷远处开始自左向右移动，遇到和多边形的第一个交点的时候，进入到了多边形的内部，遇到第二个交点的时候，离开了多边形，.....所以很容易看出当 L 和多边形的交点数目 C 是奇数的时候，P 在多边形内，是偶数的话 P 在多边形外。

但是有些特殊情况要加以考虑。如图下图(a)(b)(c)(d)所示。在图(a)中，L 和多边形的顶点相交，这时候交点只能计算一个；在图(b)中，L 和多边形顶点的交点不应被计算；在图(c)和(d) 中，L 和多边形的一条边重合，这条边应该被忽略不计。如果 L 和多边形的一条边重合，这条边应该被忽略不计。

为了统一起见，我们在计算射线 L 和多边形的交点的时候，1。对于多边形的水平边不作考虑；2。对于多边形的顶点和 L 相交的情况，如果该顶点是其所属的边上纵坐标较大的顶点，则计数，否则忽略；3。对于 P 在多边形边上的情形，直接可判断 P 属于多边行。由此得出算法的伪代码如下：

```
count ← 0;
以 P 为端点，作从右向左的射线 L;
for 多边形的每条边 s
do if P 在边 s 上
then return true;
if s 不是水平的
then if s 的一个端点在 L 上
if 该端点是 s 两端点中纵坐标较大的端点
then count ← count+1
else if s 和 L 相交
then count ← count+1;
if count mod 2 = 1
then return true;
else return false;
```

其中做射线 L 的方法是：设 P'的纵坐标和 P 相同，横坐标为正无穷大（很大的一个正数），则 P 和 P'就确定了射线 L。

判断点是否在多边形中的这个算法的时间复杂度为 O(n)。

另外还有一种算法是用带符号的三角形面积之和与多边形面积进行比较，这种算法由于使用浮点数运算所以会带来一定误差，不推荐大家使用。

13. 判断线段是否在多边形内：

线段在多边形内的一个必要条件是线段的两个端点都在多边形内，但由于多边形可能为凹，所以这不

能成为判断的充分条件。如果线段和多边形的某条边内交（两线段内交是指两线段相交且交点不在两线段的端点），因为多边形的边的左右两侧分属多边形内外不同部分，所以线段一定会有一部分在多边形外(见图 a)。于是我们得到线段在多边形内的第二个必要条件：线段和多边形的所有边都不内交。

线段和多边形交于线段的两端点并不会影响线段是否在多边形内；但是如果多边形的某个顶点和线段相交，还必须判断两相邻交点之间的线段是否包含于多边形内部（反例见图 b)。

因此我们可以先求出所有和线段相交的多边形的顶点，然后按照 X-Y 坐标排序(X 坐标小的排在前面，对于 X 坐标相同的点，Y 坐标小的排在前面，这种排序准则也是为了保证水平和垂直情况的判断正确)，这样相邻的两个点就是在线段上相邻的两交点，如果任意相邻两点的中点也在多边形内，则该线段一定在多边形内。

证明如下：

命题 1：

如果线段和多边形的两相邻交点 P1 ，P2 的中点 P' 也在多边形内，则 P1，P2 之间的所有点都在多边形内。

证明：

假设 P1,P2 之间含有不在多边形内的点，不妨设该点为 Q，在 P1，P'之间，因为多边形是闭合曲线，所以其内外部之间有界，而 P1 属于多边行内部，Q 属于多边性外部，P'属于多边性内部，P1-Q-P'完全连续，所以 P1Q 和 QP'一定跨越多边形的边界，因此在 P1,P'之间至少还有两个该线段和多边形的交点，这和 P1P2 是相邻两交点矛盾，故命题成立。证毕。

由命题 1 直接可得出推论：

推论 2：

设多边形和线段 PQ 的交点依次为 P1,P2,.....Pn，其中 Pi 和 Pi+1 是相邻两交点，线段 PQ 在多边形内的充要条件是：P，Q 在多边形内且对于 i =1, 2,....., n-1，Pi ,Pi+1 的中点也在多边形内。

