**实验报告**

**一、BBF算法设计**

原有的KDTree最近邻查找算法中，是沿着KDTree搜索至叶子节点后依次回溯，回溯的路程就是之前查找时的逆序。而BBF算法则在此基础上，利用了这些点的信息，在回溯时为各个需要回溯的结点区分优先级，优先查找离查询点更近、更可能包含最近邻点的空间。

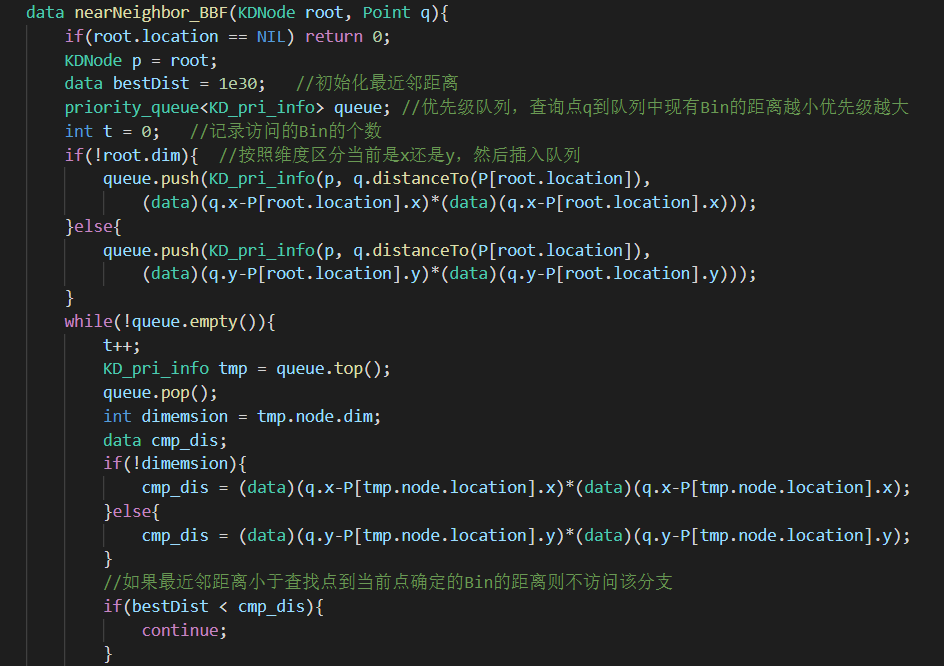
具体过程如下：通过对回溯可能需要的路过的结点加入优先队列，并按照查找点到该结点确定的Bin的距离进行排序，然后每次取优先级最高的节点（即距离最短的结点），计算该节点到查找点的距离是否比最近邻距离小，如果是则更新最近邻距离。如果查找点在切分维度上在该节点左部，则把他的右孩子加入到队列中，同时检索它的左孩子，否则就把他的左孩子加入到队列中，同时检索它的右孩子。这样一直重复检索，并加入队列，直到检索到叶子节点。上述过程又不断重复直到队列为空、算法结束。

此外，BBF算法设立了一个限制，如果访问的Bin的个数超过该限制，不管队列是不是为空，都会停止运行，返回当前的最近邻点作为结果。并且也用E\_max限制原有的KDTree最近邻查找算法中最多访问E\_max个叶节点，变为Restricted NN方法。

