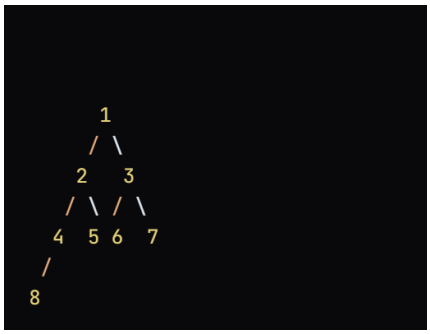
Program -17

Aim:- Write a Program to perform Boundary Traversal on BST

Working oof program :-



Breakdown of the Boundary Traversal Steps

1. **Root Node**: The root node **1** is added first.
2. **Left Boundary**: The left boundary nodes (excluding leaf nodes) are:
    - Start from **2** (the left child of **1**).
    - Then go to **4** (the left child of **2**).
    - Finally, go to **8** (the left child of **4**), but since **8** is a leaf, we stop here.
    - The left boundary collected so far: **1, 2, 4**.
3. **Leaf Nodes**: The leaf nodes are:
    - **8** (left child of **4**).
    - **5** (right child of **2**).
    - **6** (left child of **3**).
    - **7** (right child of **3**).
    - The leaf nodes collected: **8, 5, 6, 7**.

4. **Right Boundary**: The right boundary nodes (excluding leaf nodes) are:
    - Start from **3** (the right child of **1**).
    - Then go to **7** (the right child of **3**), but since **7** is a leaf, we stop here.
    - The right boundary collected (in reverse order): **3**.

Algorithm for Boundary Traversal

1. **Print the Root: Start by printing the root node.**
2. **Print the Left Boundary: Traverse the left subtree and print the nodes that are part of the left boundary (excluding leaf nodes).**
3. **Print the Leaf Nodes: Traverse the entire tree and print all leaf nodes.**
4. **Print the Right Boundary: Traverse the right subtree and print the nodes that are part of the right boundary (excluding leaf nodes). This should be done in reverse order.**

Code:-

```
import java.util.ArrayList;
import java.util.List;

// Definition for a binary tree node
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
```

```
    TreeNode(int x) {
        val = x;
        left = null;
        right = null;
    }
}

public class BoundaryTraversal {

    // Function to perform boundary traversal
    public void boundaryTraversal(TreeNode root) {
        if (root == null) {
            return; // If the tree is empty, return
        }

        List<Integer> boundary = new ArrayList<>();
        boundary.add(root.val); // Step 1: Add the root

        // Step 2: Add left boundary (excluding leaf nodes)
        addLeftBoundary(root.left, boundary);

        // Step 3: Add leaf nodes
        addLeaves(root, boundary);

        // Step 4: Add right boundary (excluding leaf nodes)
        addRightBoundary(root.right, boundary);
```

```
        // Print the boundary traversal
        for (int val : boundary) {
            System.out.print(val + " ");
        }
    }

    // Function to add left boundary nodes
    private void addLeftBoundary(TreeNode node, List<Integer> boundary) {
        while (node != null) {
            if (node.left != null || node.right != null) { // Check if it's not a leaf
                boundary.add(node.val);
            }
            node = node.left != null ? node.left : node.right; // Go down the left or right child
        }
    }

    // Function to add leaf nodes
    private void addLeaves(TreeNode node, List<Integer> boundary) {
        if (node == null) {
            return;
        }
        if (node.left == null && node.right == null) { // Check if it's a leaf
            boundary.add(node.val);
        }
        addLeaves(node.left, boundary); // Traverse left subtree
```

```
        addLeaves(node.right, boundary); // Traverse right subtree
    }

    // Function to add right boundary nodes
    private void addRightBoundary(TreeNode node, List<Integer> boundary) {
        List<Integer> temp = new ArrayList<>();
        while (node != null) {
            if (node.left != null || node.right != null) { // Check if it's not a leaf
                temp.add(node.val);
            }
            node = node.right != null ? node.right : node.left; // Go down the right
or left child
        }
        // Add right boundary in reverse order
        for (int i = temp.size() - 1; i >= 0; i--) {
            boundary.add(temp.get(i));
        }
    }

    // Main method to test the boundary traversal
    public static void main(String[] args) {
        // Create a sample binary search tree
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
```

```
        root.right.left = new TreeNode(6);
        root.right.right = new TreeNode(7);
        root.left.left.left = new TreeNode(8); // Adding more nodes for better boundary traversal

        BoundaryTraversal traversal = new BoundaryTraversal();
        System.out.println("Boundary Traversal of the binary search tree:");
        traversal.boundaryTraversal(root); // Perform boundary traversal
    }
}
```

Output:-