**一、实验题目**

二叉查找树

**二、实验目的：**

mplementing a binary tree and its common operations with VC++ or Python，and then use this tree to make decision or classification.

利用VC++/Python语言实现对二叉树的基本操作设计，用二叉树进行简单的判定或者分类，要求二叉树结点可以插入和删除。

**三、实验设备与环境**

微型计算机、macOS 系列操作系统 、Pycharm系列软件

**四、实验内容**

建立一棵二叉查找树，实现插入、删除和查找功能。

**五、概要设计（思路、算法、步骤等）**

5.1 二叉查找树具有以下特点：若左子树不空，则左子树上所有结点的值均小于或等于它的根结点的值；若右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值；左、右子树也分别为二叉排序树。

5.2 所有树结构都是由一个一个的节点构成的，所以先实现一个节点类。

5.3 创建二叉查找树

从输入处读入一个列表，若列表长度大于0，则将列表中的第一个元素赋值给root（根）节点，并遍历列表，把接下来的元素依次插入；若列表长度为0，返回NULL。

在二叉树中添加节点时，要先创建节点，有了节点类，实例化一个节点类的实例即可，节点初始化时是一个孤立的节点，指向的父节点、左子节点、右子节点都为空。将节点添加到树中后，才属于树的一部分。

5.4 二叉查找树插入元素

在二叉查找树插入元素时，要分情况讨论：当访问到最后得到的是None，说明元素不存在查找树上，当前结点值等于key时查找成功，返回True，当结点值小于key时，说明目标有可能存在右子树，递归右子树；当结点值大于key时，说明目标有可能存在左子树，递归左子树。

