# 计算几何课程作业实验报告

——二维空间上的区间树构建与查询实现

姓名：贺磊

学号：BY1906008

## 1. 简介

该课程作业实现了如图 1所示的二维空间上的区间查找树，并能够在构建好的区间树上进行快速的区间查询，时间复杂度为O(log2 n+k)，在区间查询的数量很大时，会取得比较好的效果。

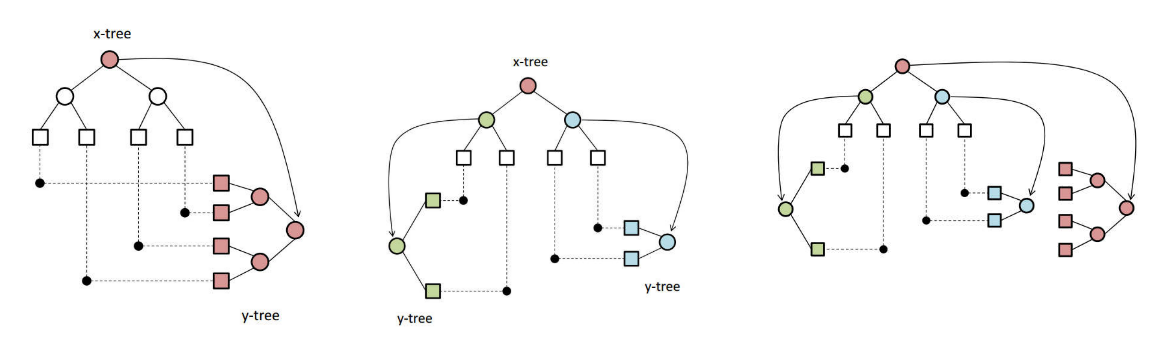


图 1 二维空间上区间查找树示意图

## 2.实现方法

### 2.1 数据结构

我们使用叶节点来表示二维空间上的点。代码表述如图 2所示：

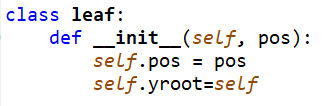


图 2 叶节点数据结构

中间节点指向叶节点与中间节点，用于构建区间树。中间节点中包含如下信息：以该节点为根节点的子树中所有叶节点的区间信息，指向左子树的指针，指向右子树的指针，指示区间树中位数的叶节点（相应维度）的值。中间节点类别定义的代码表述如图 3所示：

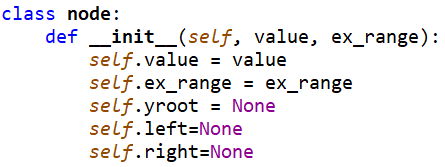


图 3 中间节点数据结构

本课程作业中的二维空间上的区间树的构建方法为，先构建在x轴上的区间树，对于x维度上的区间树中的每个非叶节点，构建以该节点为根节点的子树上所有叶节点集合的在y维度上的区间树，并在该x维度区间树中间节点上设置指向y维度上区间树的指针。

### 2.2 区间树构建方法

区间树的构建主要分为一下几个部分：在x轴上的区间树构建，为x轴上的区间树中间节点构建以y轴坐标为标准的区间树

#### 2.2.1 x轴上的区间树构建

在本节我们介绍在x轴上区间树的构建方法。

首先，我们对点集中的点按照x坐标进行排序。这样做方便在x轴上的区间树构建。我们使用递归的方法，首先算出点集中点在x轴上坐标的中位数，并将点集分为两个部分，分别对这两个部分进行x轴上的区间树构建。数据结构如2.1中所介绍。

#### 2.2.2 y轴上区间树的构建

在本节我们介绍在y轴上区间树的构建方法。

以中间节点为根的子树中包含的叶节点作为点集整体的一个子集，y轴上的区间树建立在这个子集上。y轴上区间树的构建可以分为三个步骤：首先，提取出点集子集中的所有叶节点；之后，这些叶节点按照y坐标进行归并排序；最后，利用排好序的叶节点集合构建y轴上的区间树。构建y轴上的区间树的代码框架如下：

