**使用python实现创建二插树和实现树的各种遍历**

**算法实现思路：**

**先构建树的节点结构，在借用节点结构创建树结构。。。其他的各种遍历按以前c语言的实现算法就行。深度和广度分别使用栈和队列的工具**

**前序遍历：按根左右实现**

**中序遍历：按左根右实现**

**后序遍历：按左右根实现**

**class** TreeNode:  
 *'''二叉搜索树节点的定义'''* **def** \_\_init\_\_(self, val):  
 self.val = val  
 self.left = **None** self.right = **None  
  
class** OperationTree:  
 *'''二叉树操作'''* **def** create(self, List):  
 *'''二叉搜索树插入操作'''* root = TreeNode(List[0])  
 lens = len(List)  
 **if** lens >= 2:  
 root.left = self.create(List[1])  
 **if** lens >= 3:  
 root.right = self.create(List[2])  
 **return** root  
  
 **def** query(self, root, data):  
 *'''二叉树查找操作'''* **if** root == **None**:  
 **return False  
 if** root.val == data:  
 **return True  
 elif** root.left:  
 **return** self.query(root.left, data)  
 **elif** root.right:  
 **return** self.query(root.right, data)  
  
 **def** PreOrder(self, root):  
 *'''打印二叉树(先序)'''* **if** root == **None**:  
 **return** print(root.val, end=**' '**)  
 self.PreOrder(root.left)  
 self.PreOrder(root.right)  
  
 **def** InOrder(self, root):  
 *'''中序打印'''* **if** root == **None**:  
 **return** self.InOrder(root.left)  
 print(root.val, end=**' '**)  
 self.InOrder(root.right)  
  
 **def** BacOrder(self, root):  
 *'''后序打印'''* **if** root == **None**:  
 **return** self.BacOrder(root.left)  
 self.BacOrder(root.right)  
 print(root.val, end=**' '**)  
  
 **def** BFS(self, root):  
 *'''广度优先'''* **if** root == **None**:  
 **return** *# queue队列，保存节点* queue = []  
 *# res保存节点值，作为结果  
 #vals = []* queue.append(root)  
  
 **while** queue:  
 *# 拿出队首节点* currentNode = queue.pop(0)  
 *#vals.append(currentNode.val)* print(currentNode.val, end=**' '**)  
 **if** currentNode.left:  
 queue.append(currentNode.left)  
 **if** currentNode.right:  
 queue.append(currentNode.right)  
 *#return vals* **def** DFS(self, root):  
 *'''深度优先'''* **if** root == **None**:  
 **return** *# 栈用来保存未访问节点* stack = []  
 *# vals保存节点值，作为结果  
 #vals = []* stack.append(root)  
  
 **while** stack:  
 *# 拿出栈顶节点* currentNode = stack.pop()  
 *#vals.append(currentNode.val)* print(currentNode.val, end=**' '**)  
 **if** currentNode.right:  
 stack.append(currentNode.right)  
 **if** currentNode.left:  
 stack.append(currentNode.left)  
 *#return vals***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 List1 = [1,[2,[4,[8],[9]],[5]],[3,[6],[7]]]  
 op = OperationTree()  
 tree1 = op.create(List1)  
 print(**'先序打印：'**,end = **''**)  
 op.PreOrder(tree1)  
 print(**""**)  
 print(**'中序打印：'**,end = **''**)  
 op.InOrder(tree1)  
 print(**""**)  
 print(**'后序打印：'**,end = **''**)  
 op.BacOrder(tree1)  
 print(**""**)  
 print(**'BFS打印 ：'**,end = **''**)  
 bfs = op.BFS(tree1)  
 *#print(bfs)* print(**""**)  
 print(**'DFS打印 ：'**,end = **''**)  
 dfs = op.DFS(tree1)  
 *#print(dfs)* print(**""**)

