Assignment 23 Solutions

Q1. 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.

Examples :

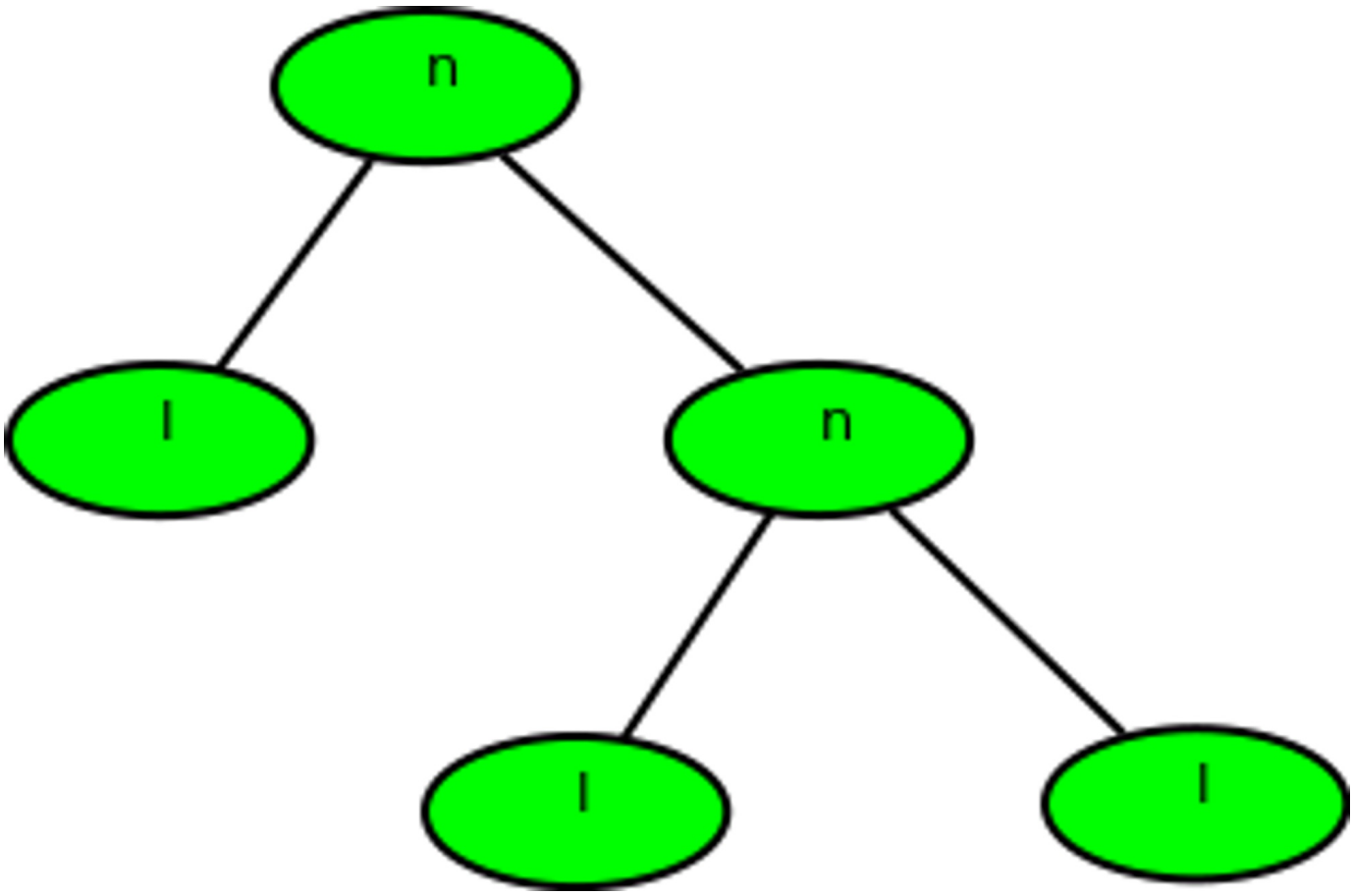
Input : nlnll

Output : 2

Explanation :

In [3]:
```
from IPython.display import Image
Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\btree1.png")
```

Out[3]:

