矩形区域 N 点的快速凸包求解

问题描述

在矩形区域内存在 N 个点，从中选取最少的点组成的多边形可包围住矩形区域内的其他所有点。一个更形象的例子为，一个木板上钉有 N 个钉子，在这些钉子围成的多边形外端使用一根橡皮筋套住，当松开橡皮筋后橡皮筋会由于弹性回缩固定在一些顶点上，“囊括”所有的钉子，这些固定的顶点就是要求的组成凸包的点，如图 1 所示。

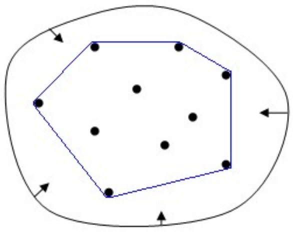

图 1. 凸包形成的示例

实现思路

可以观察发现，**1.凸包上的相邻两边夹角不超过 180°，2.坐标横纵分量最大或小的点一定在凸包上**。基于这两点发现，可以将凸包分割成上凸包和下凸包两部分，两者的划分基于横坐标最小（最左端的点）和最大的点（最右端的点）的连线进行划分。

对所有的点按照横坐标-纵坐标的优先级进行升序排序，排序后的结果中，第一个和最后一个点就是上下凸包划分的基准。得到划分后，下凸包一定是从最小值按逆时针一直“左拐”直到最大值，上凸壳一定是从最大值“左拐”到最小值，因此我们可以升序枚举求出下凸壳，然后降序求出上凸壳。

判断一个点是否为凸包上的点，需要额外依赖它前面的两个点，需要判断新的点是否在前两个点连线的逆时针方向，具体而言是判断他们的连线的斜率大小。如图 2 所示给出一个判别示例。

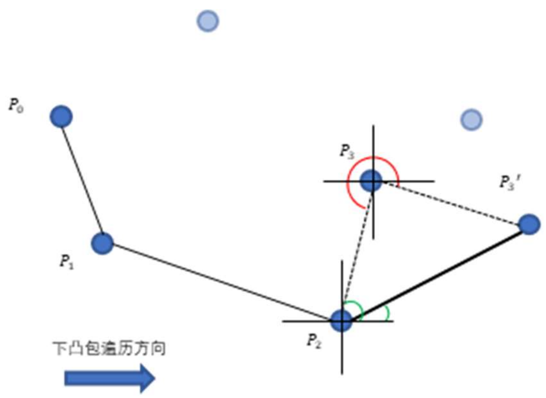

图 2. 判断凸包点的示例

在下凸包中，遍历顺序为 P0-P1-P2-P3-P3’，在判断 P3 是否为凸包上的点时，可见 P3 的前一个点 P2 与 P3 连线以及 P3 的前两个点 P1 与 P2 的连线相比，前者的斜率更大，因此暂且认为 P3 为凸包上的点；然而遍历到 P3’时，其与上一个点 P3 形成的连线比其与上两个点 P2 和 P3 形成的连线斜率相比斜率更小，因此 P3 不是凸包上的点，暂且考虑 P3’作为凸包上的点进行下一轮的判断，把 P3’的索引纳入一个栈中进行存储。

在这个方法中不需要考虑共线的点，因为共线的情况已经在排序的过程中进行处理了。在处理完下凸包后，又要倒序地遍历所有节点，重复判断的操作直到回到初始的最左边的点上，即可求出上凸包。最后仅需注意去除上凸包的栈中的最后一个元素，因为它在开始遍历的时候已经使用过一次了。

最后将队列中的元素作为索引，在点集中取出即为凸包上的点。具体的代码实现见附录代码 tubao.py。

性能分析

时间复杂度：$O(nlogn)$，其中 n 为数组的长度。首先需要对数组进行排序，时间复杂度为 $O(nlogn)$，每次添加栈中添加元素后，判断新加入的元素是否在凸包上，因此每个元素都可能进行入栈与出栈一次，最多需要的时间复杂度为 $O(2n)$，因此总的时间复杂度为 $O(n\,log\,n)$。

空间复杂度：$O(n)$，其中 n 为数组的长度。首先该解法需要快速排序，需要的栈空间为 $O(log\,n)$，用来标记元素是否存在重复访问的空间复杂度为 $O(n)$，需要栈来保存当前判别的凸包上的元素，栈中最多有 n 个元素，所需要的空间为 $O(n)$，因此总的空间复杂度为 $O(n)$。

在 test.py 中，设定了测试环境，对 n=1000~10000 进行了时间测试，测试的结果如图 3 所示。可见时间随 n 的规模增加基本呈现线性关系，而不是$nlogn$的曲线趋势，分析原因为此时的 n 还比较小，而时间记录的颗粒度无法达到此场景下$logn$级别的精度，因此主要呈现的是接近线性的关系。

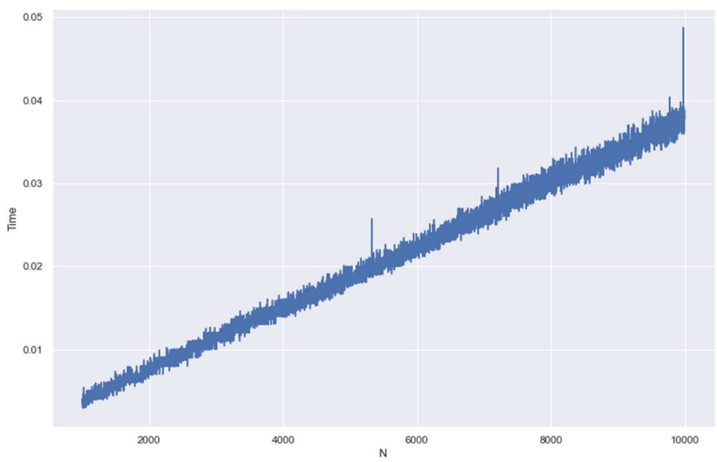

图 3. 实际测试时间情况

参考文献

1. 587. 安装栅栏 题解 - 力扣（LeetCode）