In [6]:

```
class BinaryTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
```

In [7]:

```
btn1 = BinaryTreeNode(1)
btn2 = BinaryTreeNode(4)
btn3 = BinaryTreeNode(5)
```

In [8]:

```
btn1.left = btn2
btn1.right = btn3
```

Printing a Simple tree ¶

In [9]:

```
# preordered traversing
def printTree(root):
    if root == None:
        return
    print(root.data)
    printTree(root.left)
    printTree(root.right)
```

In [10]:

```
btn1 = BinaryTreeNode(1)
btn2 = BinaryTreeNode(4)
btn3 = BinaryTreeNode(5)
```

In [11]:

```
btn1.left = btn2
btn1.right = btn3
```

In [12]:

```
printTree(btn1)
```

```
1
4
5
```

Printing a Detailed tree

In [13]:

```
def printTreeDetailed(root):
    if root == None:
        return
    print(root.data, end=":")
    if root.left != None:
        print("L", root.left.data, end = ',')
    if root.right != None:
        print("R", root.right.data, end = '')
    print()
    printTreeDetailed(root.left)
    printTreeDetailed(root.right)
```

In [14]:

```
btn1 = BinaryTreeNode(1)
btn2 = BinaryTreeNode(4)
btn3 = BinaryTreeNode(5)
btn4 = BinaryTreeNode(2)
btn5 = BinaryTreeNode(3)

btn1.left = btn2
btn1.right = btn3
btn2.left = btn4
btn2.right = btn5
```

In [15]:

```
printTreeDetailed(btn1)
```

```
1:L 4,R 5
4:L 2,R 3
2:
3:
5:
```

Taking input for Tree

In [16]:

```
def treeInput():
    rootData = int(input())
    if rootData == -1:
        return None

    root = BinaryTreeNode(rootData)
    leftTree = treeInput()
    rightTree = treeInput()
    root.left = leftTree
    root.right = rightTree
    return root
```

In [19]:

```
root = treeInput()
printTreeDetailed(root)
```

```
7
2
3
1
-1
-1
-1
-1
4
3
-1
-1
-1
7:L 2,R 4
2:L 3,
3:L 1,
1:
4:L 3,
3:
```

Number of Nodes in a tree

In [20]:

```
def numNodes(root):
    if root == None:
        return 0
    leftCount = numNodes(root.left)
    rightCount = numNodes(root.right)
    return 1 + leftCount + rightCount
```

In [22]:

```
root = treeInput()
printTreeDetailed(root)
print("="*10)
print("Number of Nodes : ",numNodes(root))
```

```
7
9
-1
-1
8
4
-1
-1
5
-1
-1
7:L 9,R 8
9:
8:L 4,R 5
4:
5:
==========
Number of Nodes :  5
```

Post Oredered Traversing in a Tree

In [23]:

```
def printTree(root):
    if root == None:
        return
    printTree(root.left)
    printTree(root.right)
    print(root.data)

```

In [24]:

```
btn1 = BinaryTreeNode(1)
btn2 = BinaryTreeNode(2)
btn3 = BinaryTreeNode(3)
btn4 = BinaryTreeNode(4)
btn5 = BinaryTreeNode(5)

btn1.left = btn2
btn1.right = btn3
btn2.left = btn4
btn2.right = btn5
```

In [25]:

```
printTree(btn1)
```

```
4
5
2
3
1
```

In-orederd Traversing in Tree

In [26]:

```
def printTree(root):
    if root == None:
        return
    printTree(root.left)
    print(root.data)
    printTree(root.right)

```

In [29]:

```
btn1 = BinaryTreeNode(1)
btn2 = BinaryTreeNode(2)
btn3 = BinaryTreeNode(3)
btn4 = BinaryTreeNode(9)
btn5 = BinaryTreeNode(4)

btn1.left = btn2
btn1.right = btn3
btn2.left = btn4
btn2.right = btn5
```

In [30]:

```
printTree(btn1)
```

```
9
2
4
1
3
```

Sum of all Nodes of a binary Tree

In [43]:

```
def sum_of_nodes(root):
    if root == None:
        return 0
    leftSum = sum_of_nodes(root.left)
    rightSum = sum_of_nodes(root.right)
    return root.data + leftSum + rightSum
```