5.5 删除节点

删除根节点的处理方法，为了保证删除根节点后依旧是一颗完整的二叉查找树，这里可以用左子树中的最大值和右子树中的最小值来代替根节点，然后在子树中删除相应的叶节点

1）若root值为None, 说明二叉树中不存在要删除的值，直接返回

2）如果root值大于key，key可能在左子树，递归

3）如果root值小于key，key可能在右子树，递归

4）若root值刚好是key，说明已经找到了要删除的结点，进行删除处理操作：

a) 如果root没有左右子树了，直接删除结点，并且返回

b) 如果root还有左子树，则寻找左子树中的最大值，用于替换root，然后在左子树中删除结点

c) 如果root还有右子树，则寻找右子树的最小树，用于替换root，然后在右子树中删除结点

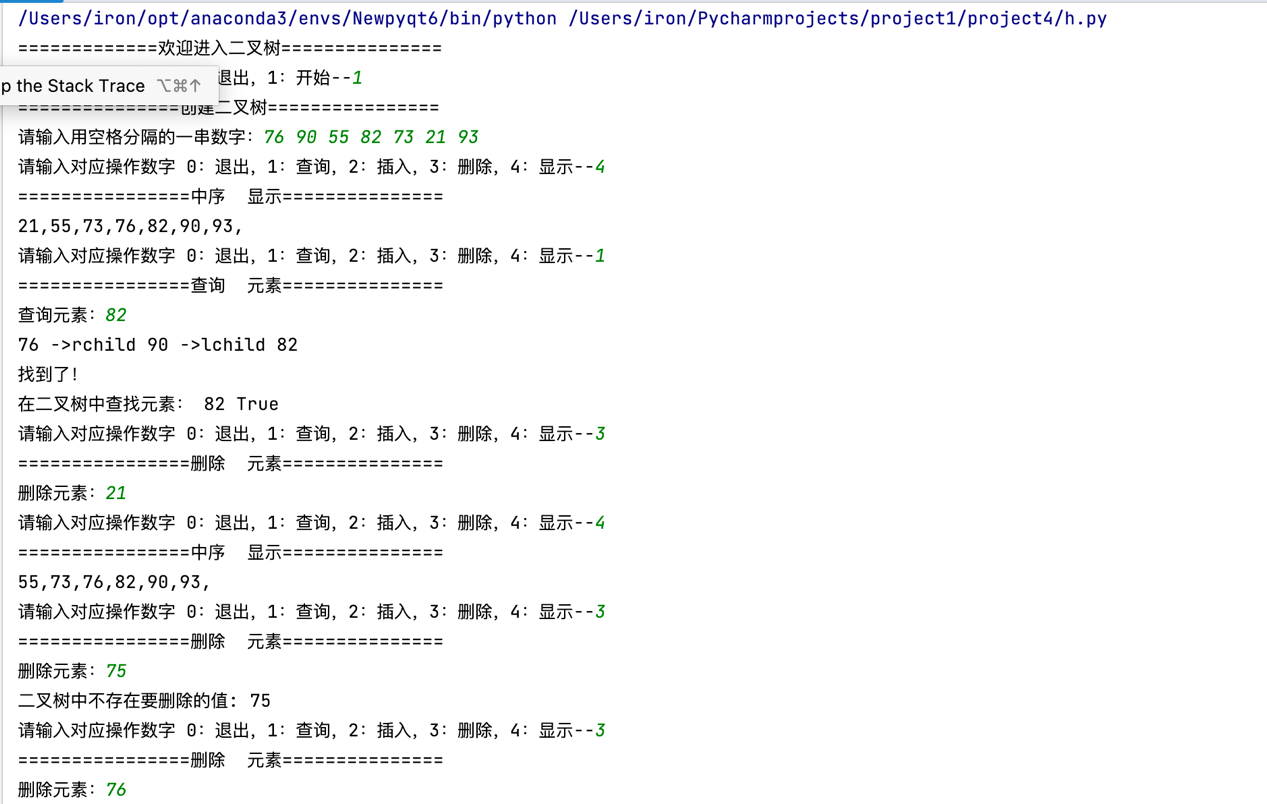
5.6 中序遍历二叉树

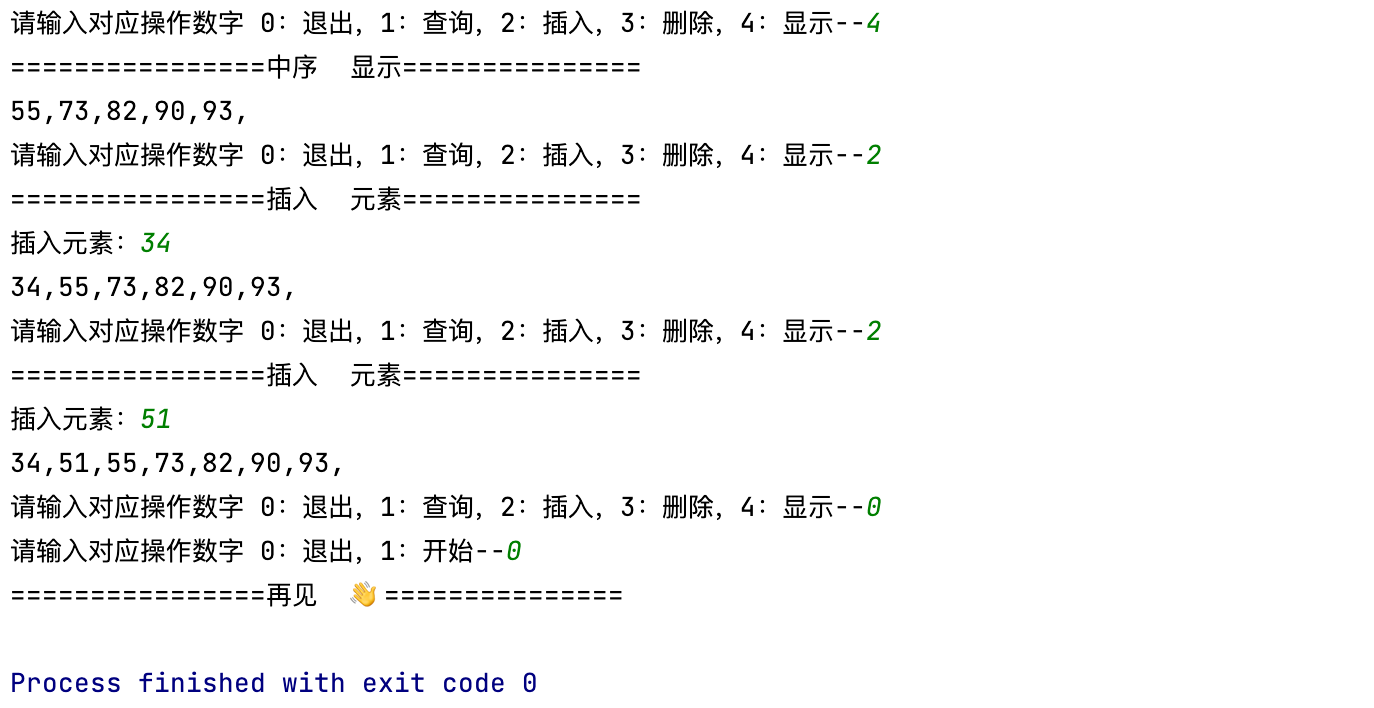
先遍历左子树，再遍历根节点，再遍历右子树。

**六、详细设计（核心代码、算法流程图等）**

*# -\*- coding: utf-8 -\*-  
# @Time : 2023/4/7  
# @Author : @yaowanxin  
# @Software: PyCharm  
# @description: hhhhhh*class tree\_node:  
 def \_\_init\_\_(self, data, lchild=None, rchild=None):  
 self.data = data  
 self.lchild = lchild  
 self.rchild = rchild  
  
class bst\_tree:  
  
 *#查找函数* def bst\_search(self, root, key):  
 str1 = ""  
 *# 当访问到最后得到的是None，说明元素不存在查找树上* if root == None:  
 print(None)  
 print("找不到！")  
 return False  
 *# 当前结点值等于key时查找成功* if root.data == key:  
 print(root.data)  
 print("找到了！")  
 return True  
 *# 当结点值小于key时，说明目标有可能存在右子树* elif root.data < key:  
 str1 = str1 + str(root.data) + " ->rchild "  
 print(str1, end="")  
 return self.bst\_search(root.rchild, key)  
  
 *# 当结点值大于key时，说明目标有可能存在左子树* if root.data > key:  
 str1 = str1 + str(root.data) + " ->lchild "  
 print(str1, end="")  