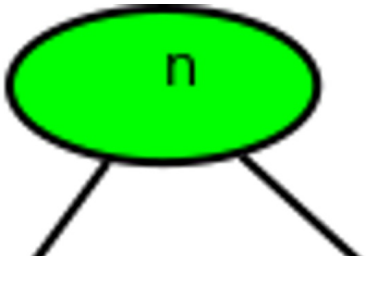


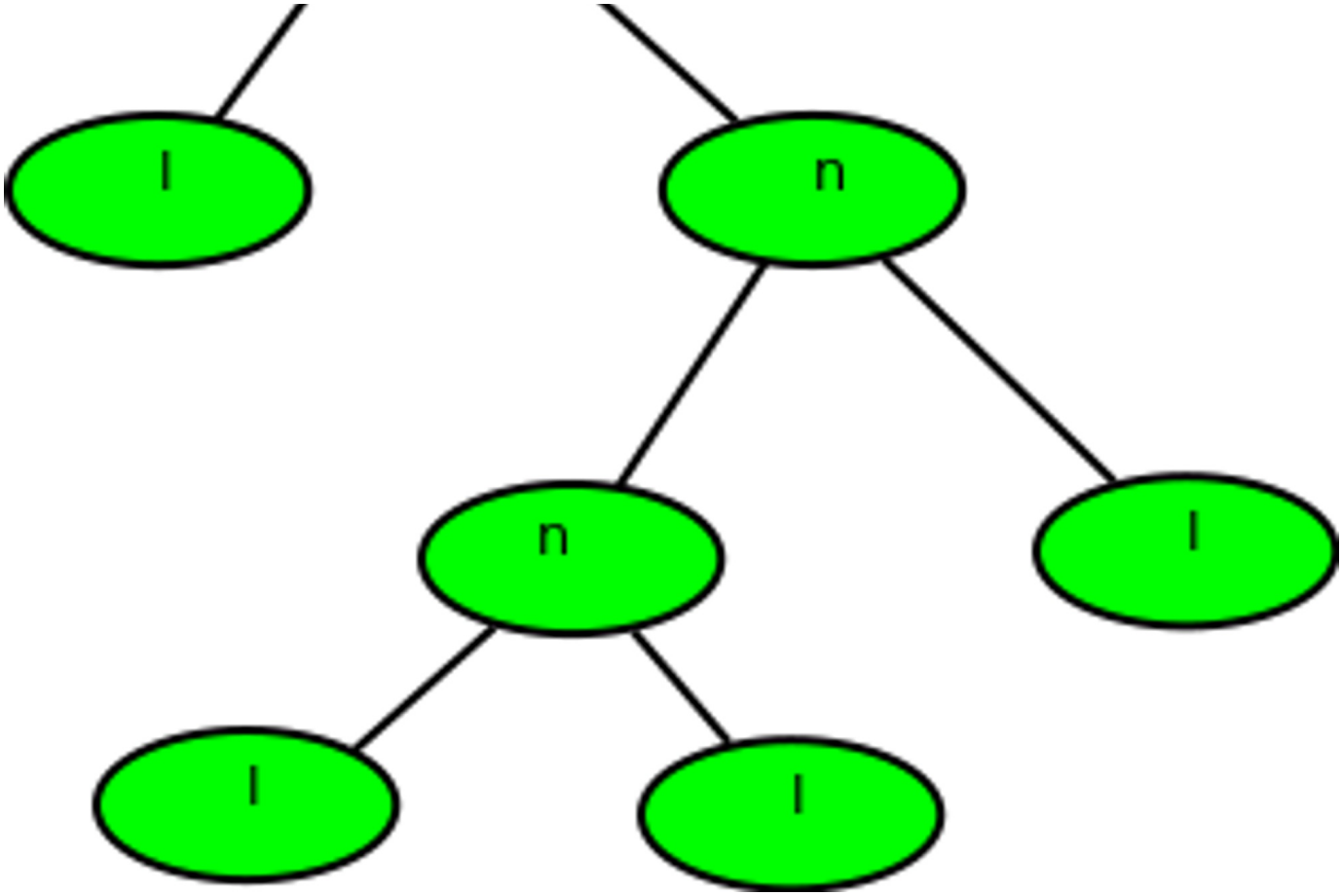
Input : nlnnlll

Output : 3

In [4]:
```
from IPython.display import Image
Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\dia2.png")
```

Out[4]:





In [13]:
```
def calculateDepth(preorder, index):
    if index >= len(preorder):
        return 0

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

    depth = 1
    depth += calculateDepth(preorder, index + 1)
    depth += calculateDepth(preorder, index + 2)

    return depth

def calculateTreeDepth(preorder):
    return calculateDepth(preorder, 0)
```