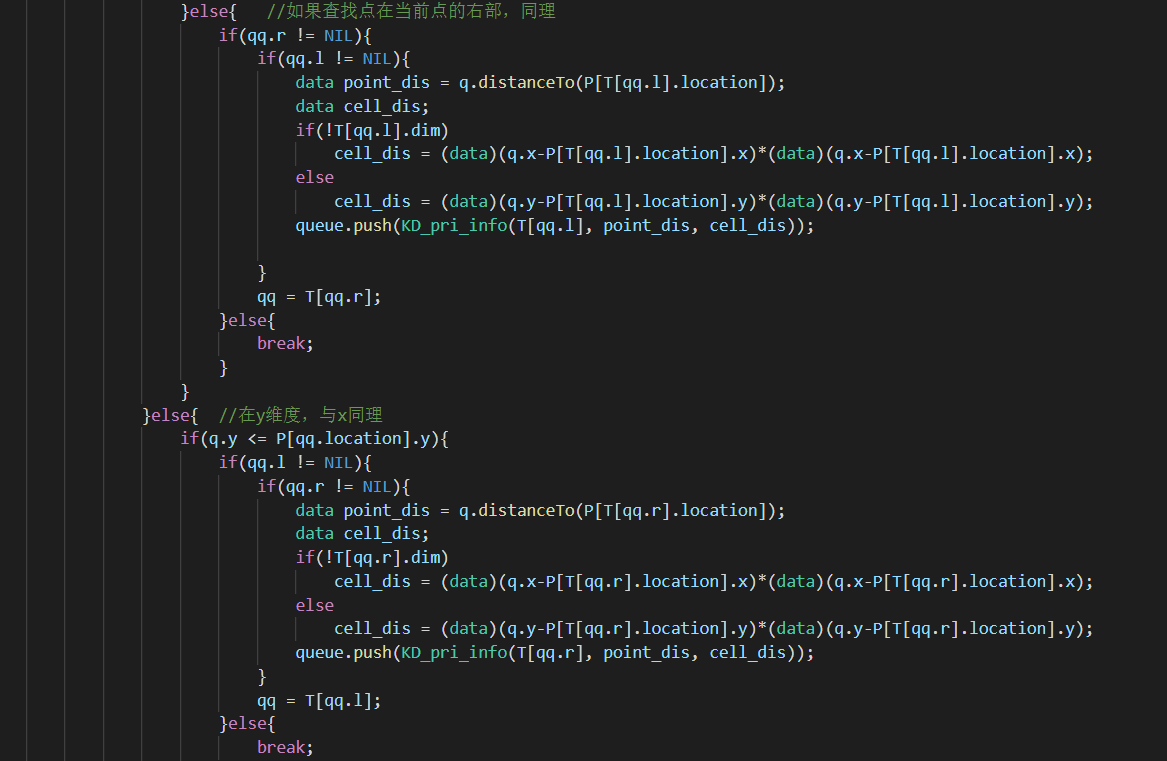
**二、代码实现及说明**

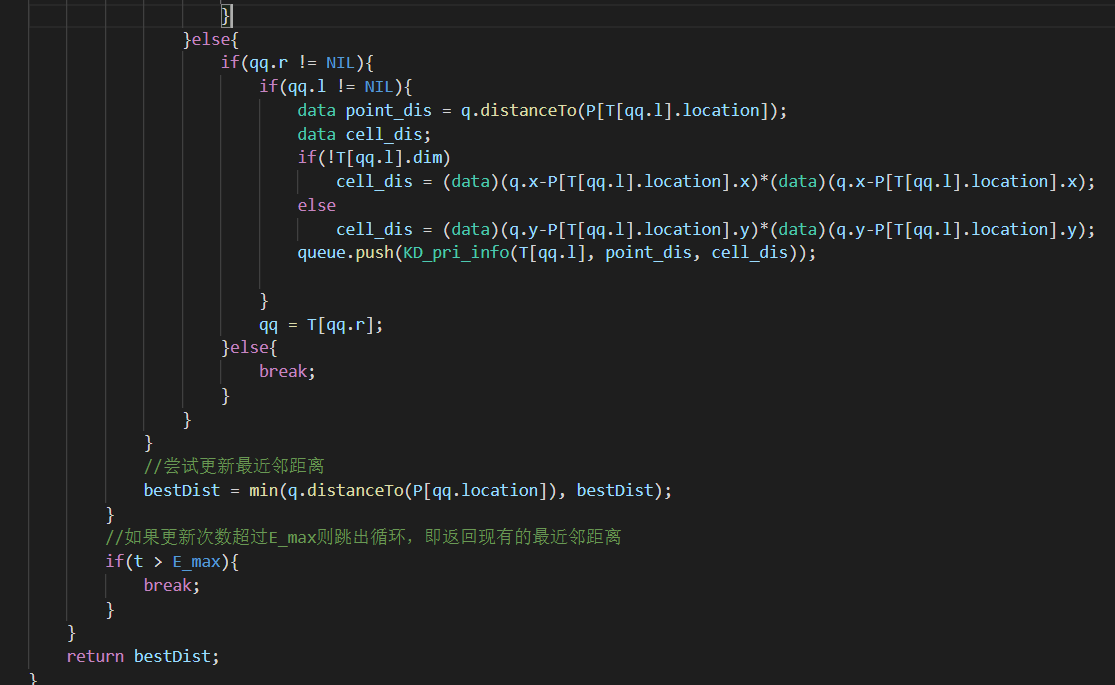
BBF算法的代码详见“KDTree\_BBF.cpp”文件，Restricted NN方法则详见“KD\_RestrictedNN.cpp”文件。由于其余部分都与实验5相同，所以仅展示实现BBF算法的函数以及更改部分。