 return self.bst\_search(root.lchild, key)  
  
 *# 二叉查找数元素插入  
 # 根据二叉查找树的左小右大的特性，当 当前结点值大于key则说明值插入到左子树，否则在右子树  
 # 当root == None时，说明就是插入的位置* def bst\_insert(self, root, key):  
 *# 当值为None，创建新结点* if root == None:  
 root = tree\_node(key)  
 *# 存在时* elif root.data == key:  
 print("此元素已存在，不必再次插入！")  
 pass  
 *# 当结点值小于key时，说明要往右子树插* elif root.data < key:  
 root.rchild = self.bst\_insert(root.rchild, key)  
 *# 当结点值大于key时，说明要往左子树插* elif root.data > key:  
 root.lchild = self.bst\_insert(root.lchild, key)  
 return root  
  
 *## 寻找二叉查找树以root为根节点的最大权值* def bst\_search\_max(self, root):  
 if root.rchild:  
 return self.bst\_search\_max(root.rchild)  
 else:  
 return root  
  
 *# 删除  
 # 删除根节点的处理方法，为了保证删除根节点后依旧是一颗完整的二叉查找树，这里可以用左子树中的最大值和右子树中的最小值来代替根节点，然后在子树中删除相应的叶节点  
 # 1）若root值为None, 说明二叉树中不存在要删除的值，直接返回  
 # 2）如果root值大于key，key可能在左子树，递归  
 # 3）如果root值小于key，key可能在右子树，递归  
 # 4）若root值刚好是key，说明已经找到了要删除的结点，进行删除处理操作：  
 # a) 如果root没有左右子树了，直接删除结点，并且返回  
 # b) 如果root还有左子树，则寻找左子树中的最大值，用于替换root，然后在左子树中删除结点  
 # c) 如果root还有右子树，则寻找右子树的最小树，用于替换root，然后在右子树中删除结点* def bst\_delete(self, root, key):  
 *# 若root值为None, 说明二叉树中不存在要删除的值，直接返回* if root is None:  
 print("二叉树中不存在要删除的值: " + str(key))  
 return root  
 *# 如果key比root的值小，那么要在root的左子树中删除* if key < root.data:  
 root.lchild = self.bst\_delete(root.lchild, key)  
  
