数据结构PJ实验报告

周煜敏 12307130088

一、程序运行环境： CPU: Intel Core 2 Duo T6500 (2.1GHz)

内存：4G

操作系统：windows 7（64位）

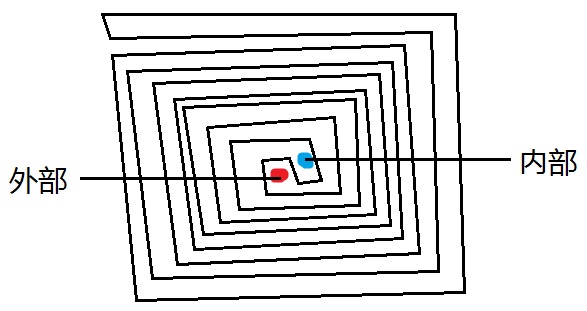
二、所用数据结构：数组

理由：没有大量插入删除操作，需要快速的元素访问，顶点、目标点坐标按自然数顺序排列。因此选择数组作为存储方式。

三、算法

所用算法1：射线法判断交点个数

依据：拓扑学中的若当曲线定理：平面上一条简单闭曲线 C 恰好可以把平面分成两个区域，一个是内部，一个是外部。换句话说，平面上的点被分为了两类：在曲线外部的点集 A，和在曲线内部的点集 B 。在同一点集中的任意两点都可以用一条不与 C 相交的曲线相连，而连接一对不属于同一点集的两点的连线必然和 C 相交。因此平面中两点的连线，若属于A，B中同一集合，则一定经过多边形偶数条边，反之则经过奇数条边。



引出从目标点以水平方向直至正无穷远处的射线，

1. 在不经过多边形顶点的情况下，交点数为奇数时，目标点位于多边形内。反之，交点数为偶数时，目标点位于多边形外。

求交点的实现方式：

对于每个目标点遍历多边形的所有边PiPi+1

Pi（x1,y1）与Pi+1（x2,y2）所构成的直线方程为

(y-y1)(x2-x1) = (x-x1)(y2-y1) ……

目标点Q（x0,y0）所引出的与x轴平行的射线为

y=y0(x>x0) ……

代入得

交点横坐标为

x = (y0-y1)(x2-x1)/(y2-y1) + x1 （x>x0）……（\*）

射线和多边形的边的拓扑关系有以下几种情况：

1. 射线不经过多边形的顶点和边

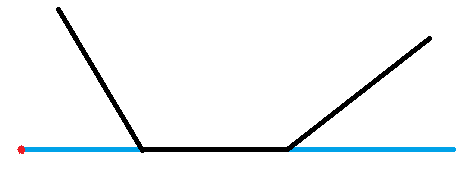
按照公式（\*）计算是否为交点

1. 特殊情况1：与边重合

I．目标点落在多边形边上，不符合条件

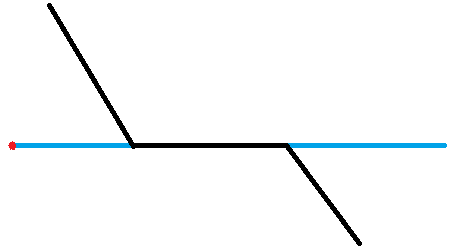
II．

未改变内与外的性质，算两个交点



III．

改变了内与外的性质，算一个交点

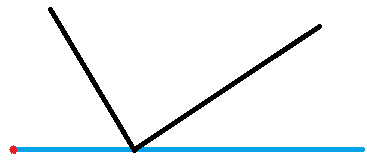


1. 特殊情况2：与顶点重合

I．目标点落在多边形顶点上，不符合条件

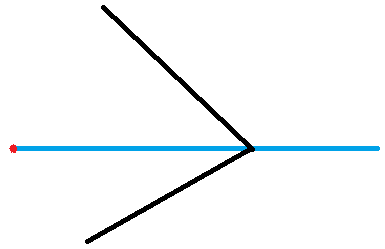
II．

未改变内与外的性质，不算交点



III．

改变了内与外的性质，算一个交点



优化：

1. 外接矩形

对于任意的多边形，如果目标点位于外接矩形之外，则必然位于多边形之外。而外接矩形能够在预处理时简单通过记录多边形横纵坐标的最值确定。经测试，这一优化在给定样例中可提速50%左右。但优化的优劣取决于测试样例中可优化点的比例。