实现BBF算法的函数，流程与算法设计中的描述相同：

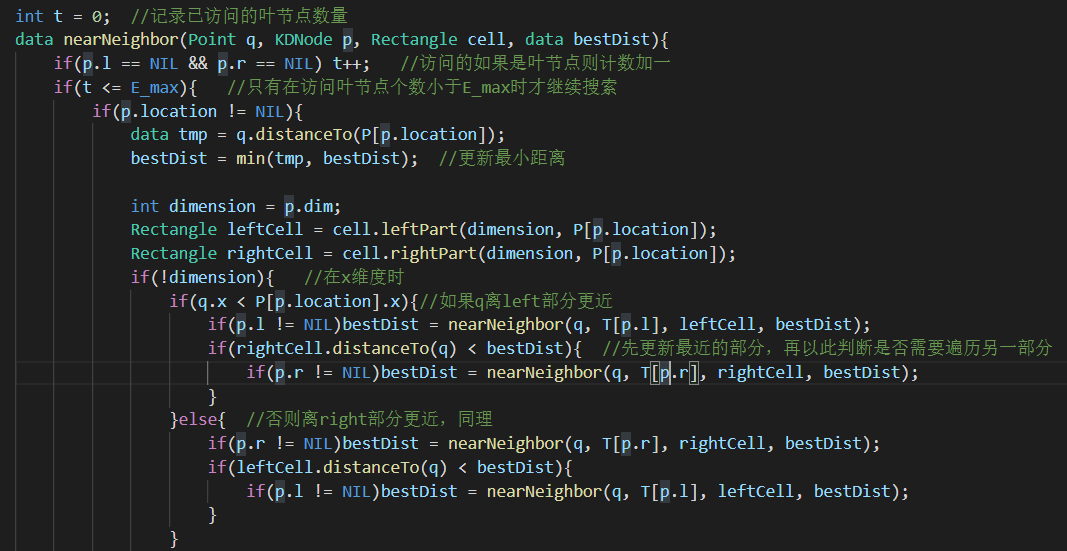


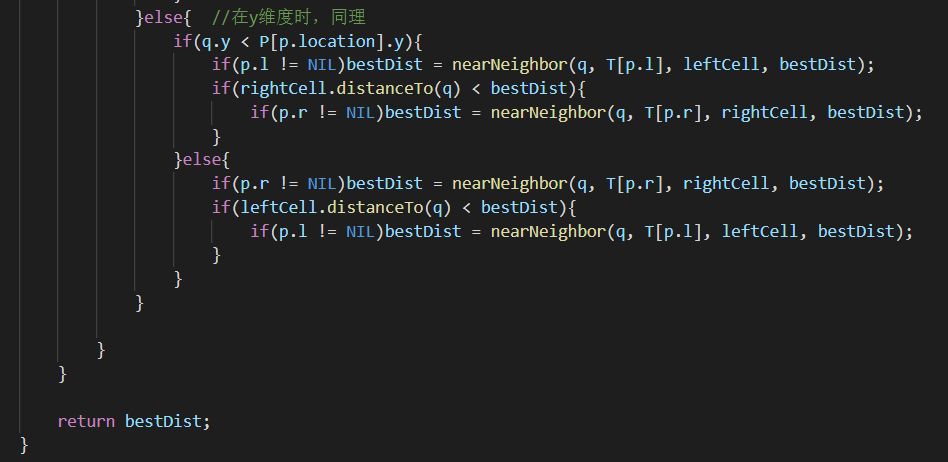






Restricted NN方法的更改如下，添加了t变量记录访问的叶节点个数并进行限制：

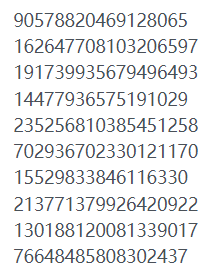
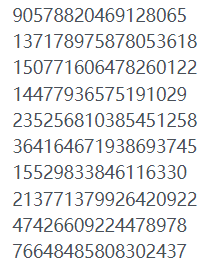




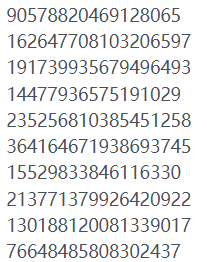
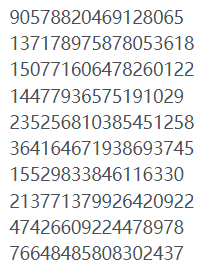
**三、BBF和Restricted NN方法结果对比**

使用所给数据测试E\_max取不同值时两种方法找到结果的差异，由于所给数据较多且较大，所以只选择数据量最小的“1.in”来展示，由于数据量很小，所以对应的E\_max限制也很小。令E\_max分别取3，20和50。

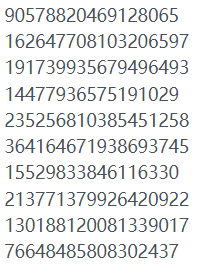
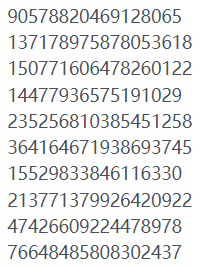
E\_max=3时，BBF（左）与Restricted NN（右）：

E\_max=20时，BBF（左）与Restricted NN（右）：

E\_max=50时，BBF（左）与Restricted NN（右）：

将以上输出与正确答案对比，发现RestrictedNN算法输出结果全部正确，BBF算法却存在部分错误。猜测是因为BBF算法本就是近似算法，不能完全保证正确性，并且其更适用于高维数据，因此在二维小数据量的测试样例上表现差于RestrictedNN。

自行实验了其他数据量较大的样例（“2.in”-“10.in”），并修改E\_max，也发现BBF算法的正确率还是比较高的，但相比之下仍然是RestrictedNN正确率更高。