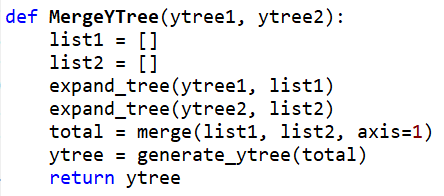


图 4 构建y轴上区间树的代码框架

#### 2.2.3 区间树整体构建方式

区间树的构建如课件中的伪代码所示：

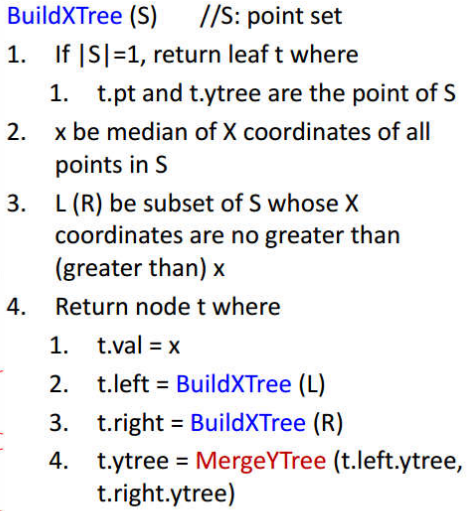


图 5 区间树构建的伪代码

### 2.3 查询方法

区间树构建完成后，我们可以快速地查询分布在指定区域内的点集，不用每次遍历点集中所有的点。

区间查询的伪代码如下所示：

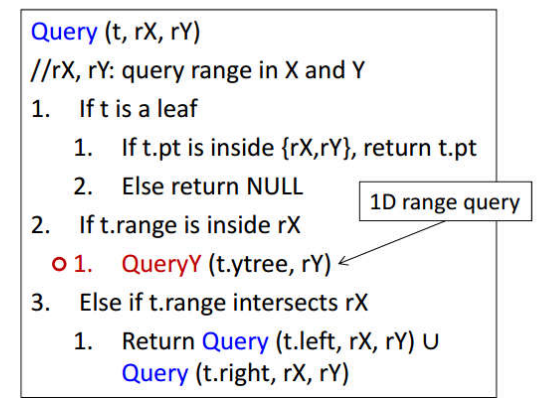


图 6区间查询伪代码

该方法使用递归的方式进行区间查询。如果是叶节点，那么直接判断是否在区间中；如果是非叶节点，如果以该节点为根节点的子树在x维度上的范围完全被包含在查询范围内，那么，对该节点的y区间树进行区间查询；如果该节点为根节点的子树在x维度上的范围与查询范围交叉，那么对该节点的左右子树分别进行查询。

## 3.实验

### 3.1 数据

本课程论文中的数据采用随机生成的分布在二维平面上的点集。实现过程中，由于二维空间上的点能够被映射到二维空间中的指定方形区域中。因此，在这个实验中，随机生成的点集在x轴与y轴上都分布在[-1, 1] 的范围内。

在压缩包中我们提供多个已经生成的点集，分别包含20000、40000、70000个点。可以在运行时直接加载。（见文件20k.npy, 40k.npy, 70k.npy）