In [18]:
```
# Example 1
preorder = 'nlnll'
depth = calculateTreeDepth(preorder)
print(depth)

# Example 2
preorder = 'nlnnlll'
depth = calculateTreeDepth(preorder)
print(depth)
```

```
2
3
```

Q2. 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
```

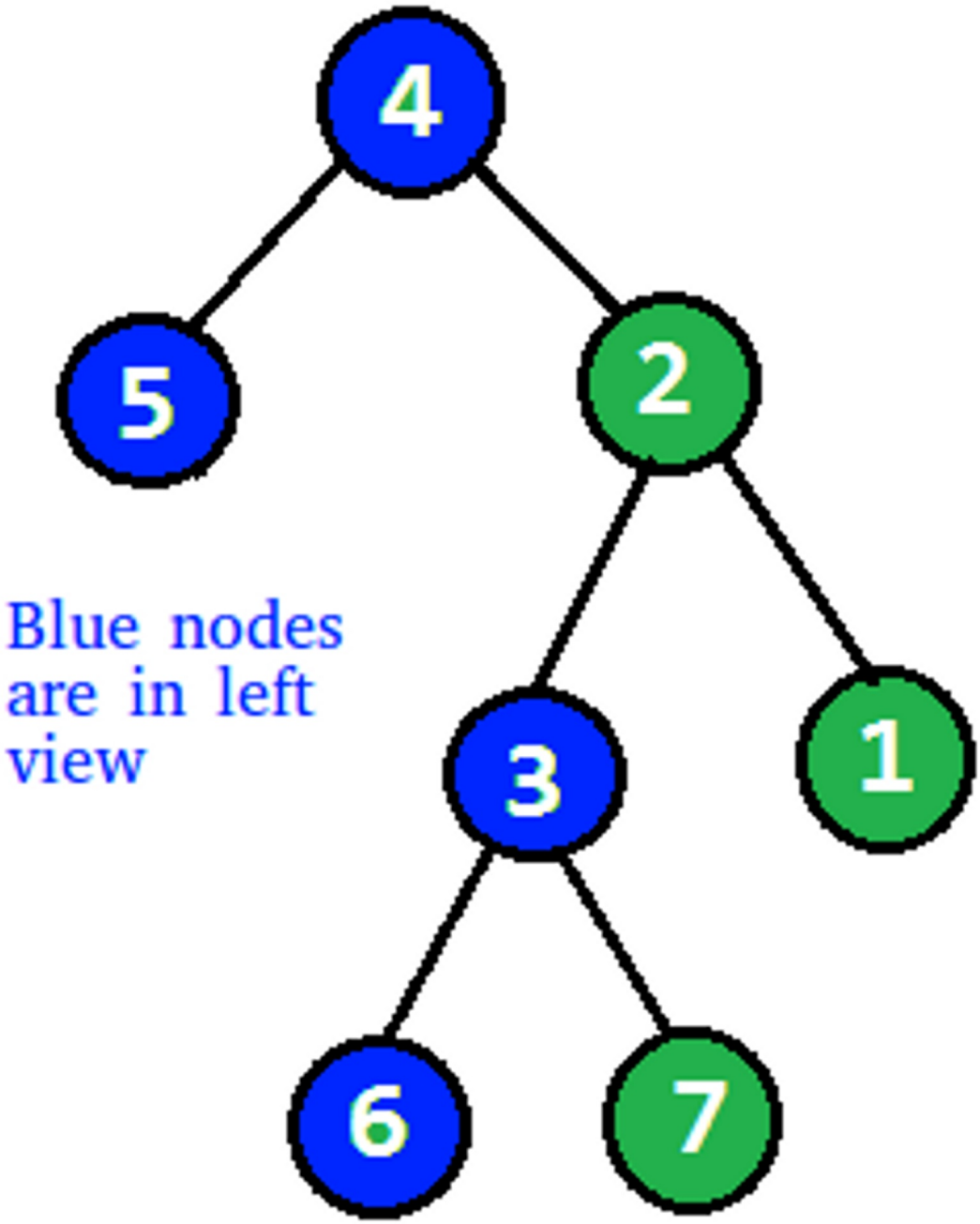
```
      /   \
     3     1
    / \
   6   7
```

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

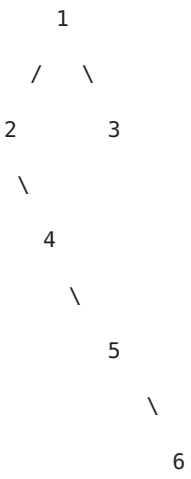
Explanation:

In [19]:
```
from IPython.display import Image
Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\left-view.png")
```

Out[19]:



Input:



Output: 1 2 4 5 6

In [32]:
```
from collections import deque

