```java
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

import java.util.*;

class Program {
  // Average case: when the tree is balanced
  // O(n) time | O(n) space - where n is the number of nodes in the Binary Tree
  public static int allKindsOfNodeDepths(BinaryTree root) {
    Map<BinaryTree, Integer> nodeCounts = new HashMap<BinaryTree, Integer>();
    Map<BinaryTree, Integer> nodeDepths = new HashMap<BinaryTree, Integer>();
    addNodeCounts(root, nodeCounts);
    addNodeDepths(root, nodeDepths, nodeCounts);
    return sumAllNodeDepths(root, nodeDepths);
  }

  public static int sumAllNodeDepths(BinaryTree node, Map<BinaryTree, Integer> nodeDepths) {
    if (node == null) return 0;
    return sumAllNodeDepths(node.left, nodeDepths) + sumAllNodeDepths(node.right, nodeDepths) + nodeDepths.get(node);
  }

  public static void addNodeDepths(BinaryTree node, Map<BinaryTree, Integer> nodeDepths, Map<BinaryTree, Integer> nodeCounts) {
    nodeDepths.put(node, 0);
    if (node.left != null) {
      addNodeDepths(node.left, nodeDepths, nodeCounts);
      nodeDepths.put(node, nodeDepths.get(node) + nodeDepths.get(node.left) + nodeCounts.get(node.left));
    }
    if (node.right != null) {
      addNodeDepths(node.right, nodeDepths, nodeCounts);
      nodeDepths.put(node, nodeDepths.get(node) + nodeDepths.get(node.right) + nodeCounts.get(node.right));
    }
  }

  public static void addNodeCounts(BinaryTree node, Map<BinaryTree, Integer> nodeCounts) {
    nodeCounts.put(node, 1);
    if (node.left != null) {
      addNodeCounts(node.left, nodeCounts);
      nodeCounts.put(node, nodeCounts.get(node) + nodeCounts.get(node.left));
    }
    if (node.right != null) {
      addNodeCounts(node.right, nodeCounts);
      nodeCounts.put(node, nodeCounts.get(node) + nodeCounts.get(node.right));
    }
  }

  static class BinaryTree {
    int value;
    BinaryTree left;
    BinaryTree right;

    public BinaryTree(int value) {
      this.value = value;
      left = null;
      right = null;
    }
  }
}
```