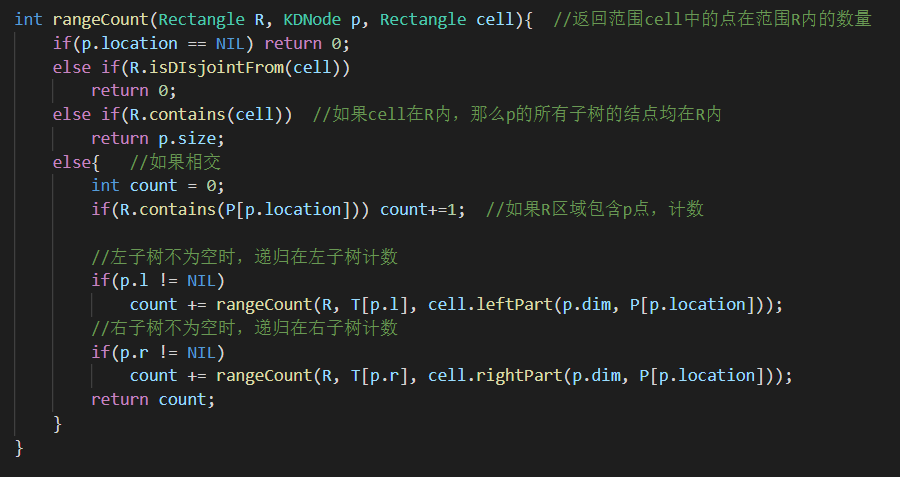
实验报告

一、设计思路

1. 范围查询：

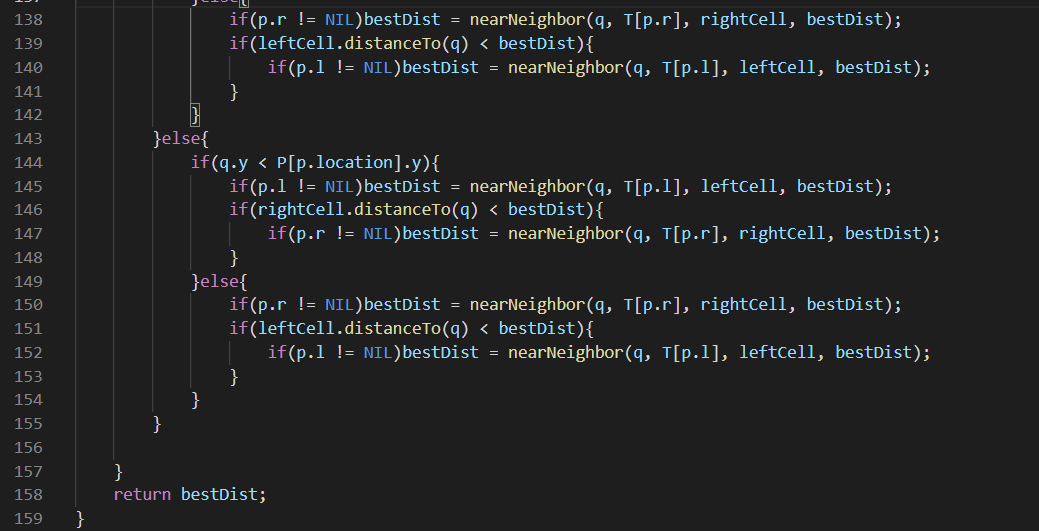
为了进行范围查询（查询某个范围内有多少点），设计了一个Rectangle类，用于描述每个KDTree结点代表的范围。然后从根节点及其对应的范围开始遍历，如果当前节点代表的范围cell在目标范围R内，则当前节点的所有子树上的节点都在R中，就无需再遍历子树，直接加上结点数即可；如果cell与R相交，则需要递归在左右子树中继续遍历；如果cell与R没有交集，则当前节点的所有子树上的节点都不在R中，也无需遍历了。这样就起到了剪枝的效果，降低了时间复杂度。



2. 最近邻查询：

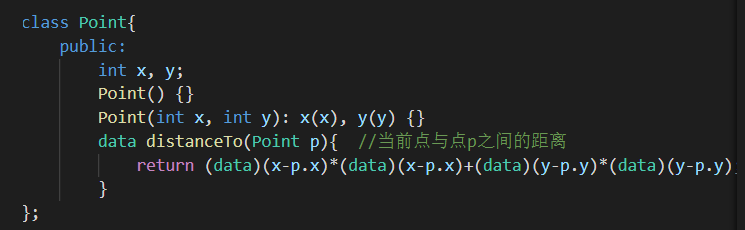
最近邻查询要查找与目标点q距离最近的点。虽然已经为每个KDNode结点分配了对应的范围cell，但是最近邻不一定在cell内，所以需要回溯查找。首先仍然从根节点出发，计算当前节点与q的距离，并据此更新最近距离，然后将当前节点代表的范围分为左右两部分，并递归在左右子树中查找。这样，就会先查找到q所在的子树，从而更有可能先找到更接近q的点，从而起到剪枝的作用。此外，在递归查找左右子树时，优先查找离q更近的子树范围，然后据此判断另一棵子树时候还有必要遍历（如果在对应维度上的距离小于当前最好距离才遍历），也起到剪枝的效果。



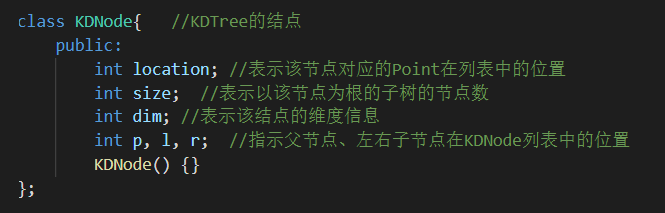


二、使用的数据结构

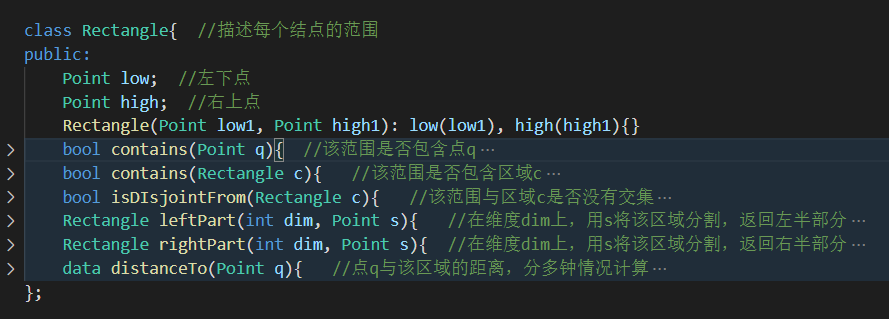
1. Point类 ：表示一个点，有x、y坐标。distanceTo函数计算当前点与p的距离



2. KDNode类：表示一个KDTree结点，具体意义如下：



3. Rectangle类：用左下点和右上点描述每个KDNode结点代表的范围。还有contains、isDisjointFrom等函数计算该区域与某个点或某个区域的相交关系。leftPart和rightPart将该范围分割成两部分。



为了节省空间，没有采用指针的方式表示，而是建立了Point和KDNode的两个数组用于存放所有的点和KDTree结点。结点对应的点在P中的位置和左右子节点在T中的位置都以下标的形式表示。



三、测试结果

按照说明修改部分代码，测试yoj 790和791题，均通过，证明程序正确。





四、代码

代码详见“KDTree.cpp”。其中递归建立KDTree的代码如下，每次都选择中位数作为新的根节点，再递归建立左右子树。