在实际编程中，没有必要计算所有的交点，首先应判断线段和多边形的边是否内交，倘若线段和多边形的某条边内交则线段一定在多边形外；如果线段和多边形的每一条边都不内交，则线段和多边形的交点一定是线段的端点或者多边形的顶点，只要判断点是否在线段上就可以了。

至此我们得出算法如下：

```
if 线端 PQ 的端点不都在多边形内
then return false;
点集 pointSet 初始化为空;
for 多边形的每条边 s
do if 线段的某个端点在 s 上
then 将该端点加入 pointSet;
else if s 的某个端点在线段 PQ 上
then 将该端点加入 pointSet;
else if s 和线段 PQ 相交 // 这时候已经可以肯定是内交了
then return false;
将 pointSet 中的点按照 X-Y 坐标排序;
for pointSet 中每两个相邻点 pointSet[ i ] , pointSet[ i+1]
do if pointSet [ i] , pointSet[ i+1] 的中点不在多边形中
then return false;
return true;
```

这个过程中的排序因为交点数目肯定远小于多边形的顶点数目 n，所以最多是常数级的复杂度，几乎可以忽略不计。因此算法的时间复杂度也是 O(n)。

14. 判断折线是否在多边形内：

只要判断折线的每条线段是否都在多边形内即可。设折线有 m 条线段，多边形有 n 个顶点，则该算法的时间复杂度为 O(m*n)。

15. 判断多边形是否在多边形内：

只要判断多边形的每条边是否都在多边形内即可。判断一个有 m 个顶点的多边形是否在一个有 n 个顶点的多边形内复杂度为 O(m*n)。

16. 判断矩形是否在多边形内：

将矩形转化为多边形，然后再判断是否在多边形内。

17 判断圆是否在多边形内：

只要计算圆心到多边形的每条边的最短距离，如果该距离大于等于圆半径则该圆在多边形内。计算圆心到多边形每条边最短距离的算法在后文阐述。

18. 判断点是否在圆内：

计算圆心到该点的距离，如果小于等于半径则该点在圆内。

19. 判断线段、折线、矩形、多边形是否在圆内：

因为圆是凸集，所以只要判断是否每个顶点都在圆内即可。

20. 判断圆是否在圆内：

设两圆为 O1,O2，半径分别为 r1，r2，要判断 O2 是否在 O1 内。先比较 r1，r2 的大小，如果 r1<r2 则 O2 不可能在 O1 内；否则如果两圆心的距离大于 r1 - r2 ，则 O2 不在 O1 内；否则 O2 在 O1 内。

21. 计算点到线段的最近点：

如果该线段平行于 X 轴（Y 轴），则过点 point 作该线段所在直线的垂线，垂足很容易求得，然后计算出垂足，如果垂足在线段上则返回垂足，否则返回离垂足近的端点；如果该线段不平行于 X 轴也不平行于 Y 轴，则斜率存在且不为 0。设线段的两端点为 pt1 和 pt2，斜率为：k = ( pt2.y - pt1. y ) / (pt2.x - pt1.x )；该直线方程为：y = k* ( x - pt1.x) + pt1.y。其垂线的斜率为 - 1 / k，垂线方程为：y = (-1/k) * (x - point.x) + point.y 。

联立两直线方程解得：x = ( k^2 * pt1.x + k * (point.y - pt1.y ) + point.x ) / ( k^2 + 1) ， y = k * ( x - pt1.x) + pt1.y;然后再判断垂足是否在线段上，如果在线段上则返回垂足；如果不在则计算两端点到垂足的距离，选择距离垂足较近的端点返回。

