**Assignment 22 - Tree**

**Question-1:**

Given a Binary Tree (Bt), convert it to a Doubly Linked List(DLL). The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be the same as in Inorder for the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.

**Sol:**

# Python program for conversion of

# binary tree to doubly linked list.

class Node:

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

self.right = None

self.data = val

self.left = None

# Global variable used in convert

prev = None

def BinaryTree2DoubleLinkedList(root):

# Base case

if root is None:

return root

# Recursively convert left subtree

head = BinaryTree2DoubleLinkedList(root.left);

# Since we are going to change prev,

# we need to use global keyword

global prev

# If prev is empty, then this is the

# first node of DLL

if prev is None :

head = root

else:

root.left = prev

prev.right = root

# Update prev

prev = root;

# Recursively convert right subtree

BinaryTree2DoubleLinkedList(root.right);

return head

def print\_dll(head):

# Function to print nodes in given

# doubly linked list

while head is not None:

print(head.data, end=" ")

head = head.right

# Driver program to test above functions

# Let us create the tree as

# shown in above diagram

if \_\_name\_\_ == '\_\_main\_\_':

root = Node(10)

root.left = Node(12)

root.right = Node(15)

root.left.left = Node(25)

root.left.right = Node(30)

root.right.left = Node(36)

head = BinaryTree2DoubleLinkedList(root)

# Print the converted list

print\_dll(head)

**Question-2**

A Given a binary tree, the task is to flip the binary tree towards the right direction that is clockwise. See the below examples to see the transformation.

In the flip operation, the leftmost node becomes the root of the flipped tree and its parent becomes its right child and the right sibling becomes its left child and the same should be done for all left most nodes recursively.

**Sol:**

# Python3 program to flip  
# a binary tree  
from collections import deque  
  
  
# A binary tree node structure  
class Node:  
  
 def \_\_init\_\_(self, key):  
 self.data = key  
 self.left = None  
 self.right = None  
  
  
# method to flip the  
# binary tree  
def flipBinaryTree(root):  
 # Initialization of  
 # pointers  
 curr = root  
 next = None  
 temp = None  
 prev = None  
  
 # Iterate through all  
 # left nodes  
 while (curr):  
 next = curr.left  
  
 # Swapping nodes now, need temp  
 # to keep the previous right child  
  
 # Making prev's right as curr's  
 # left child  
 curr.left = temp  
  
 # Storing curr's right child  
 temp = curr.right  
  
 # Making prev as curr's right  
 # child  
 curr.right = prev  
  
 prev = curr  
 curr = next  
 return prev  
  
  
# Iterative method to do level  
# order traversal line by line  
def printLevelOrder(root):  
 # Base Case  
 if (root == None):  
 return  
  
 # Create an empty queue for  
 # level order traversal  
 q = deque()  
  
 # Enqueue Root and initialize  
 # height  
 q.append(root)  
  
 while (1):  
 # nodeCount (queue size) indicates  
 # number of nodes at current level.  
 nodeCount = len(q)  
 if (nodeCount == 0):  
 break  
  
 # Dequeue all nodes of current  
 # level and Enqueue all nodes  
 # of next level  
 while (nodeCount > 0):  
 node = q.popleft()  
 print(node.data, end=" ")  
  
 if (node.left != None):  
 q.append(node.left)  
  
 if (node.right != None):  
 q.append(node.right)  
 nodeCount -= 1  
  
 print()  
  
  
# Driver code  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = Node(1)  
 root.left = Node(2)  
 root.right = Node(3)  
 root.right.left = Node(4)  
 root.right.right = Node(5)  
  
 print("Level order traversal of given tree")  
 printLevelOrder(root)  
  
 root = flipBinaryTree(root)  
  
 print("\nLevel order traversal of the flipped"  
 " tree")  
 printLevelOrder(root)

**Question-3:**

Given a binary tree, print all its root-to-leaf paths without using recursion. For example, consider the following Binary Tree.

