ASSISGNMENT 23

1. Given preorder of a binary tree, calculate its **depth(or height)** [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.

```
def calculateDepth(preorder):
    def dfs(preorder, index):
        if index == len(preorder):
            return -1

        if preorder[index] == 'l':
            return 0

        index += 1
        left_depth = dfs(preorder, index)
        right_depth = dfs(preorder, index + left_depth)
        return max(left_depth, right_depth) + 1

    return dfs(preorder, 0)
```

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.

```
def leftView(root):
    leftmost = {}

    def dfs(node, level, leftmost):
        if node is None:
            return

        if level not in leftmost:
            leftmost[level] = node.val

        dfs(node.left, level + 1, leftmost)
        dfs(node.right, level + 1, leftmost)

    dfs(root, 0, leftmost)

    sorted_leftmost = sorted(leftmost.items(), key=lambda x: x[0])

    for level, val in sorted_leftmost:
        print(val)

# Sample binary tree node definition
```

```
class Node:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```

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.

```
def rightView(root):
    rightmost = {}

    def dfs(node, level, rightmost):
        if node is None:
            return

        rightmost[level] = node.val

        dfs(node.right, level + 1, rightmost)
        dfs(node.left, level + 1, rightmost)

    dfs(root, 0, rightmost)

    sorted_rightmost = sorted(rightmost.items(), key=lambda x: x[0])

    for level, val in sorted_rightmost:
        print(val)

# Sample binary tree node definition
class Node:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```

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.

```
import heapq

def bottomView(root):
    bottomViewDict = {}

    def dfs(node, horizontal_distance, level):
        if node is None:
            return

        if horizontal_distance not in bottomViewDict or level >= bottomViewDict[horizontal_distance][1]:
            bottomViewDict[horizontal_distance] = (node.val, level)
        dfs(node.left, horizontal_distance - 1, level + 1)
        dfs(node.right, horizontal_distance + 1, level + 1)

    dfs(root, 0, 0)

    minHeap = []
    for distance, (val, level) in bottomViewDict.items():
        heapq.heappush(minHeap, (distance, val))

    while minHeap:
        distance, val = heapq.heappop(minHeap)
        print(val)

# Sample binary tree node definition
class Node:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```