22. 计算点到折线、矩形、多边形的最近点：

只要分别计算点到每条线段的最近点，记录最近距离，取其中最近距离最小的点即可。

23. 计算点到圆的最近距离及交点坐标：

如果该点在圆心，因为圆心到圆周任一点的距离相等，返回 UNDEFINED。

连接点 P 和圆心 O，如果 PO 平行于 X 轴，则根据 P 在 O 的左边还是右边计算出最近点的横坐标为 centerPoint.x - radius 或 centerPoint.x + radius。如果 PO 平行于 Y 轴，则根据 P 在 O 的上边还是下边计算出最近点的纵坐标为 centerPoint.y -+radius 或 centerPoint.y - radius。如果 PO 不平行于 X 轴和 Y 轴，则 PO 的斜率存在且不为 0，这时直线 PO 斜率为 k = ( P.y - O.y )/ ( P.x - O.x )。直线 PO 的方程为：y = k * ( x - P.x) + P.y。设圆方程为(x - O.x ) ^2 + ( y - O.y ) ^2 = r ^2，联立两方程组可以解出直线 PO 和圆的交点，取其中离 P 点较近的交点即可。

24. 计算两条共线的线段的交点：

对于两条共线的线段，它们之间的位置关系有下图所示的几种情况。图(a)中两条线段没有交点；图 (b) 和 (d) 中两条线段有无穷焦点；图 (c) 中两条线段有一个交点。设 line1 是两条线段中较长的一条，line2 是较短的一条，如果 line1 包含了 line2 的两个端点，则是图(d)的情况，两线段有无穷交点；如果 line1 只包含 line2 的一个端点，那么如果 line1 的某个端点等于被 line1 包含的 line2 的那个端点，则是图(c)的情况，这时两线段只有一个交点，否则就是图(b)的情况，两线段也是有无穷的交点；如果 line1 不包含 line2 的任何端点，则是图(a)的情况，这时两线段没有交点。

25. 计算线段或直线与线段的交点:

设一条线段为 L0 = P1P2，另一条线段或直线为 L1 = Q1Q2 ，要计算的就是 L0 和 L1 的交点。

1. 首先判断 L0 和 L1 是否相交（方法已在前文讨论过），如果不相交则没有交点，否则说明 L0 和 L1 一定有交点，下面就将 L0 和 L1 都看作直线来考虑。
2. 如果 P1 和 P2 横坐标相同，即 L0 平行于 Y 轴
   a) 若 L1 也平行于 Y 轴，
      i. 若 P1 的纵坐标和 Q1 的纵坐标相同，说明 L0 和 L1 共线，假如 L1 是直线的话他们有无穷的交点，假如 L1 是线段的话可用"计算两条共线线段的交点"的算法求他们的交点（该方法在前文已讨论过）；
      ii. 否则说明 L0 和 L1 平行，他们没有交点；
   b) 若 L1 不平行于 Y 轴，则交点横坐标为 P1 的横坐标，代入到 L1 的直线方程中可以计算出交点纵坐标；
3. 如果 P1 和 P2 横坐标不同，但是 Q1 和 Q2 横坐标相同，即 L1 平行于 Y 轴，则交点横坐标为 Q1 的横坐标，代入到 L0 的直线方程中可以计算出交点纵坐标；
4. 如果 P1 和 P2 纵坐标相同，即 L0 平行于 X 轴
   a) 若 L1 也平行于 X 轴，
      i. 若 P1 的横坐标和 Q1 的横坐标相同，说明 L0 和 L1 共线，假如 L1 是直线的话他们有无穷的交点，假如 L1 是线段的话可用"计算两条共线线段的交点"的算法求他们的交点（该方法在前文已讨论过）；
      ii. 否则说明 L0 和 L1 平行，他们没有交点；
   b) 若 L1 不平行于 X 轴，则交点纵坐标为 P1 的纵坐标，代入到 L1 的直线方程中可以计算出交点横坐标；
5. 如果 P1 和 P2 纵坐标不同，但是 Q1 和 Q2 纵坐标相同，即 L1 平行于 X 轴，则交点纵坐标为 Q1 的纵坐标，代入到 L0 的直线方程中可以计算出交点横坐标；
6. 剩下的情况就是 L1 和 L0 的斜率均存在且不为 0 的情况
   a) 计算出 L0 的斜率 K0，L1 的斜率 K1 ；
   b) 如果 K1 = K2
      i. 如果 Q1 在 L0 上，则说明 L0 和 L1 共线，假如 L1 是直线的话有无穷交点，假如 L1 是线段的话可用"计算两条共线线段的交点"的算法求他们的交点（该方法在前文已讨论过）；
      ii. 如果 Q1 不在 L0 上，则说明 L0 和 L1 平行，他们没有交点。
   c) 联立两直线的方程组可以解出交点来