1. 没有交点的三个简单判断
2. 如果Pi 与Pi+1的横坐标x1，x2都比目标点Q的横坐标x0小的话，则射线必然与PiPi+1没有交点。
3. 如果Pi 与Pi+1的纵坐标y1，y2都比目标点Q的纵坐标y0小的话，则射线必然与PiPi+1没有交点。
4. 如果Pi 与Pi+1的纵坐标y1，y2都比目标点Q的纵坐标y0大的话，则射线必然与PiPi+1没有交点。

同样，这样的优化与多边形的形状和目标点的比例有关，但至少这样的优化对于一般多边形内的点，可大大缩小需要计算（\*）式的次数。

所用算法2：网格法

运用分治法，将外接矩形等分成1000\*1000(dens=1000)的小矩形网格。则网格与多边形的关系有以下三种：

1. 多边形无边与网格相交，网格整体在多边形外，标记为0
2. 多边形无边与网格相交，网格整体在多边形内，标记为1
3. 多边形有边与网格相交，标记为2

由于1000\*1000的切分较细，大多数网格都标记了0,1，从统计角度上说，仅有多边形外部轮廓处在的网格才标记为2，因此最终复杂度会接近O（m）

显然切分越细，与处理时间就越长，但速度也就越快。选取1000是因为考虑到精度问题，若切分更细，则每次计算会除以更大的数，带来更大的误差，也许会造成错误。

接下来唯一要解决的事就是确定目标点所在的网格的位置，因为横纵坐标等分，所以寻找网格位置仅需要时间复杂度为O（1）的数学计算即可。若网格整体在多边形外，则显然网格内的点（包括边界）也在多边形外，反之亦然。

两者的结合：在网格法生效，即网格与多边形无交点时，直接运用预处理的网格结果；否则采用射线法再完整的做一遍。

四、可能存在的问题与不足：

精度！！由于测试数据有限，自己随机生成的点列和多边形也很难判断某些边界条件判断时可能存在的错误。此外对于题目数据中给出的vmax和每个点的时间性质没有加以利用，若能想到如何利用，也许程序会更优化。

五、复杂度：

最差时间复杂度O（mn），最好时间复杂度O（m），平均时间复杂度接近O（m），空间复杂度为O（m）。预处理时间复杂度O（n\*dens2）

六、测试方法：

我自己写了一个C的小程序来判断文件中的前N行是否相等，这样能够准确判断算法是否正确并及时发现错误。

#include<stdio.h>

#include<string.h>

int main()

{

FILE \*f1,\*f2;

int i,row=1,flag=1;

char str1[6];

char str2[6];

while (row>0)

{

if ( ( f1 = fopen("D:\\Result.txt", "r") ) == NULL )

{

printf("\nCannot open Results.txt!\n");

return 1;

}

if ( ( f2 = fopen("D:\\key.txt", "r") ) == NULL )

{

printf("\nCannot open key.txt!\n");

return 1;

}

printf("输入查询行数：\n");

scanf("%d",&row);

for (i=0;i<row;i++)

{

fscanf(f1,"%s",str1);

fscanf(f2,"%s",str2);

if (strcmp(str1,str2)!=0)

{

printf("%d\tWRONG\n",i);

flag=0;

fclose(f1);

fclose(f2);

break;

}

}

fclose(f1);

fclose(f2);

if (flag) printf("RIGHT\n");

continue;

}

}

七、其他算法的尝试与比较：

1. 曾思考过对平面和多边形进行梯形索引（如图）并进行树状二分查找，其时间复杂度为O（mlog2n），但是测试样例所给出的边数n=100，log2n约为8，加之我的射线算法中有许多可直接进行下一循环的条件，实际不一定比需要较为复杂指针移动操作的建立索引算法慢。

2. 本想对于多边形所有的边按纵坐标进行排序，将多边形进行水平切分记录每个切分中的边。在根据二分查找找到目标点所在区域，在运用射线法求解。这样的算法无需扫描所有的边，而只需扫描极少的边就能得出答案。但是这仍然是一个O（mlog2n）的算法，常数优化依然是个问题，此外当有边与x轴平行时，又不得不采用其他方法，效率会降低。

参考论文：

1. 王燕平, 刘永和. 射线法判断平面中的点在多边形内外的算法[J]. 山西建筑, 2007, 33(33): 364-365.
2. 郭雷, 王洵, 王晓蒲. 有向回路法和网格法: 多边形内外点判别的新算法[J]. 计算机工程与应用, 2002, 19(2): 119-122.