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

def leftView(root):
    if root is None:
        return []

    queue = deque()
    queue.append(root)
    leftView = []

    while queue:
        size = len(queue)

        for i in range(size):
            node = queue.popleft()

            if i == 0:
                leftView.append(node.value)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

    return leftView
```

In [41]:
```
# Create the binary tree
root = Node(4)
root.left = Node(5)
root.right = Node(2)
root.right.left = Node(3)
root.right.right = Node(1)
root.right.left.left = Node(6)
root.right.left.right = Node(7)

# Calculate the left view
result = leftView(root)

# Print the left view
for node in result:
    print(node, end=' ')
```

```
4 5 3 6
```

In [57]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def leftView(root):
    if root is None:
        return []
```

```
    leftView = []
    queue = [(root, 1)]

    while queue:
        node, level = queue.pop(0)

        if level > len(leftView):
            leftView.append(node.value)

        if node.left:
            queue.append((node.left, level + 1))
        if node.right:
            queue.append((node.right, level + 1))

    return leftView
```

In [61]:

```
# Create the binary tree
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.right = Node(4)
root.left.right.right = Node(5)
root.left.right.right.right = Node(6)

# Calculate the left view
result = leftView(root)

# Print the left view
for node in result:
    print(node, end=' ')
```

```
1 2 4 5 6
```

Q3.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

In [69]:
```
from collections import deque

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

def rightView(root):
    if root is None:
        return []

    queue = deque()
    queue.append(root)
    rightView = []

    while queue:
        size = len(queue)

        for i in range(size):
            node = queue.popleft()

            if i == size - 1:
                rightView.append(node.value)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

    return rightView
```

In [74]:
```
# Create the binary tree
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.right.right = Node(8)

# Calculate the right view
result = rightView(root)

# Print the right view
for node in result:
    print(node, end=' ')
```

```
1 3 7 8
```

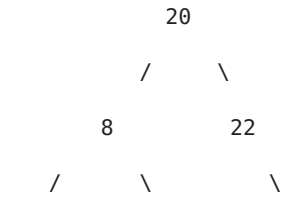
In [81]:
```
# Create the binary tree
root = Node(1)
root.left = Node(8)
root.left.left = Node(7)

# Calculate the right view
result = rightView(root)

# Print the right view
for node in result:
    print(node, end=' ')
```

```
1 8 7
```

Q4. 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.

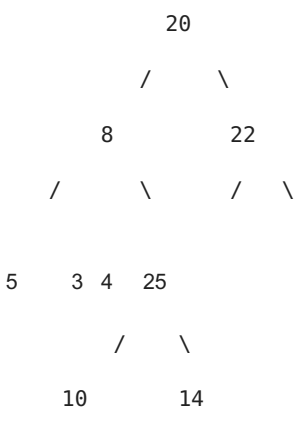
Examples:

Input:



Output: 5, 10, 3, 14, 25.

Input:



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.

In [90]:

```
from collections import deque

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

def bottomView(root):
    if root is None:
        return []

    bottomView = {}
    queue = deque()
    queue.append((root, 0))

    while queue:
        node, horizontalDistance = queue.popleft()

        bottomView[horizontalDistance] = node.value

        if node.left:
            queue.append((node.left, horizontalDistance - 1))
        if node.right:
            queue.append((node.right, horizontalDistance + 1))

    sortedHorizontalDistances = sorted(bottomView.keys())
    result = [bottomView[hd] for hd in sortedHorizontalDistances]

    return result
```

In [97]:

```
# Create the binary tree
```

```
root = Node(20)
root.left = Node(8)
root.right = Node(22)
root.left.left = Node(5)
root.left.right = Node(3)
root.right.right = Node(25)
root.left.right.left = Node(10)
root.left.right.right = Node(14)

# Calculate the bottom view
result = bottomView(root)

# Print the bottom view
for node in result:
    print(node, end=' ')
```

```
5 10 3 14 25
```

In [102…

```
# Create the binary tree
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)

# Calculate the bottom view
result = bottomView(root)

# Print the bottom view
for node in result:
    print(node, end=' ')
```

```
5 10 4 14 25
```

In [ ]: