TASK 7.1

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        this.val = val;
    }
}
public class LongestUnivaluePath {
    private int maxLength = 0;
    public int longestUnivaluePath(TreeNode root) {
        if (root == null) return 0;
        dfs(root);
        return maxLength;
    }
    private int dfs(TreeNode node) {
        if (node == null) return 0;
        int leftPath = dfs(node.left);
        int rightPath = dfs(node.right);
        int leftLength = 0, rightLength = 0;
        if (node.left != null && node.left.val == node.val) {
            leftLength = leftPath + 1;
        }
        if (node.right != null && node.right.val == node.val) {
            rightLength = rightPath + 1;
        }
        maxLength = Math.max(maxLength, leftLength + rightLength);
        return Math.max(leftLength, rightLength);
    }
    public static void main(String[] args) {
        TreeNode root = new TreeNode(5);
        root.left = new TreeNode(4);
        root.right = new TreeNode(5);
        root.left.left = new TreeNode(1);
        root.left.right = new TreeNode(1);
        root.right.right = new TreeNode(5);
        LongestUnivaluePath solution = new LongestUnivaluePath();
        System.out.println(solution.longestUnivaluePath(root));
    }
}
```

TASK 7.2

```
import java.util.HashMap;
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        this.val = val;
    }
}
public class PathSumIII {
    public int pathSum(TreeNode root, int targetSum) {
        HashMap<Long, Integer> prefixSumMap = new HashMap<>();
        prefixSumMap.put(0L, 1); // Initialize to handle cases where prefix sum itself equals
targetSum
        return dfs(root, 0, targetSum, prefixSumMap);
    }
    private int dfs(TreeNode node, long currentSum, int targetSum, HashMap<Long, Integer>
prefixSumMap) {
        if (node == null) return 0;
        currentSum += node.val;
        int count = prefixSumMap.getOrDefault(currentSum - targetSum, 0);
        prefixSumMap.put(currentSum, prefixSumMap.getOrDefault(currentSum, 0) + 1);
        count += dfs(node.left, currentSum, targetSum, prefixSumMap);
        count += dfs(node.right, currentSum, targetSum, prefixSumMap);
        prefixSumMap.put(currentSum, prefixSumMap.get(currentSum) - 1);
        return count;
    }
    public static void main(String[] args) {
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(5);
        root.right = new TreeNode(-3);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(2);
        root.right.right = new TreeNode(11);
        root.left.left.left = new TreeNode(3);
        root.left.left.right = new TreeNode(-2);
        root.left.right.right = new TreeNode(1);
        PathSumIII solution = new PathSumIII();
        System.out.println(solution.pathSum(root, 8)); // Output: 3
    }
}
```

TASK 7.3

```
import java.util.*;
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        this.val = val;
    }
}
public class BinaryTreeLevelOrderTraversal {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> currentLevel = new ArrayList<>();
            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll();
                currentLevel.add(currentNode.val);
                if (currentNode.left != null) queue.offer(currentNode.left);
                if (currentNode.right != null) queue.offer(currentNode.right);
            }
            result.add(currentLevel);
        }
        return result;
    }
    public static void main(String[] args) {
        // Example usage:
```

```
        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.right = new TreeNode(6);
        BinaryTreeLevelOrderTraversal traversal = new BinaryTreeLevelOrderTraversal();
        List<List<Integer>> levels = traversal.levelOrder(root);
        System.out.println("Level order traversal:");
        for (List<Integer> level : levels) {
            System.out.println(level);
        }
    }
}
```