Program -16

Aim:-Write a Program to Traverse a Tree using Level Order Traversal.

Working oof program :-

Initial State

- The queue initially contains the root node:
  - Queue: **[1]**

First Iteration

- The loop checks if the queue is empty (it's not).
- **currentNode** is set to **1** (the root node).
- The value **1** is printed.
- The left child (**2**) and right child (**3**) of **1** are added to the queue.
- Queue after processing: **[2, 3]**

Second Iteration

- The loop checks if the queue is empty (it's not).
- **currentNode** is set to **2**.
- The value **2** is printed.
- The left child (**4**) and right child (**5**) of **2** are added to the queue.
- Queue after processing: **[3, 4, 5]**

Third Iteration

- The loop checks if the queue is empty (it's not).
- **currentNode** is set to **3**.
- The value **3** is printed.

- The left child (**6**) and right child (**7**) of **3** are added to the queue.
- Queue after processing: **[4, 5, 6, 7]**

Fourth Iteration

- The loop checks if the queue is empty (it's not).
- **currentNode** is set to **4**.
- The value **4** is printed.
- Since **4** has no children, nothing is added to the queue.
- Queue after processing: **[5, 6, 7]**

Subsequent Iterations

- The process continues for nodes **5**, **6**, and **7**, printing their values and adding any children to the queue (if they exist).

Algorithm for Level Order Traversal

1. **Initialize a Queue**: Start by creating a queue to hold the nodes of the tree.
2. **Enqueue the Root**: Add the root node of the tree to the queue.
3. **While the Queue is Not Empty**:
    - Dequeue a node from the front of the queue.
    - Process the node (e.g., print its value).
    - If the dequeued node has a left child, enqueue the left child.
    - If the dequeued node has a right child, enqueue the right child.
4. **Repeat** until the queue is empty.

Code:-

```
import java.util.LinkedList;

import java.util.Queue;
```

```
// 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 LevelOrderTraversal {

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

        Queue<TreeNode> queue = new LinkedList<>(); // Create a queue
        queue.add(root); // Enqueue the root node

        while (!queue.isEmpty()) { // While the queue is not empty
            TreeNode currentNode = queue.poll(); // Dequeue the front node
```

```
            System.out.print(currentNode.val + " "); // Process the node (print its
value)

            // Enqueue left child
            if (currentNode.left != null) {
                queue.add(currentNode.left);
            }

            // Enqueue right child
            if (currentNode.right != null) {
                queue.add(currentNode.right);
            }
        }
    }

    // Main method to test the level order traversal
    public static void main(String[] args) {
        // Create a sample binary 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);

        LevelOrderTraversal traversal = new LevelOrderTraversal();
```

```
        System.out.println("Level Order Traversal of the binary tree:");
        traversal.levelOrder(root); // Perform level order traversal
    }
}
```

Output:-