Day13

Assignment 1: Balanced Binary Tree Check

Write a function to check if a given binary tree is balanced. A balanced tree is one where the height of two subtrees of any node never differs by more than one.

A)

Certainly! Below is the Java implementation of the function to check if a given binary tree is balanced. The approach is similar to the one described in the Python version: we use a helper function to recursively determine if subtrees are balanced and to calculate their heights simultaneously.

Java Code :

```
 Package packageBinaryTree;

public class TreeNode {
            int val;
              TreeNode left;
              TreeNode right;

              public TreeNode(int x) {
                 val = x;
              }
            }
packageBinaryTree;

public class BalancedBinaryTree {
            public boolean isBalanced(TreeNode root) {
      return checkBalanceAndHeight(root).balanced;
   }

   private BalanceStatusWithHeight checkBalanceAndHeight(TreeNode node) {
      if (node == null) {
         return new BalanceStatusWithHeight(true, 0);
      }

      BalanceStatusWithHeight leftResult = checkBalanceAndHeight(node.left);
      if (!leftResult.balanced) {
         return new BalanceStatusWithHeight(false, 0);
      }

      BalanceStatusWithHeight rightResult = checkBalanceAndHeight(node.right);
      if (!rightResult.balanced) {
         return new BalanceStatusWithHeight(false, 0);
      }

      boolean balanced = Math.abs(leftResult.height - rightResult.height) <= 1;
      int height = Math.max(leftResult.height, rightResult.height) + 1;
```

```
        return new BalanceStatusWithHeight(balanced, height);
    }

    private static class BalanceStatusWithHeight {
        boolean balanced;
        int height;

        BalanceStatusWithHeight(boolean balanced, int height) {
            this.balanced = balanced;
            this.height = height;
        }
    }

    public static void main(String[] args) {
        // Example usage:
        // Constructing a simple balanced binary tree
        //       1
        //      / \
        //     2   3
        //    / \
        //   4   5
        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);

        BalancedBinaryTree treeChecker = new BalancedBinaryTree();
        System.out.println(treeChecker.isBalanced(root));  // Output: true
    }
}
```

Output:

```
 True
```

This indicates that the binary tree constructed in the main method is balanced, as confirmed by the isBalanced method.

Explanation:

1. TreeNode Class: This class defines the structure of a node in the binary tree with an initializer that sets the value, left child, and right child.

2. isBalanced Function: This function determines if the tree is balanced by calling an inner helper function check_balance_and_height.

3. check_balance_and_height Function:

- It recursively checks each node to determine if the subtrees are balanced and computes their heights.

- If a node is None, it returns True for balanced and 0 for height.

- It recursively checks the left and right subtrees.

- It determines if the current node is balanced by checking if the left and right subtrees are balanced and if the height difference between them is at most 1.

- It calculates the height of the current node as one more than the height of its taller subtree.

4. The main function isBalanced returns the balanced status obtained from the root node.

This approach ensures that each node is visited only once, making the time complexity O(n), where n is the number of nodes in the tree.

Summary:

The code provides functionality to check if a binary tree is balanced.

It utilizes recursion to traverse the tree and calculate the height of each subtree.

If the height difference between left and right subtrees of any node is greater than one, the tree is considered unbalanced.

The example usage demonstrates how to construct a simple balanced binary tree and check its balance status.