这个算法并不复杂，但是要分情况讨论清楚，尤其是当两条线段共线的情况需要单独考虑，所以在前文将求两条共线线段的算法单独写出来。另外，一开始就先利用矢量叉乘判断线段与线段（或直线）是否相交，如果结果是相交，那么在后面就可以将线段全部看作直线来考虑。需要注意的是，我们可以将直线或线段方程改写为 ax+by+c=0 的形式，这样一来上述过程的部分步骤可以合并，缩短了代码长度，但是由于先要求出参数，这种算法将花费更多的时间。

26. 求线段或直线与折线、矩形、多边形的交点：

分别求与每条边的交点即可。

27. 求线段或直线与圆的交点:

设圆心为 O，圆半径为 r，直线（或线段）L 上的两点为 P1,P2。

1. 如果 L 是线段且 P1，P2 都包含在圆 O 内，则没有交点；否则进行下一步。
2. 如果 L 平行于 Y 轴，
   a) 计算圆心到 L 的距离 dis；
   b) 如果 dis > r 则 L 和圆没有交点；
   c) 利用勾股定理，可以求出两交点坐标，但要注意考虑 L 和圆的相切情况。
3. 如果 L 平行于 X 轴，做法与 L 平行于 Y 轴的情况类似；

4. 如果 L 既不平行 X 轴也不平行 Y 轴，可以求出 L 的斜率 K，然后列出 L 的点斜式方程，和圆方程联立即可求解出 L 和圆的两个交点；

5. 如果 L 是线段，对于 2，3，4 中求出的交点还要分别判断是否属于该线段的范围内。

28. 凸包的概念：

点集 Q 的凸包(convex hull)是指一个最小凸多边形，满足 Q 中的点或者在多边形边上或者在其内。下图中由红色线段表示的多边形就是点集 Q={p0,p1,...p12}的凸包。

29. 凸包的求法：

现在已经证明了凸包算法的时间复杂度下界是 O(n*logn),但是当凸包的顶点数 h 也被考虑进去的话，Krikpatrick 和 Seidel 的剪枝搜索算法可以达到 O(n*logh)，在渐进意义下达到最优。最常用的凸包算法是 Graham 扫描法和 Jarvis 步进法。本文只简单介绍一下 Graham 扫描法，其正确性的证明和 Jarvis 步进法的过程大家可以参考《算法导论》。

对于一个有三个或以上点的点集 Q，Graham 扫描法的过程如下：

令 p0 为 Q 中 Y-X 坐标排序下最小的点

设<p1,p2,...pm>为对其余点按以 p0 为中心的极角逆时针排序所得的点集（如果有多个点有相同的极角，除了距 p0 最远的点外全部移除

压 p0 进栈 S

压 p1 进栈 S

压 p2 进栈 S

for i ← 3 to m

do while 由 S 的栈顶元素的下一个元素、S 的栈顶元素以及 pi 构成的折线段不拐向左侧

对 S 弹栈

压 pi 进栈 S

return S;

此过程执行后，栈 S 由底至顶的元素就是 Q 的凸包顶点按逆时针排列的点序列。需要注意的是，我们对点按极角逆时针排序时，并不需要真正求出极角，只需要求出任意两点的次序就可以了。而这个步骤可以用前述的矢量叉积性质实现。

四、结语

尽管人类对几何学的研究从古代起便没有中断过，但是具体到借助计算机来解决几何问题的研究，还只是停留在一个初级阶段，无论从应用领域还是发展前景来看，计算几何学都值得我们认真学习、加以运用，希望这篇文章能带你走进这个丰富多彩的世界。

