1.给出一棵二叉树,返回其中序遍历：

递归和非递归实现：

中序遍历应该遵循左根右的顺序

1. 递归

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

"""

@param root: The root of binary tree.

@return: Inorder in ArrayList which contains node values.

"""

def inorderTraversal(self, root):

# write your code here

a=zhongxu(root,a=[])

return a

def zhongxu(root,a):

if root!=None:

zhongxu(root.left,a)

a.append(root.val)

zhongxu(root.right,a)

return a

1. 非递归：
2. 首先要沿着左支树一直走才能找到最左边的点，如果有就一直走，没有就要放入数组中，然后判断右枝点，有就要继续走没有就要把该节点的父节点拿出来并把值放到数组里，这样一来思路就是顺着左枝走，走到左支点为空是记录该节点值，并往右支点走，如果右支点为空就要把该节点的父节点拿出来，else的话就把右节点当作新节点。这样使用数据结构就是栈。
3. **def** midOrder(self):
4. **if** **not** self.root:
5. **return**
6. stackNode = []
7. node = None
8. node = self.root
9. **while** stackNode **or** node:
10. **while** node:
11. stackNode.append(node)
12. node = node.leftNode
13. node = stackNode.pop()
14. **print** node.value,
15. node = node.rightNode

2.给出一棵二叉树，返回其节点值的层次遍历（逐层从左往右访问）

思路：要想能层次遍历必须同时找到左右支点，这样递归时就必须同时出现左支点和右支点

当然有很多技巧只要把两个递归放到同一表达式中即可，下面就是怎么确定是要找的层，可以做个index来表明当前层然后把index相同的放入列表中即可，

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

"""

@param root: The root of binary tree.

@return: Level order in a list of lists of integers

"""

def levelOrder(self, root):

# write your code here

q=[]

b=[]

for i in range(100):

if not temp2(root,i,temp=q,a=[]):

break

b.append(q)

q=[]

return b

def temp2(root,level,temp,a):

if not root or level <0:

return 0

if level==0:

temp.append(root.val)

return 1

return temp2(root.left,level-1,temp,a) + temp2(root.right,level-1,temp,a)

3. 给定一个二叉树，找出所有路径中各节点相加总和等于给定 目标值 的路径。

一个有效的路径，指的是从根节点到叶节点的路径。

**样例**

给定一个二叉树，和 目标值 = 5:

1

/ \

2 4

/ \

2 3

返回：

[

[1, 2, 2],

[1, 4]

]

思路类似于深度搜索，结束条件就是到叶节点而且和等于给定值。递归实现(不满足条件时出栈入栈)当然这一题考的是叶节点可以一直搜索到叶节点（虽然可能比较麻烦）

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

# @param {TreeNode} root the root of binary tree

# @param {int} target an integer

# @return {int[][]} all valid paths

def binaryTreePathSum(self, root, target):

# Write your code here

suml=0

sum2=0

path=[]

h=[]

n=sum(root,target,suml,path,h)

return n

def sum(root,target,suml,path,h):

if root!=None:

path=path+[root.val]

suml=suml+root.val

if suml==target and root.left==None and root.right==None:

h.append(path)

if root.left!=None:

sum(root.left,target,suml,path,h)

if root.right!=None:

sum(root.right,target,suml,path,h)

else:

return h

return h

4. 给一棵二叉树，找出从根节点到叶子节点的所有路径

这种直接找到叶节点并保存路径即可

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

# @param {TreeNode} root the root of the binary tree

# @return {List[str]} all root-to-leaf paths

def binaryTreePaths(self, root):

# Write your code here

#a=test(root,a=[])

#return a

a=test(root,'',a=[])

return a

def test(root,s,a):

if root !=None:

s+='->'+str(root.val)

if root.left==None and root.right==None:

s=s[2:]

a.append(s)

test(root.left,s,a)

test(root.right,s,a)

return a

5. 给出一棵二叉树，返回其节点值的后序遍历。

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

"""

@param root: The root of binary tree.

