QPOINT 题解

长春吉大附中实验学校 吴一凡

November 10, 2015

1 题目大意

在二维平面上，给定 n 个两两之间没有交点的简单多边形，有 q 个询问，每次给定一个点，要求判断这个点在哪个简单多边形中，或者判断这个点不在任何一个多边形中。
要求强制在线。
数据范围 $n \le 10^5$，所有多边形的总点数 $3 \le k \le 3 \times 10^5$，$q \le 10^5$。

2 算法讨论

2.1 算法 1

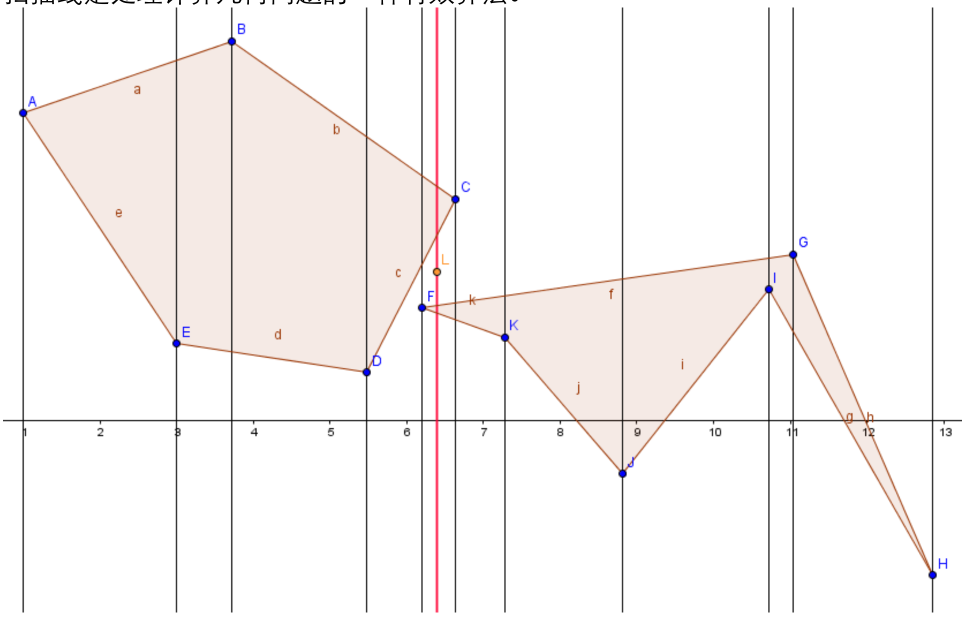
考虑如何判定一个点是否在一个简单多边形中。
令多边形上有 k 个点，我们可以利用射线法或者转角法做到 $O(k)$。
对于每组询问，我们都利用这两种方法之一来判定，时间复杂度 $O(qk)$。

2.2 算法 2

对于只有凸多边形的情况，我们可以利用更加快速的方法来判定一个点是否在这个凸多边形中。
我们选定一个在凸多边形内部的点 p，并维护凸多边形上所有点关于点 p 的极角序。
那么对于询问点，我们找到询问点以点 p 为原点的极角，并在极角序中找到这个极角的前驱、后继点 x, y，那么我们能够发现若询问点能够在凸多边形内部，必定有询问点在三角形 pxy 中。反之一定不在。
这样我们只需要 $O(1)$ 判定一个点在不在一个三角形中就行了。
于是我们 $O(\log k)$ 就能知道一个点在不在凸多边形中。
因此如果都是凸多边形，时间复杂度就能做到 $O(qn \log k)$。
但可惜的是，这个方法显然不能对凹多边形适用；而如果将这个凸多边形拆分成三角形处理也是没有意义的。

2.3 算法 3

扫描线是处理计算几何问题的一种有效算法。



我们对于每个不同的横坐标做一条垂直于 x 轴的垂线。
我们可以发现，无论多边形凸凹与否，在两条相邻的垂线之间，都会有一些只可能在左右两端才相交的线段，这些线段的依据上下的位置关系构成了一个有序序列。
令两条相邻的垂线的横坐标分别为 l, r，则在两条垂线中间夹着的线段我们可以用 $x = \frac{l+r}{2}$ 时的纵坐标值的大小作为顺序。
我们只考虑原多边形的横坐标增大的线段，并记录这个线段下面的区域的编号，如果是某个多边形内部的区域，则编号为这个多边形的编号；否则编号为空区域。
对于询问 (x, y)，我们只需二分找到 x 在哪两条相邻垂线之间，然后再在这个对应的线段序列中找到横坐标等于 x 时的纵坐标 $\geq y$ 且纵坐标最小的线段（其实也就是后继），直接看看这条线段下面的区域是什么就行了。
现在我们只要知道怎么维护这些线段就行了。
如果允许离线，那我们可以将所有的询问按照横坐标从小到大排序，同时用一颗平衡树维护所有的线段：我们对于每条线段处理出出现时间和消失时间，依次处理每两条相邻垂线之间的线段，在出现时间将对应的线段插入平衡树，在删除时间将对应的线段从平衡树中删除，这样每条线段只会被增删各一次，时间复杂度 $O(klogk)$。
每处理完相邻两条垂线之间的线段，我们回答横坐标也处于这两条垂线之间的询问。
这样做的时间复杂度为 $O(k \log k + q \log q)$，空间复杂度 $O(k)$。

