深度优先搜索

class Node():  
 def \_\_init\_\_(self, data=-1):  
 self.data = data  
 self.left = None  
 self.right = None  
class Tree():  
 def \_\_init\_\_(self):  
 self.root = Node()  
 def add(self, data):  
 # 为树加入节点  
 node = Node(data)  
 if self.root.data == -1: # 如果树为空，就对根节点赋值  
 self.root = node  
 else:  
 myQueue = []  
 treeNode = self.root  
 myQueue.append(treeNode)  
 while myQueue: # 对已有的节点进行层次遍历  
 treeNode = myQueue.pop(0)  
 if not treeNode.left:  
 treeNode.left = node  
 return  
 elif not treeNode.right:  
 treeNode.right = node  
 return  
 else:  
 myQueue.append(treeNode.left)  
 myQueue.append(treeNode.right)  
  
 def DG\_First(self, root):  
 if not root:  
 return  
 print(root.data,end="," )  
 self.DG\_First(root.left)  
 self.DG\_First(root.right)  
  
 def ZD\_First(self, root):  
 if not root:  
 return  
 myStack = []  
 node = root  
 while myStack or node:  
 while node: # 从根节点开始，一直寻找他的左子树  
 print(node.data,end="," )  
 myStack.append(node)  
 node = node.left  
 node = myStack.pop() # while结束表示当前节点node为空，即前一个节点没有左子树了  
 node = node.right # 查看它的右子树  
  
 def DG\_In(self, root):  
 if not root:  
 return  
 self.DG\_In(root.left)  
 print(root.data, end=",")  
 self.DG\_In(root.right)  
  
 def ZD\_In(self, root):  
 if not root:  
 return  
 myStack = []  
 node = root  
 while myStack or node: # 从根节点开始，一直寻找它的左子树  
 while node:  
 myStack.append(node)  
 node = node.left  
 node = myStack.pop()  
 print(node.data,end="," )  
 node = node.right  
  
 def DG\_Late(self, root):  
 if not root:  
 return  
 self.DG\_Late(root.left)  
 self.DG\_Late(root.right)  
 print(root.data, end=",")  
  
 def ZD\_Late(self, root):  
 # 先遍历根节点，再遍历右子树，最后是左子树  
 if not root:  
 return  
 myStack1 = []  
 myStack2 = []  
 node = root  
 while myStack1 or node:  
 while node:  
 myStack2.append(node)  
 myStack1.append(node)  
 node = node.right  
 node = myStack1.pop()  
 node = node.left  
 while myStack2:  
 print(myStack2.pop().data, end=",")  
  
 def level\_order\_queue(self, root): # 队列实现层次遍历（非递归）  
 if not root:  
 return  
 myQueue = []  
 node = root  
 myQueue.append(node)  
 while myQueue:  
 node = myQueue.pop(0)  
 print(node.data,end="," )  
 if node.left:  
 myQueue.append(node.left)  
 if node.right:  
 myQueue.append(node.right)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 主函数  
 datas = [3,5,8,9,7,4,1,2]  
 tree = Tree() # 新建一个树对象  
 for data in datas:  
 tree.add(data) # 逐个加入树的节点  
  
 print('递归前序遍历：')  
 tree.DG\_First(tree.root)  
  
 print('\n堆栈前序遍历')  
 tree.ZD\_First(tree.root)  
  
 print("\n\n递归中序遍历：")  
 tree.DG\_In(tree.root)  
  
 print("\n堆栈中序遍历：")  
 tree.ZD\_In(tree.root)  
  
 print('\n\n递归后序遍历：')  
 tree.DG\_Late(tree.root)  
  
 print('\n堆栈后序遍历：')  
 tree.ZD\_Late(tree.root)

