Perform Boundary Traversal on a Binary Search Tree (BST)

Mr. Vikas Kumar
Assistant Professor
Industry Embedded Program

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Boundary Traversal on a Binary Search Tree (BST)**

Lesson Objectives
Understand the concept of **boundary traversal** in a binary tree or BST
Learn how to print all nodes appearing on the **boundary (outline)** of the tree.
Implement boundary traversal using a combination of **traversal methods**.
Analyze time and space complexity and understand its **real-world significance** in visualization and tree layout problems.

Problem Statement:

Given a Binary Search Tree (BST), print all the nodes that form its **boundary traversal** in **anticlockwise order**, starting from the root node.

Concept

- **Boundary Traversal** of a tree means printing all the nodes that appear on the **outer boundary** (visible from outside).
- The boundary includes:
  i. **Left Boundary –** Nodes on the left edge (excluding leaves).
  ii. **Leaf Nodes –** All leaf nodes (from left to right).
  iii. **Right Boundary –** Nodes on the right edge (excluding leaves, printed in reverse).
- This traversal forms the **"perimeter view"** of the tree — similar to tracing the tree outline clockwise.

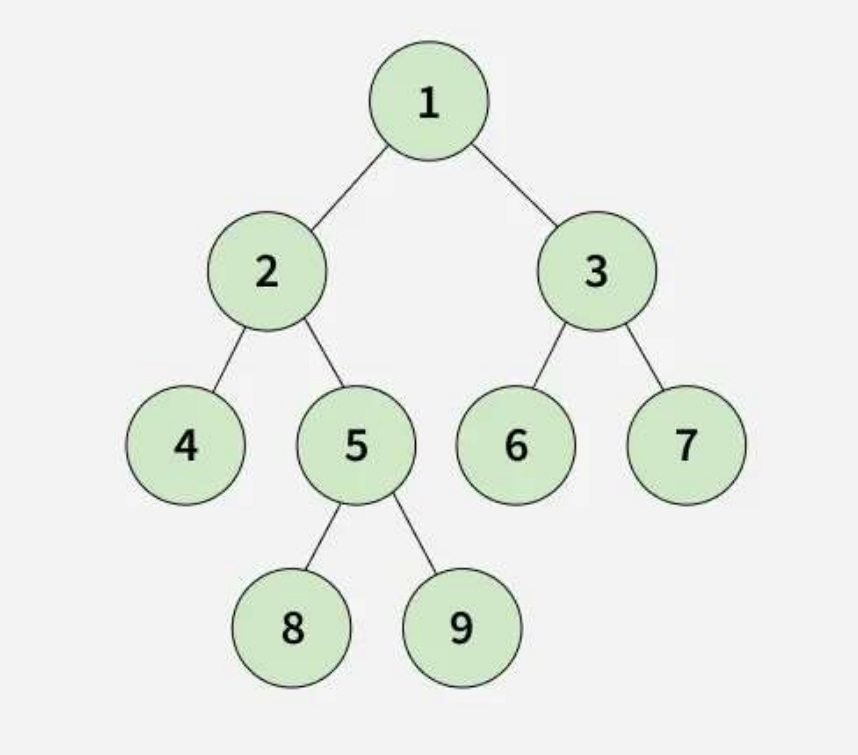
Algorithm/Logic

1. **Print Root Node** (if not null).
2. **Print Left Boundary:** Traverse down the left side until reaching leaf nodes.
3. **Print All Leaf Nodes:** Perform an inorder traversal to print every leaf node.
4. **Print Right Boundary (Bottom to Top):** Traverse down the right side, collect nodes, and print them in reverse order.
5. Combine all results to form the **boundary traversal sequence**.