2.4　算法 4

将上面的离线算法改成在线算法，只需要将用来维护线段的平衡树可持久化即可。
我们使用函数式 Treap，使用两个操作 Merge 和 Split 分别表示将两个有序序列前后拼接在一起，以及将一个有序序列分离成前后两个大小给定的有序序列，这两个操作互为逆过程。
使用函数式实现，由于 Treap 需要满足随机权值的堆性质，深度期望为 $O(\log n)$，因此两个操作的时间复杂度均为期望 $O(\log n)$，消耗空间也为期望 $O(\log n)$。
我们利用上面两个操作，也很容易实现平衡树的可持久化插入、删除，且时间复杂度期望 $O(\log n)$，消耗空间期望 $O(\log n)$。
因此，我们将平衡树可持久化就能满足这道题目的需求，时间复杂度 $O(k \log k + q \log k)$，空间复杂度 $O(k \log k)$。

3　简单多边形生成算法

3.1　凸多边形

3.1.1　算法 1

直接在平面上随机生成若干个点，求出凸包，就得到了一个凸多边形。
这样做的问题是凸包上的点数期望非常少，一般不会超过 100。

3.1.2　算法 2

随机生成一个圆，在圆的边界上生成若干个点，将这些点关于以圆心为原点的极角序排序并顺次连接，显然能够形成一个凸多边形。
这样就能生成点数为 10^5 级别的凸多边形了。

3.2　凹多边形

3.2.1　算法 1

在平面上随机生成一个点集，在点集的凸包上找一对对踵点，将这两个点组成的直线称为划分线。
不妨将对踵点设为 p, q。
对于向量 $\vec{pq}$ 左侧的点，我们尝试生成一条 $p \to q$ 且遍历所有向量左侧的点的路径。
对于向量 $\vec{qp}$ 左侧的点，我们尝试生成一条 $q \to p$ 且遍历所有向量左侧的点的路径。
最后只需要将两条路径合并即可。
考虑如何生成路径:
对于向量 $\vec{pq}$ 左侧的点，对于点集中的所有点 x，我们用向量 $\vec{px}$ 在向量 $\vec{pq}$ 上的投影从小到大排序。随后只要将这些点顺次连接就行了。

对于向量 $\vec{pq}$ 左侧的点同理。
这样我们就生成了一个简单多边形。
时间复杂度 $O(n \log n)$。

3.3　算法 2

在平面上随机生成一个点集，再随机生成一个点，将点集按照以这个点为坐标系原点的极角序排序，若极角序相同则按照距离从小到大排序。
要注意，随机生成的点最好在点集的凸包内，不然可能就会出现多边形自交的情况。
这样同样将所有的点顺次连接即可。
时间复杂度 $O(n \log n)$。

3.4　算法 3

考虑利用分治法求解。
定义过程 $solve(S, be, en, R)$ 表示对于点集 S，be, en 是 S 中的两个点，现在要生成一个从 be 出发到 en 结束且包括 S 中所有点的路径，使得所有的边只在端点处相交，并将这条路径存储在 R 中。
考虑递归边界：若 $S = be, en$，则显然直接返回 $R = be \to en$ 即可。
否则我们随机在 $S - \{be, en\}$ 中选出一个点 r 来，再随机在线段 be, en 上选出一点 r' 来，我们将 $S - \{be, en, r\}$ 按照在射线 $r'r$ 的左侧还是右侧分为两个集合 S_l, S_r。
我们可以发现，这样原来我们要求的路径变成了两部分，首先是从 be 出发到 r 的路径，然后是从 r 出发到 en 的路径。而这两部分显然都能利用刚才的过程递归解决，且如果子过程得到的路径不交，由于两侧的点集只在 r 处存在交集，显然最终得到的路径也是不交的。
那么我们如何生成多边形呢?
一开始我们随机选出两个点 p, q，分别利用分治求出线段 pq 两侧的从 p 到 q 和从 q 到 p 的两条路径，最后再进行合并就行了。
考虑时间复杂度：生成一个点数为 n 的多边形，期望时间复杂度为 $O(n \log n)$，最坏时间复杂度为 $O(n^2)$。
生成的多边形看起来还是很一般的：