hello jack

binary tree, output node.val of it in post-order.

list [node.val,.....]

1

**1**

**\**

**2**

**/**

**3**

output: [3, 2, 1]

input: TreeNode, None

output: [] , []

Method1 : recursive method

1. dfs(root) -> return postorder traversal of the tree

corner case

dfs(root.left) dfs(root.right) append(root.val)

Time O(n) , Space O(n)

stack [recursion] - O(logN) height

res [....leaf] O(N)

1

2

3

4

5

Method2: iterative method

1. stack, stack to store all left nodes till the left most node
2. stack pop(), store it.val in res, node is the left most node
3. node -> node.right
4. do loop step 1 again untill that we can’t
5. go step 2 untill stack is None

class solution:

def dfs(self, root): #return postorder traversal of the tree

if not root:

return []

res = []

res += self.dfs(root.left)

res += self.dfs(root.right)

res.append(root.val)

return res

def iterative(self, root):

if not root:

return []

res, stack = [], [root]

while stack:

node = stack.pop()

res.append(node.val)

if node.left:

stack.append(node.left)

if node.right:

stack.append(node.right)

return res[::-1]

res post order [left, right, root] <- stack [root, right, left]. consider visited node and unvisited node。

Method2: iterative method

1. stack store the root
2. res = []
3. pop node and store the left and right
4. until the stack is None
5. res[::-1]

[1, 2, 3, 4]

1

2 3

4 [4, 2,3, 1]

res = [] stack = [1]

loop1:

node = 1 stack = []

res = [1] stack[2, 3]

loop 2:

node = 3 stack = [2]

res = [1, 3] stack = [2]

loop3:

node = 2 stack = []

res = [1, 3, 2] stack = [4]

loop4:

node = 4 stack = []

res = [1, 3, 2, 4] stack = []

res[::-1] = [4,2,3,1]

[1, 2, 3, 4, 5, 6, 7]

1

2 3

4 5 6 7 [4, 5, 2, 6, 7, 3, 1] [4,5,2,6,7,3,1]

res = [] stack[1]

loop1:

node = 1, res = [1], stack = [2, 3]

loop2:

node = 3, res = [1, 3], stack = [2, 6, 7]

loop3:

node = 7, res= [1,3,7] stack = [2,6]

loop4:

node = 6 res = [1,3,7,6] stack = [2]

loop5:

node = 2 res= [1,3,7,6,2] stack = [4,5]

loop6:

node = 5 res = [1,3,7,6,2,5] ,stack = [4]

loop7:

node = 4, res = [1,3,7,6,2,5.4] stack = []

res[::-1] = [4,5,2,6,7,3,1]

self.res = 0

dfs(root, 0)

def dfs(node, depth):

if not node.legt and not node.right:

self.res = max(self.res, depth)

return depth

dfs(node.left, depth + 1) # 左右不知道，传给他信息

dfs(node.right, depth + 1)

def depth(root): # 假设左右都知道了

if not root:

return 0

l, r = depth(node.left), depth(node.right)

return max(l, r) + 1