@return: Postorder in ArrayList which contains node values.

"""

def postorderTraversal(self, root):

# write your code here

a=houxu(root,a=[])

return a

def houxu(root,a):

if root !=None:

houxu(root.left,a)

houxu(root.right,a)

a.append(root.val)

return a

6. 给出一棵二叉树，返回其节点值的前序遍历

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

"""

@param root: The root of binary tree.

@return: Preorder in ArrayList which contains node values.

"""

def preorderTraversal(self, root):

# write your code here

a=qianxu(root,a=[])

return a

def qianxu(root,a):

if root==None:

return

if root !=None:

a.append(root.val)

qianxu(root.left,a)

qianxu(root.right,a)

return a

7. 假设你正在爬楼梯，需要n步你才能到达顶部。但每次你只能爬一步或者两步，你能有多少种不同的方法爬到楼顶部？

思路对于这种计算次数的大多使用动态规划，假设每次只能走两步或一步，那么第三个台阶的方法应该是第二台阶方法与第一个台阶方法之和。但整体限制了n个阶梯，因此可以数组a[i]代表i台阶的方法，a[i]=a[i-1]+a[i-2](递推关系可以从后项往前相推)

class Solution:

"""

@param n: An integer

@return: An integer

"""

def climbStairs(self, n):

# write your code here

a={0:1,1:1}

for i in range(2,n+1):

a[i]=a[i-1]+a[i-2]

return a[n]

8.

深度复制一个二叉树。

给定一个二叉树，返回一个他的 **克隆品** 。

思路二叉树的创建一般是通过递归来实现，不过这个要按照原有树的结构的创建。

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

this.val = val

this.left, this.right = None, None

"""

class Solution:

"""

@param {TreeNode} root: The root of binary tree

@return {TreeNode} root of new tree

"""

def cloneTree(self, root):

# Write your code here

if root!=None:

a=TreeNode(root.val)

else:

return

a.left=self.cloneTree(root.left)

a.right=self.cloneTree(root.right)

return a

9. 比较两个字符串A和B，确定A中是否包含B中所有的字符。字符串A和B中的字符都是 **大写字母。**在 A 中出现的 B 字符串里的字符不需要连续或者有序。

**思路假如直接使用in来判段无法区分aabb，与aaa的情况，因此最直接想法是把B中的字符个数统计下，如果出现在A中并且小于等于A中出现个数，即可。查询哈希表永远是最快的**

class Solution:

"""

@param A : A string includes Upper Case letters

@param B : A string includes Upper Case letters

@return : if string A contains all of the characters in B return True else return False

"""

def compareStrings(self, A, B):

# write your code here

dict={}

temp={}

for i in range(len(B)):

dict[B[i]]=dict.get(B[i],0)+1

for j in range(len(A)):

temp[A[j]]=temp.get(A[j],0)+1

for keys in dict:

if keys in temp:

if dict[keys]<=temp[keys]:

c=[]

else: return False

else: return False

return True

10. 给一个排序数组（从小到大），将其转换为一棵高度最小的排序二叉树。

思路高度最小的树就是平衡二叉树，这样必须从中间取值当作根节点这样才能保证左右支树高度尽可能相同，至与偶数时出现左面多还是应该右面多都是一样的，这样把根点做出来了，那么都保持这一准则就可以递归构造左右节点，出口是排序数组中不存在点，也就是左边的序号比右边的序号要大。

"""

Definition of TreeNode:

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left, self.right = None, None

"""

class Solution:

"""

@param A: a list of integer

@return: a tree node

"""

def sortedArrayToBST(self, A):

# write your code here

length=len(A)

if length==0:

return None

else:

return newTreeNode (A, 0,length-1)

def newTreeNode (A, start,end) :

if (start > end) :

return None

node = TreeNode(0);

mid = end - (end - start)/2;

node.val = A[mid]

node.left = newTreeNode(A, start, mid - 1);

node.right = newTreeNode(A, mid + 1, end);

return node