In [45]:

```
root = treeInput()
printTreeDetailed(root)
print("="*20)
print("Sum of Nodes : ",sum_of_nodes(root))
```

```
1
2
4
-1
-1
5
-1
-1
3
-1
-1
1:L 2,R 3
2:L 4,R 5
4:
5:
3:
====================
Sum of Nodes :  15
```

In [46]:

```
def largestData(root):
    if root == None:
        return -1
    leftLargest = largestData(root.left)
    rightLargest = largestData(root.right)
    largest = max(leftLargest, rightLargest, root.data)
    return largest
```

In [49]:

```
root = treeInput()
printTreeDetailed(root)
print("Largest Node of Tree : ",largestData(root))
```

```
1
2
-1
-1
3
4
-1
-1
5
-1
-1
1:L 2,R 3
2:
3:L 4,R 5
4:
5:
Largest Node of Tree :  5
```

Height of Binary Tree

In [18]:

```
def find_Height(root):

    if (root == None):
        return 0

    leftHeight = find_Height(root.left)
    rightHeight = find_Height(root.right)

    result = max(leftHeight, rightHeight) + 1

    return result
```

In [21]:

```
root = treeInput()
printTreeDetailed(root)
print("Height of Binary Tree : ",find_Height(root))
```

```
8
2
-1
-1
3
4
-1
-1
5
-1
-1
8:L 2,R 3
2:
3:L 4,R 5
4:
5:
Height of Binary Tree :  3
```

In [22]:

```
root = treeInput()
printTreeDetailed(root)
print("Height of Binary Tree : ",find_Height(root))
```

```
10
20
30
40
50
-1
-1
-1
-1
-1
-1
10:L 20,
20:L 30,
30:L 40,
40:L 50,
50:
Height of Binary Tree :  5
```

Number of LeafNode in a Binary Tree

In [23]:

```
def NumOfLeafNodes(root):
    if root == None:
        return 0
    if root.left == None and root.right == None:
        return 1
    numLeafLeft = NumOfLeafNodes(root.left)
    numLeafRight = NumOfLeafNodes(root.right)

    return numLeafLeft + numLeafRight
```

In [25]:

```
root = treeInput()
printTreeDetailed(root)
NumOfLeafNodes(root)
```

```
1
2
4
-1
-1
5
8
-1
-1
9
-1
-1
3
6
-1
-1
7
-1
-1
1:L 2,R 3
2:L 4,R 5
4:
5:L 8,R 9
8:
9:
3:L 6,R 7
6:
7:
```

Out[25]:

```
5
```

Printing Nodes at k depth

In [31]:

```
def printNodeDepthK(root, k):
    if root == None:
        return
    if k == 0:
        print(root.data)
        return
    leftNode = printNodeDepthK(root.left, k-1)
    rightNode = printNodeDepthK(root.right, k-1)
```

In [33]:

```
root = treeInput()
printTreeDetailed(root)
printNodeDepthK(root,3)
```

```
4
5
-1
-1
6
4
2
-1
-1
1
-1
-1
3
-1
-1
4:L 5,R 6
5:
6:L 4,R 3
4:L 2,R 1
2:
1:
3:
2
1
```

Removing leaf nodes

In [1]:

```
def remove_leaf(root):
    if root.data == None:
        return None
    if root.left==None and root.right==None:
        return None
    root.left = remove_leaf(root.left)
    root.right = remove_leaf(root.right)

    return root
```

In [17]:

```
root = treeInput()
printTreeDetailed(root)
root = remove_leaf(root)
printTreeDetailed(root)
```

```
1
2
4
-1
-1
5
8
-1
-1
9
-1
-1
3
6
-1
-1
7
-1
-1
```

Check if tree is Balanced or Not

In [18]:

```
def height(root):
    if root == None:
        return 0
    return (1+max(height(root.left), height(root.right)))
```

In [19]:

```
def isBalanced(root):
    if root == None:
        return True
    lh = height(root.left)
    rh = height(root.right)
    if lh - rh > 1 or rh - lh > 1:
        return False

    isLeftBalanced = isBalanced(root.left)
    isRightBalanced = isBalanced(root.right)
    if isLeftBalanced and isRightBalanced:
        return True
    else:
        return False
```

In [21]:

```
root = treeInput()
printTreeDetailed(root)
print(isBalanced(root))
```

```
1
2
-1
-1
3
-1
-1
1:L 2,R 3
2:
3:
True
```

In [ ]:

```

```