目录

```
/* 需要包含的头文件 */
#include <cmath >

/* 常用的常量定义 */
const double INF = 1E200
const double EP = 1E-10
const int MAXV = 300
const double PI = 3.14159265

/* 基本几何结构 */
struct POINT
{
double x;
double y; POINT(double a=0, double b=0) { x=a; y=b;} //constructor
};
struct LINESEG
{
POINT s;
POINT e; LINESEG(POINT a, POINT b) { s=a; e=b;}
LINESEG() { }
};
struct LINE // 直线的解析方程 a*x+b*y+c=0 为统一表示，约定 a >= 0
{
double a;
double b;
double c; LINE(double d1=1, double d2=-1, double d3=0) {a=d1; b=d2; c=d3;}
};
```

```
/*******************\
* *
* 点的基本运算 *
* *
\*******************/

double dist(POINT p1,POINT p2) // 返回两点之间欧氏距离
{
return( sqrt( (p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y) ) );
}
bool equal_point(POINT p1,POINT p2) // 判断两个点是否重合
{
return ( (abs(p1.x-p2.x)<EP)&&(abs(p1.y-p2.y)<EP) );
}

/*****************************************************************************
r=multiply(sp,ep,op),得到(sp-op)*(ep-op)的叉积
r>0:ep 在矢量 opsp 的逆时针方向；
r=0：opspep 三点共线；
r<0:ep 在矢量 opsp 的顺时针方向
*****************************************************************************/

double multiply(POINT sp,POINT ep,POINT op)
{
return((sp.x-op.x)*(ep.y-op.y)-(ep.x-op.x)*(sp.y-op.y));
}

/*****************************************************************************
r=dotmultiply(p1,p2,op),得到矢量(p1-op)和(p2-op)的点积，如果两个矢量都非零矢量
r<0:两矢量夹角为锐角；r=0：两矢量夹角为直角；r>0:两矢量夹角为钝角
*****************************************************************************/
double dotmultiply(POINT p1,POINT p2,POINT p0)
{
return ((p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y));
}

/* 判断点 p 是否在线段 l 上，条件：(p 在线段 l 所在的直线上)&& (点 p 在以线段 l 为对角线的矩形内) */
bool online(LINESEG l,POINT p)
{
return((multiply(l.e,p,l.s)==0)
&&( ( (p.x-l.s.x)*(p.x-l.e.x)<=0 )&&( (p.y-l.s.y)*(p.y-l.e.y)<=0 ) ) );
}

// 返回点 p 以点 o 为圆心逆时针旋转 alpha(单位：弧度)后所在的位置
```

```
POINT rotate(POINT o,double alpha,POINT p)
{
POINT tp;
p.x-=o.x;
p.y-=o.y;
tp.x=p.x*cos(alpha)-p.y*sin(alpha)+o.x;
tp.y=p.y*cos(alpha)+p.x*sin(alpha)+o.y;
return tp;
}

/* 返回顶角在 o 点，起始边为 os，终止边为 oe 的夹角(单位：弧度)
角度小于 pi，返回正值
角度大于 pi，返回负值
可以用于求线段之间的夹角
*/
double angle(POINT o,POINT s,POINT e)
{
double cosfi,fi,norm;
double dsx = s.x - o.x;
double dsy = s.y - o.y;
double dex = e.x - o.x;
double dey = e.y - o.y;

cosfi=dsx*dex+dsy*dey;
norm=(dsx*dsx+dey*dey)*(dex*dex+dey*dey);
cosfi /= sqrt( norm );

if (cosfi >= 1.0 ) return 0;
if (cosfi <= -1.0 ) return -3.1415926;

fi=acos(cosfi);
if (dsx*dey-dsy*dex>0) return fi; // 说明矢量 os 在矢量 oe 的顺时针方向
return -fi;
}




/*****************************\
* *
* 线段及直线的基本运算 *
* *
\*****************************/
```

```
/* 判断点与线段的关系,用途很广泛
本函数是根据下面的公式写的，P 是点 C 到线段 AB 所在直线的垂足

AC dot AB
r = ---------
||AB||^2
(Cx-Ax)(Bx-Ax) + (Cy-Ay)(By-Ay)
= -------------------------------
L^2

r has the following meaning:

r=0 P = A
r=1 P = B
r<0 P is on the backward extension of AB
r>1 P is on the forward extension of AB
0<r<1 P is interior to AB
*/
double relation(POINT p,LINESEG l)
{
LINESEG tl;
tl.s=l.s;
tl.e=p;
return dotmultiply(tl.e,l.e,l.s)/(dist(l.s,l.e)*dist(l.s,l.e));
}

// 求点 C 到线段 AB 所在直线的垂足 P
POINT perpendicular(POINT p,LINESEG l)
{
double r=relation(p,l);
POINT tp;
tp.x=l.s.x+r*(l.e.x-l.s.x);
tp.y=l.s.y+r*(l.e.y-l.s.y);
return tp;
}
/* 求点 p 到线段 l 的最短距离,并返回线段上距该点最近的点 np
注意：np 是线段 l 上到点 p 最近的点，不一定是垂足 */
double ptolinesegdist(POINT p,LINESEG l,POINT &np)
{
double r=relation(p,l);
if(r<0)
{
np=l.s;
```

```
return dist(p,l.s);
}
if(r>1)
{
np=l.e;
return dist(p,l.e);
}
np=perpendicular(p,l);
return dist(p,np);
}

// 求点 p 到线段 l 所在直线的距离,请注意本函数与上个函数的区别
double ptoldist(POINT p,LINESEG l)
{
return abs(multiply(p,l.e,l.s))/dist(l.s,l.e);
}

/* 计算点到折线集的最近距离,并返回最近点.
注意：调用的是 ptolineseg()函数 */
double ptopointset(int vcount,POINT pointset[],POINT p,POINT &q)
{
int i;
double cd=double(INF),td;
LINESEG l;
POINT tq,cq;

for(i=0;i<vcount-1;i++)
{
l.s=pointset[ i ];
l.e=pointset[i+1];
td=ptolinesegdist(p,l,tq);
if(td<cd)
{
cd=td;
cq=tq;
}
}
q=cq;
return cd;
}
/* 判断圆是否在多边形内.ptolineseg()函数的应用 2 */
bool CircleInsidePolygon(int vcount,POINT center,double radius,POINT polygon[])
{
POINT q;
```

```
double d;
q.x=0;
q.y=0;
d=ptopointset(vcount,polygon,center,q);
if(d<radius||fabs(d-radius)<EP)
return true;
else
return false;
}

/* 返回两个矢量 l1 和 l2 的夹角的余弦(-1 --- 1)注意：如果想从余弦求夹角的话，注意反余弦函数的定义域是从 0 到 pi */
double cosine(LINESEG l1,LINESEG l2)
{
return (((l1.e.x-l1.s.x)*(l2.e.x-l2.s.x) +
(l1.e.y-l1.s.y)*(l2.e.y-l2.s.y))/(dist(l1.e,l1.s)*dist(l2.e,l2.s))) );
}
// 返回线段 l1 与 l2 之间的夹角 单位：弧度 范围(-pi，pi)
double lsangle(LINESEG l1,LINESEG l2)
{
POINT o,s,e;
o.x=o.y=0;
s.x=l1.e.x-l1.s.x;
s.y=l1.e.y-l1.s.y;
e.x=l2.e.x-l2.s.x;
e.y=l2.e.y-l2.s.y;
return angle(o,s,e);
}
// 如果线段 u 和 v 相交(包括相交在端点处)时，返回 true
bool intersect(LINESEG u,LINESEG v)
{
return( (max(u.s.x,u.e.x)>=min(v.s.x,v.e.x))&& //排斥实验
(max(v.s.x,v.e.x)>=min(u.s.x,u.e.x))&&
(max(u.s.y,u.e.y)>=min(v.s.y,v.e.y))&&
(max(v.s.y,v.e.y)>=min(u.s.y,u.e.y))&&