### 3.2 正确性

为了保证正确性，我们提供了一个对比实验方法，该方法简单地遍历整个点集，判断每个点是否在查询区间内。我们对比两个查询结果点的数量是否相同。实验结果如下。

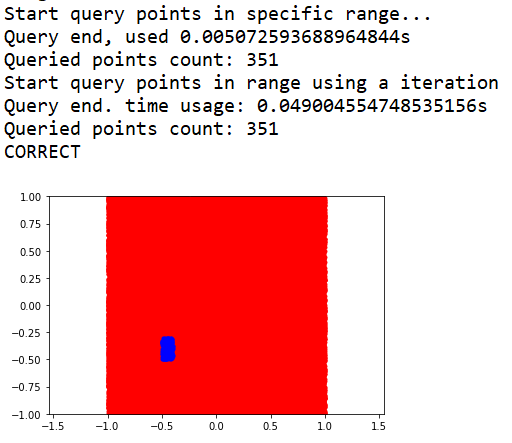


图 7 对比实验及正确性展示

如图 6所示，查询出来在区域内的点用蓝色点来表示，而其他没在查询区间的点用红色来表示。在后面的实验中我们也将使用这种方法来展示查询结果。查询出来的点集中分布在查询区域内。并且，两个方法中区间查询的结果是相同的，查找到的点的数量也相同，在这个实验中均为351个点。

更多的实验结果如图 8所示。

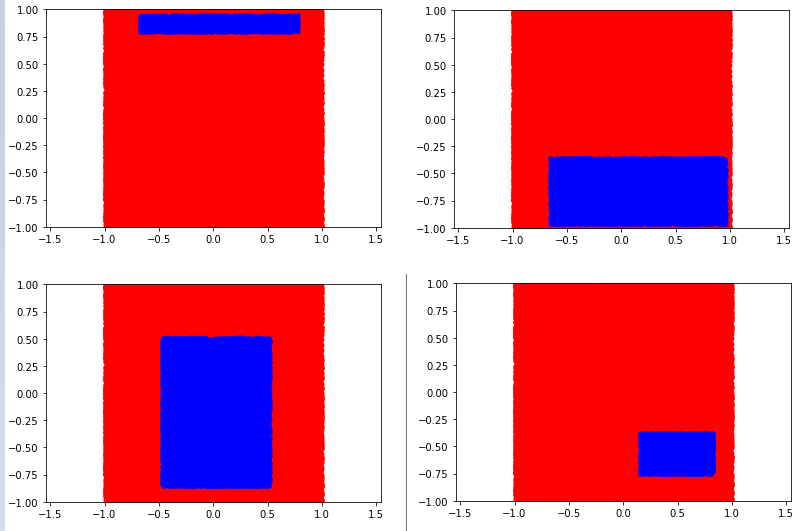


图 8 更多的区间查询结果

### 3.3 运行时间对比

我们随机生成查询区域，并循环运行多次，统计总查询时间对比两种方法的效率。

对比实验方法不需要对数据进行预处理，只对点集中所有点进行遍历，判断每一个点是否在查询区间范围内。这种方法运行时间相对稳定，不需要预处理，但速度较慢。

使用构建区间树并进行区间查询的方法，需要提前对点集进行排序并构建区间树，需要更长的预处理时间与构建区间树消耗的存储空间。但是这种方法能够大幅减少遍历点的个数从而在查询区域较少的时候获得更快的速度。当查询区域接近整个二维空间范围时，查询时间与

在本课程实验中，我们随机生成查询区域并进行区间查询1次、10次、100次、1000次，并统计运行时间。结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 查询次数 | 1 | 10 | 100 | 1000 | 10000 |
| 循环总查询时间(s) | 0.0539 | 0.5084 | 5.1994 | 53.4058 | 558.8850 |
| 区间树总查询时间(s) | 5.3689 | 5.5753 | 6.7451 | 20.2457 | 157.2543 |

从上面实验结果我们可以看到，当查询次数比较少的时候，构建区间树结构占据程序运行时间的主要部分。而随着查询次数的增加，构建区间树的时间耗费被平摊到更多的查询上后，区间树查询的优势就体现出来了。实验结果与我们之前的分析吻合。

## 4. 小结

在平面或高维空间中进行区间查询时，构建空间中的区间树是非常有效的方法。在区间查询次数比较多时，性能上会远远优于暴力破解的方法。本课程论文中实现了二维空间上的区间树构建以及在区间树上进行区域查询的功能，在查询次数很大时取得了较好的查询效率。该方法查询的时间复杂度为O(log2 n+k)，可以通过fractional cascading减少一个log n 因子。