**Input:**

6

/ \\

3 5

/ \ \ 2 5 4 / \ 7 4

**Output:**

There are 4 leaves, hence 4 root to leaf paths - 6->3->2 6->3->5->7 6->3->5->4 6->5>4

**Sol:**

# Python3 program to Print root to

# leaf path without using recursion

# Helper function that allocates a new

# node with the given data and None left

# and right pointers.

class newNode:

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

self.data = data

self.left = self.right = None

# Function to print root to leaf path for a

# leaf using parent nodes stored in map

def printTopToBottomPath(curr, parent):

stk = []

# start from leaf node and keep on appending

# nodes into stack till root node is reached

while (curr):

stk.append(curr)

curr = parent[curr]

# Start popping nodes from stack

# and print them

while len(stk) != 0:

curr = stk[-1]

stk.pop(-1)

print(curr.data, end = " ")

print()

# An iterative function to do preorder

# traversal of binary tree and print

# root to leaf path without using recursion

def printRootToLeaf(root):

# Corner Case

if (root == None):

return

# Create an empty stack and

# append root to it

nodeStack = []

nodeStack.append(root)

# Create a map to store parent

# pointers of binary tree nodes

parent = {}

# parent of root is None

parent[root] = None

# Pop all items one by one. Do following

# for every popped item

# a) append its right child and set its

# parent pointer

# b) append its left child and set its

# parent pointer

# Note that right child is appended first

# so that left is processed first

while len(nodeStack) != 0:

# Pop the top item from stack

current = nodeStack[-1]

nodeStack.pop(-1)

# If leaf node encountered, print

# Top To Bottom path

if (not (current.left) and

not (current.right)):

printTopToBottomPath(current, parent)

# append right & left children of the

# popped node to stack. Also set their

# parent pointer in the map

if (current.right):

parent[current.right] = current

nodeStack.append(current.right)

if (current.left):

parent[current.left] = current

nodeStack.append(current.left)

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

# Constructed binary tree is

# 10

# / \

# 8 2

# / \ /

# 3 5 2

root = newNode(10)

root.left = newNode(8)

root.right = newNode(2)

root.left.left = newNode(3)

root.left.right = newNode(5)

root.right.left = newNode(2)

printRootToLeaf(root)

**Question-4:**

Given Preorder, Inorder and Postorder traversals of some tree. Write a program to check if they all are of the same tree.

**Examples:**

Input :

Inorder -> 4 2 5 1 3

Preorder -> 1 2 4 5 3

Postorder -> 4 5 2 3 1

Output :

Yes Explanation :

All of the above three traversals are of the same tree

1

/ \\

2 3

/ \\

4 5

**Input :**

Inorder -> 4 2 5 1 3

Preorder -> 1 5 4 2 3

Postorder -> 4 1 2 3 5

**Output :**

No

**Sol:**

# Python program to check if the given

# three traversals are of the same

# tree or not

# Function to check if all three traversals

# are of the same tree

def checktree(preorder, inorder, postorder, length):

# if the array lengths are 0,

# then all of them are obviously equal

if length == 0:

return 1

# if array lengths are 1,

# then check if all of them are equal

if length == 1:

return (preorder[0] == inorder[0]) and (inorder[0] == postorder[0])

# search for first element of preorder

# in inorder array

idx = -1

for i in range(length):

if inorder[i] == preorder[0]:

idx = i

break

if idx == -1:

return 0

# check for the left subtree

ret1 = checktree(preorder[1:], inorder, postorder, idx)

# check for the right subtree

ret2 = checktree(preorder[idx + 1:], inorder[idx + 1:],

postorder[idx:], length-idx-1)

# return 1 only if both of them are correct else 0

return (ret1 and ret2)

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

inorder = [4, 2, 5, 1, 3]

preorder = [1, 2, 4, 5, 3]

postorder = [4, 5, 2, 3, 1]

len1 = len(inorder)

len2 = len(preorder)

len3 = len(postorder)

# check if all the array lengths are equal

if (len1 == len2) and (len2 == len3):

correct = checktree(preorder, inorder,

postorder, len1)

if (correct):

print("Yes")

else:

print("No")

else:

print("No")