 *# 如果key比root的值大，那么要在root的右子树中删除* elif key > root.data:  
 root.rchild = self.bst\_delete(root.rchild, key)  
  
 *# 如果key等于root的值，那么要删除root节点* else:  
 *# 如果root没有左右子树，则直接删除root* if root.lchild is None and root.rchild is None:  
 root = None  
  
 *# 如果root只有左子树或右子树，则用子树节点代替root节点* elif root.lchild is None:  
 root = root.rchild  
 elif root.rchild is None:  
 root = root.lchild  
  
 *# 如果root既有左子树又有右子树，则用左子树的最大节点或右子树的最小节点代替root节点* else:  
 *# 找到左子树的最大节点* max\_node = self.bst\_search\_max(root.lchild)  
 *# 将左子树的最大节点的值赋给root节点* root.data = max\_node.data  
 *# 在左子树中删除最大节点* root.lchild = self.bst\_delete(root.lchild, max\_node.data)  
 return root  
  
 *# 中序遍历二叉树* def bst\_mid\_scan(self, root):  
 if root is None:  
 return  
 *# 遍历左子树* self.bst\_mid\_scan(root.lchild)  
 *# 遍历根节点* print(root.data, end=',')  
 *# 遍历右子树* self.bst\_mid\_scan(root.rchild)  
  
 *#创建二叉查找树* def bst\_create(self, list1):  
 if len(list1) != 0:  
 root = tree\_node(list1[0])  
 else:  
 return None  
 for i in range(1, len(list1)):  
 self.bst\_insert(root, list1[i])  
 return root  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 BST = bst\_tree()  
 print("=============欢迎进入二叉树===============")  
 flag1 = int(input("请输入对应操作数字 0：退出，1：开始--"))  
 while(flag1 != 0):  
  
 print("===============创建二叉树================")  
 list1 = list(map(int, input("请输入用空格分隔的一串数字：").split()))  
 root = BST.bst\_create(list1)  
 flag2 = int(input("请输入对应操作数字 0：退出，1：查询，2：插入，3：删除，4：显示--"))  
 while flag2 != 0:  
 if flag2 == 1:  
 print("================查询 元素===============")  
 n = int(input("查询元素："))  
 print("在二叉树中查找元素：", n, BST.bst\_search(root, n))  
 if flag2 == 2:  
 print("================插入 元素===============")  
 n = int(input("插入元素："))  
 BST.bst\_insert(root, n)  
 BST.bst\_mid\_scan(root)  
 print()  
 if flag2 == 3:  
 print("================删除 元素===============")  
 n = int(input("删除元素："))  
 BST.bst\_delete(root, n)  
 if flag2 == 4:  
 print("================中序 显示===============")  
 BST.bst\_mid\_scan(root)  
 print()  
 flag2 = int(input("请输入对应操作数字 0：退出，1：查询，2：插入，3：删除，4：显示--"))  
 flag1 = int(input("请输入对应操作数字 0：退出，1：开始--"))  
 print("================再见 👋===============")

**七、测试结果及分析：**





**八、总结**

本次实验利用python语言完成了二叉查找树，学到了怎么建立一棵二叉查找树，以及如何通过先序遍历的方式去查找某一个节点。先序遍历相对于中序遍历和后序遍历来说，比较简单，所以在程序里使用了先序遍历去查找，但是中序遍历和后序遍历也要掌握。本次实验的难点我觉得是删除节点，如果被删除节点没有子节点或者只有一个子节点的情况比较简单，如果有两个子节点就需要去迭代找到合适的节点。由于一开始对这里概念不是特别清楚，不知道该怎么去实现，后来通过查阅资料，掌握了这个知识点。