binary tree N nodes

nodes value

node.val represents the number of coins in that node

totally, there are n coins node.val: 0 - n

define one move: move one coin from one node to adjacent node(parent to child, child to parent)

now, I give you a tree, question: return the moves to make every node has exactly one coin

3

0 0 - > 2 step: root -》 left child, root -> right child 2

2

1 0

1

1 1

every node has same coin

max move ? min move !

input: root of the tree. treenode definition? root value repeated? Y order? N, number of node? 0-N; n coins node.val: 0 - n

output:int

corner case: None? -0, only node? -0

final, every node one coin? for some node, coin > 0, some is 0, in this case

root coin is biggest one? N; root maybe 0, child not 0?

sum up is N

example 1

4

0 0

0

move 1 to left, move 1 to right, and move 1 to left.left 4

example2

0

4 0

0

move1 to left, move 1 to parent, move 1 to parent.right 4 step

1

3 0

0

from 3 move 1 to left, move 1 to parent.right. 3 step

4 node

steps of move didn’t relate to coins location for eg1 eg2

4 coin where, step 4 step

example 4

0

0 4

0

move 1 to parent, move 1 to parent left.left, move 1 to parent.left

1 + 3 + 2 = 6

moves left balanced

right balanced

A recursion method

method:

move the coin in root and its left and right, count the step

if root.val > 0, move root.val - 1 to other, step add val -1

if root.val ==0, left/right move root, step add

if left no, right has, right -> root, right -》root-> left, step add 3

if right no,... do same things

do same things on left subtree

do same thing on right subtree

return step

time complexity: O(N) space: O(logN) height of tree logN

class mySolution:

def findSteps(self, root): # return int of steps

if not root: # corner case

return 0

if not root.left and not root.right:

return 0

step = 0

step = self.dfs(root, step)

return step

def dfs(self, root, step): # return steps

if not root:

return 0

if not root.left and not root.right:

return 0

if root.val > 0:

step += root.val - 1

if root.val == 0:

if root.left.val == 0 and root.right.val != 0:

#left = 0. right = 2 step += 2 1 coins from right to root

# right 3 coint, root is 0 ,left 0, 3 steps

# 4 right,

if

step += 3

if root.right.val == 0 and root.left.val ! = 0:

B. transfer tree to list by level order, then calculate the step

Method:

1. use level order traversal the tree, save node.val in a list [3, 0, 0]
2. find the steps to move coint
   1. scan elem from start

helper(root) :return the number of coins given to it.s parent

[3, 0,0]

root is left 0, helper() -> -1

root is right ->-1

example2

0

4 0

0 0

0 - -1

4, 0 4 2

0 kong -1 0

1 + 0 + 2 + 1 = 4

def solve(root):

self.res = 0

def helper(node): #number of coins that needs to return to his father

if not node:

return 0

left = helper(root.left)

right = helper(root.right)

#self.res += abs(left) + abs(right)

return left + right + root.val -1

helper(root)

return self.res

Hello