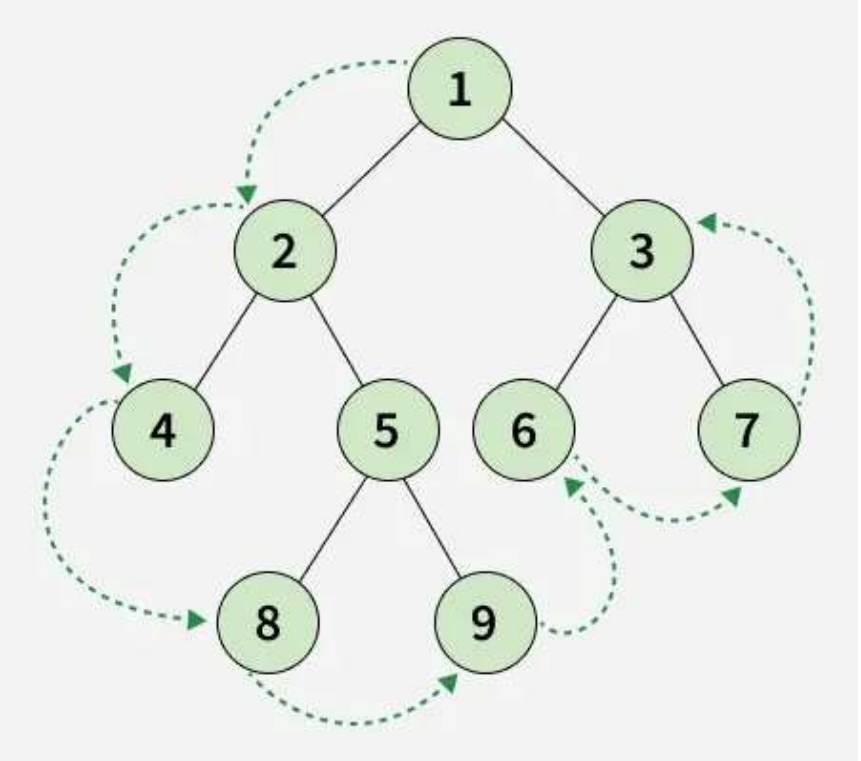
Root → Left Boundary → Leaves → Right Boundary

Visualization

Input :



Explanation:



Output : [1, 2, 4, 8, 9, 6, 7, 3]

Code Implementation

```
class Node {
    int data;
    Node left, right;

    Node(int val) {
        data = val;
        left = right = null;
    }
}

public class BoundaryTraversalExample {

    static void leftBoundary(Node root) {
        if (root == null || (root.left == null && root.right == null)) return;
        System.out.print(root.data + " ");
        if (root.left != null) leftBoundary(root.left);
        else leftBoundary(root.right);
    }

    static void printLeaves(Node root) {
        if (root == null) return;
        printLeaves(root.left);
        if (root.left == null && root.right == null)
            System.out.print(root.data + " ");
        printLeaves(root.right);
    }
```

```
    static void rightBoundary(Node root) {
        if (root == null || (root.left == null && root.right == null)) return;
        if (root.right != null) rightBoundary(root.right);
        else rightBoundary(root.left);
        System.out.print(root.data + " ");  // reverse print order
    }

    static void boundaryTraversal(Node root) {
        if (root == null) return;
        System.out.print(root.data + " "); // root
        leftBoundary(root.left);
        printLeaves(root.left);
        printLeaves(root.right);
        rightBoundary(root.right);
    }

    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        root.left.right.left = new Node(8);
        root.left.right.right = new Node(9);

        System.out.print("Boundary Traversal: ");
        boundaryTraversal(root);
    }
}
```

Output :

```
Boundary Traversal: 20 8 4 10 14 25 22
```

Time & Space Complexity

Operation	Time Complexity	Space Complexity	Explanation
Boundary Traversal	O(n)	O(h)	Each node is visited once; recursion depth = tree height (h).

Summary

- **Boundary Traversal** helps extract the **outermost visible nodes** of a tree.
- It combines **left boundary**, **leaf nodes**, and **right boundary**.
- **Time Complexity:** O(n), as all nodes are visited once.
- Useful in **visual rendering**, **map boundaries**, and **tree shape analysis**.

Practice Questions:

1. Boundary of Binary Tree — LeetCode #545

🔗 https://leetcode.com/problems/boundary-of-binary-tree/

Concept:

Print the complete boundary of a binary tree in anticlockwise order starting from the root.

Why Practice:

- Strengthens understanding of **multiple traversal combinations**.
- Helps in **decomposition of complex traversal logic** into manageable steps.

Thanks