**Assignment 23 - Heaps and Hashing**

**Question-1:**

Given preorder of a binary tree, calculate its [**depth(or height)**](https://www.geeksforgeeks.org/write-a-c-program-to-find-the-maximum-depth-or-height-of-a-tree/) [starting from depth 0]. The preorder is given as a string with two possible characters.

1. ‘l’ denotes the leaf
2. ‘n’ denotes internal node

The given tree can be seen as a full binary tree where every node has 0 or two children. The two children of a node can ‘n’ or ‘l’ or mix of both.

**Examples :**

Input : nlnll Output : 2 Explanation :

**Sol:**

#Python program to find height of full binary tree

# using preorder

# function to return max of left subtree height

# or right subtree height

def findDepthRec(tree, n, index) :

if (index[0] >= n or tree[index[0]] == 'l'):

return 0

# calc height of left subtree (In preorder

# left subtree is processed before right)

index[0] += 1

left = findDepthRec(tree, n, index)

# calc height of right subtree

index[0] += 1

right = findDepthRec(tree, n, index)

return (max(left, right) + 1)

# Wrapper over findDepthRec()

def findDepth(tree, n) :

index = [0]

return findDepthRec(tree, n, index)

# Driver program to test above functions

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

tree= "nlnnlll"

n = len(tree)

print(findDepth(tree, n))

**Question-2:**

Given a Binary tree, the task is to print the **left view** of the Binary Tree. The left view of a Binary Tree is a set of leftmost nodes for every level.

**Examples:**

***Input:***

4

/   \\

5     2

/   \\

3     1

/  \\

6    7

***Output:****4 5 3 6*

**Sol:**

# Python program to print left view of Binary Tree

# A binary tree node

class Node:

# Constructor to create a new node

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

self.data = data

self.left = None

self.right = None

# Recursive function print left view of a binary tree

def leftViewUtil(root, level, max\_level):

# Base Case

if root is None:

return

# If this is the first node of its level

if (max\_level[0] < level):

print (root.data, end = " ")

max\_level[0] = level

# Recur for left and right subtree

leftViewUtil(root.left, level + 1, max\_level)

leftViewUtil(root.right, level + 1, max\_level)

# A wrapper over leftViewUtil()

def leftView(root):

max\_level = [0]

leftViewUtil(root, 1, max\_level)

# Driver program to test above function

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

root = Node(10)

root.left = Node(2)

root.right = Node(3)

root.left.left = Node(7)

root.left.right = Node(8)

root.right.right = Node(15)

root.right.left = Node(12)

root.right.right.left = Node(14)

leftView(root)

**Question-3:**

Given a Binary Tree, print the Right view of it.

The right view of a Binary Tree is a set of nodes visible when the tree is visited from the Right side.

**Examples:**

**Input:**

1

/     \\

2        3

/   \       /  \

4     5   6    7

\\

8

**Output**:

Right view of the tree is 1 3 7 8

**Input:**

1

/

8

/

7

**Output**:

Right view of the tree is 1 8 7

**Sol:**

# Python3 program to print right

# view of Binary Tree

from collections import deque

# A binary tree node

class Node:

# A constructor to create a new

# Binary tree Node

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

self.data = val

self.left = None

self.right = None

# Function to print Right view of

# binary tree

def rightView(root):

if root is None:

return

q = deque()

q.append(root)

while q:

# Get number of nodes for each level

n = len(q)

# Traverse all the nodes of the

# current level

while n > 0:

n -= 1

# Get the front node in the queue

node = q.popleft()

# Print the last node of each level

if n == 0:

print(node.data, end=" ")

# If left child is not null push it

# into the queue

if node.left:

q.append(node.left)

# If right child is not null push

# it into the queue

if node.right:

q.append(node.right)

# Driver code

# Let's construct the tree as

# shown in example

root = Node(1)

root.left = Node(2)

root.right = Node(3)

root.left.left = Node(4)

root.left.right = Node(5)

root.right.left = Node(6)

root.right.right = Node(7)

root.right.left.right = Node(8)

rightView(root)

**Question-4:**

Given a Binary Tree, The task is to print the **bottom view** from left to right. A node **x** is there in output if x is the bottommost node at its horizontal distance. The horizontal distance of the left child of a node x is equal to a horizontal distance of x minus 1, and that of a right child is the horizontal distance of x plus 1.

**Examples:**

**Input:**

20

/     \\

8         22

/      \\         \\

5         3        25

/    \\

10       14

**Output:** 5, 10, 3, 14, 25.

**Input:**

20

/     \\

8         22

/      \\      /   \\

5         3   4     25

/    \\

10       14

**Output:**

5 10 4 14 25.

**Explanation:**

If there are multiple bottom-most nodes for a horizontal distance from the root, then print the later one in the level traversal.

**3 and 4** are both the bottom-most nodes at a horizontal distance of 0, we need to print 4.

**Sol:**

# Python3 program to print Bottom

# View of Binary Tree

# deque supports efficient pish and pop on both ends

from collections import deque

# Tree node class

class Node:

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

self.data = key

self.hd = float('inf')

self.left = None

self.right = None

# Method that prints the bottom view.

def bottomView(root):

if (root == None):

return

# Initialize a variable 'hd' with 0

# for the root element.

hd = 0

# Store minimum and maximum horizontal distance

# so that we do not have to sort keys at the end

min\_hd, max\_hd = 0, 0

hd\_dict = dict()

# Queue to store tree nodes in level

# order traversal

q = deque()

# Assign initialized horizontal distance

# value to root node and add it to the queue.

root.hd = hd

q.append(root)

# Loop until the queue is empty (standard

# level order loop)

while q:

curr\_node = q.popleft()

# Extract the horizontal distance value

# from the dequeued tree node.

hd = curr\_node.hd

# Update the minimum and maximum hd

min\_hd = min(min\_hd, hd)

max\_hd = max(max\_hd, hd)

# Put the dequeued tree node to dictionary

# having key as horizontal distance. Every

# time we find a node having same horizontal

# distance we need to update the value in

# the map.

hd\_dict[hd] = curr\_node.data

# If the dequeued node has a left child, add

# it to the queue with a horizontal distance hd-1.

if curr\_node.left:

curr\_node.left.hd = hd - 1

q.append(curr\_node.left)

# If the dequeued node has a right child, add

# it to the queue with a horizontal distance

# hd+1.

if curr\_node.right:

curr\_node.right.hd = hd + 1

q.append(curr\_node.right)

# Traverse the map from least horizontal distance to

# most horizontal distance.

for i in range(min\_hd, max\_hd+1):

print(hd\_dict[i], end = ' ')

# Driver Code

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

root = Node(20)

root.left = Node(8)

root.right = Node(22)

root.left.left = Node(5)

root.left.right = Node(3)

root.right.left = Node(4)

root.right.right = Node(25)

root.left.right.left = Node(10)

root.left.right.right = Node(14)

print("Bottom view of the given binary tree :")

bottomView(root)