**Assignment Questions 23**

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 :

!<https://media.geeksforgeeks.org/wp-content/uploads/btree1.png>

Input : nlnnlll Output : 3

!<https://media.geeksforgeeks.org/wp-content/uploads/dia2.png>

**CODE:**

**def** calculateDepth(preorder, index):

depth **=** 0

depth **+=** 1

**if** preorder[index] **==** 'n':

depth **=** max(depth, calculateDepth(preorder, index **+** 1))

depth **=** max(depth, calculateDepth(preorder, index **+** 2))

**return** depth

**def** calculateTreeDepth(preorder):

**return** calculateDepth(preorder, 0)

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*

**Explanation:**

!<https://media.geeksforgeeks.org/wp-content/cdn-uploads/left-view.png>

***Input:***

1

/   \\

2       3

\\

4

\\

5

\\

6

**Output:** 1 2 4 5 6

**CODE:**

**class** TreeNode:

**def** \_\_init\_\_(self, val**=**0, left**=None**, right**=None**):

self**.**val **=** val

self**.**left **=** left

self**.**right **=** right

**def** printLeftViewUtil(root, level, max\_level):

**if** root **is** **None**:

**return**

**if** level **>** max\_level[0]:

print(root**.**val, end**=**" ")

max\_level[0] **=** level

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

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

**def** printLeftView(root):

max\_level **=** [0]

printLeftViewUtil(root, 1, max\_level)

print()

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

**CODE:**

**class** TreeNode:

**def** \_\_init\_\_(self, val**=**0, left**=None**, right**=None**):

self**.**val **=** val

self**.**left **=** left

self**.**right **=** right

**def** printRightViewUtil(root, level, max\_level):

**if** root **is** **None**:

**return**

**if** level **>** max\_level[0]:

print(root**.**val, end**=**" ")

max\_level[0] **=** level

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

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

**def** printRightView(root):

max\_level **=** [0]

printRightViewUtil(root, 1, max\_level)

print()

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.

**CODE:**

**from** collections **import** defaultdict

**class** Node:

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

self**.**val **=** val

self**.**hd **=** 0

self**.**left **=** **None**

self**.**right **=** **None**

**def** bottomViewUtil(root, hd, bottom\_view):

**if** root **is** **None**:

**return**

bottom\_view[hd] **=** root**.**val

bottomViewUtil(root**.**left, hd **-** 1, bottom\_view)

bottomViewUtil(root**.**right, hd **+** 1, bottom\_view)

**def** bottomView(root):

**if** root **is** **None**:

**return**

bottom\_view **=** defaultdict(int)

bottomViewUtil(root, 0, bottom\_view)

sorted\_bottom\_view **=** sorted(bottom\_view**.**items(), key**=lambda** x: x[0])

**for** \_, val **in** sorted\_bottom\_view:

print(val, end**=**" ")

print()