(multiply(v.s,u.e,u.s)*multiply(u.e,v.e,u.s)>=0)&& //跨立实验
(multiply(u.s,v.e,v.s)*multiply(v.e,u.e,v.s)>=0));
}


// (线段 u 和 v 相交)&&(交点不是双方的端点) 时返回 true
bool intersect_A(LINESEG u,LINESEG v)
{
return((intersect(u,v))&&
```

```
(!online(u,v.s))&&
(!online(u,v.e))&&
(!online(v,u.e))&&
(!online(v,u.s)));
}


// 线段 v 所在直线与线段 u 相交时返回 true；方法：判断线段 u 是否跨立线段 v
bool intersect_l(LINESEG u,LINESEG v)
{
return multiply(u.s,v.e,v.s)*multiply(v.e,u.e,v.s)>=0;
}


// 根据已知两点坐标，求过这两点的直线解析方程： a*x+b*y+c = 0 (a >= 0)
LINE makeline(POINT p1,POINT p2)
{
LINE tl;
int sign = 1;
tl.a=p2.y-p1.y;
if(tl.a<0)
{
sign = -1;
tl.a=sign*tl.a;
}
tl.b=sign*(p1.x-p2.x);
tl.c=sign*(p1.y*p2.x-p1.x*p2.y);
return tl;
}

// 根据直线解析方程返回直线的斜率 k,水平线返回 0,竖直线返回 1e200
double slope(LINE l)
{
if(abs(l.a) < 1e-20)return 0;
if(abs(l.b) < 1e-20)return INF;
return -(l.a/l.b);
}

// 返回直线的倾斜角 alpha ( 0 - pi)
double alpha(LINE l)
{
if(abs(l.a)< EP)return 0;
if(abs(l.b)< EP)return PI/2;
double k=slope(l);
```

```
if(k>0)
return atan(k);
else
return PI+atan(k);
}

// 求点 p 关于直线 l 的对称点
POINT symmetry(LINE l,POINT p)
{
POINT tp;
tp.x=((l.b*l.b-l.a*l.a)*p.x-2*l.a*l.b*p.y-2*l.a*l.c)/(l.a*l.a+l.b*l.b);
tp.y=((l.a*l.a-l.b*l.b)*p.y-2*l.a*l.b*p.x-2*l.b*l.c)/(l.a*l.a+l.b*l.b);
return tp;
}

// 如果两条直线 l1(a1*x+b1*y+c1 = 0), l2(a2*x+b2*y+c2 = 0)相交，返回 true，且返回交点 p
bool lineintersect(LINE l1,LINE l2,POINT &p) // 是 L1，L2
{
double d=l1.a*l2.b-l2.a*l1.b;
if(abs(d)<EP) // 不相交
return false;
p.x = (l2.c*l1.b-l1.c*l2.b)/d;
p.y = (l2.a*l1.c-l1.a*l2.c)/d;
return true;
}

// 如果线段 l1 和 l2 相交，返回 true 且交点由(inter)返回，否则返回 false
bool intersection(LINESEG l1,LINESEG l2,POINT &inter)
{
LINE ll1,ll2;
ll1=makeline(l1.s,l1.e);
ll2=makeline(l2.s,l2.e);
if(lineintersect(ll1,ll2,inter))
{
return online(l1,inter);
}
else
return false;
}
```

可以应付大部分 zoj 上的几何题 Sample Text**Sample Text**

zoj 上的计算几何题

Vol I

1010 by pandahyx

1032 by javaman
1037 by Vegetable Bird
1041 by javaman
1081 by Vegetable Bird
1090 by Vegetable Bird

Vol II
1104 by javaman
1123 by javaman
1139 by Vegetable Bird
1165 by javaman
1199 by Vegetable Bird

Vol V
1426 by Vegetable Bird
1439 by Vegetable Bird
1460 by Vegetable Bird
1472 by Vegetable Bird

Vol VI
1597 by Vegetable Bird

Vol VII
1608 by Vegetable Bird
1648 by Vegetable Bird

Vol XII
2102 by pandahyx
2107 by pandahyx
2157 by pandahyx

Vol XIII
2234 by pandahyx

Vol XIV
2318 by ahyangyi
2394 by qherlyt

Vol XV
2403 by Vegetable Bird