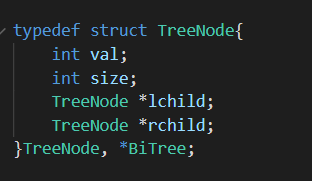
实验报告

一、数据结构设计

结点的数据结构设计如下，其中val表示该节点存储的值，size表示以该节点为根节点的子树的节点数量（包括该节点本身），lchild是指向左子树的指针，rchild是指向右子树的指针。



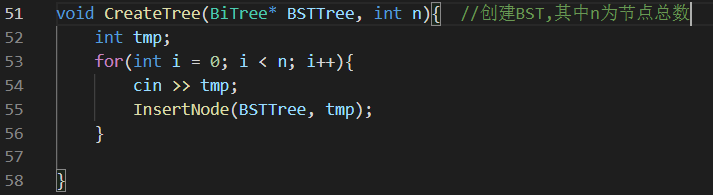
二、代码说明

本实验在普通二叉搜索树（BST）的基础上进行，所以具有插入、删除等基本功能。基本功能添加了注释，由于不是本实验的重点因此不再详细说明。需要注意之处只有插入和删除结点时会对查找路径上的结点的size进行相应处理。代码详见“BST.cpp”

1.插入结点

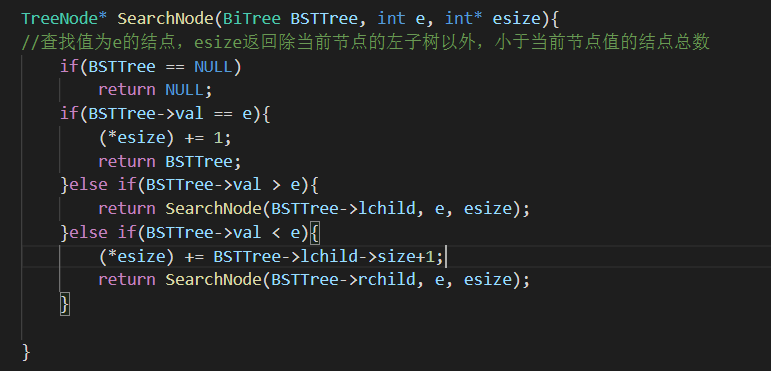


2.创建BST



3.查找结点

该函数中，为了便于在AuxBST\_Find中的使用，添加了一个参数esize，以求递归查找过程中，除所求节点的左子树以外，小于所求节点值的结点总数

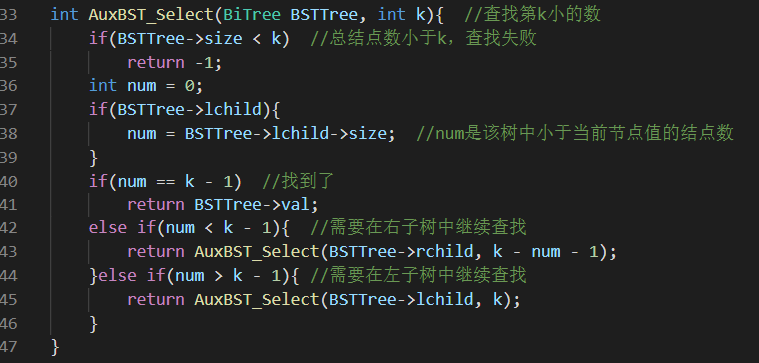


4. 删除节点



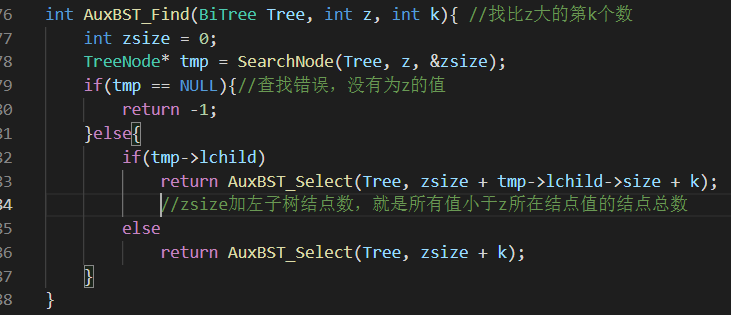
5.查找第k小的数

由于为结点添加了size属性，所以在查找第k小的数时，就不需要遍历二叉搜索树了，只需要根据当前节点及其左右子树的size值来判断第k小的数的位置，然后在对应的子树中继续查找即可。即如果左子树的结点数小于k，那么第k个数就在右子树中，继续查找右子树即可（注意在查找右子树时，k中需要去掉左子树的部分）；否则查找左子树。



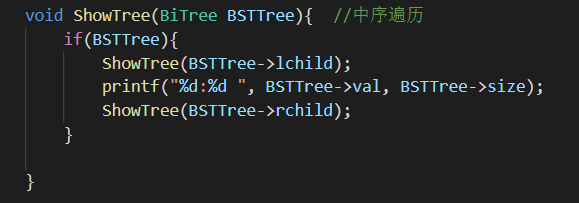
6. 找比z大的第k个数

先用查找功能找到z值对应的结点，然后求出所有小于该节点的结点的总数，即找到z值的排序rank(z)，然后调用Aux\_Select函数找到第rank(z)+k个数，就是比z大的第k个数。



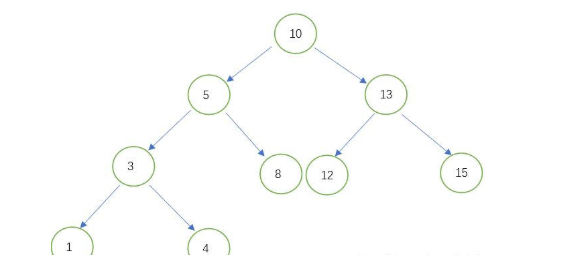
7.中序遍历BST

比较特别的是输出格式都以“**结点存储值：结点size值**”的形式输出。

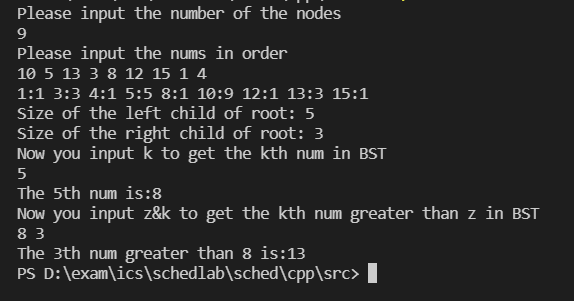


三、测试结果

按照 10 5 13 3 8 12 15 1 4的顺序输入九个结点，构造如下BST树：



按照所给的要求，对应的输出为：



经检验（还使用了其他数据检验，在此不再赘述），所给的结果都与实际符合，证明程序正确无误。

四、复杂度分析

1. k-select查找：

由于AuxBST\_Select不需要遍历二叉搜索树，在树的每一层只需要访问一棵子树，所以对应的复杂度应该与BST上的查找相同，均为O(logn)。而非强化的BST需要递归计数，直到找到第k个数为止，时间复杂度为O(n)。因此，强化后的BST在时间复杂度上有所提高。

2. 找比z大的第k个数：

该功能分别调用SearchNode和AuxBST\_Select，而这两个函数的复杂度都是O(logn)，所以AuxBST\_Find的时间复杂度也是O(logn)。而非强化的BST仍然要遍历找到z之后继续向后找k个，复杂度还是O(n)。所以强化后的BST在时间复杂度上也有所提高。