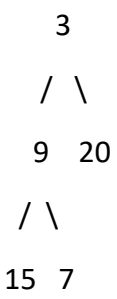
Balanced Binary Tree LeetCode

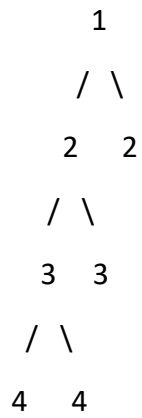
A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

Example:

Balanced Tree:



Unbalanced Tree:



Approach 1: Public function to check if a binary tree is balanced

- In the **isBalanced** function, we check if a binary tree is balanced using a recursive approach.
- We calculate the height of the left and right subtrees recursively and check if their absolute difference is less than or equal to 1.
- If both subtrees are balanced and the current tree is balanced, we return **true**; otherwise, we return **false**.

Time Complexity: O(N), where N is the number of nodes in the binary tree. This is because it visits each node in the tree only once and calculates the height of each subtree efficiently by using recursion.

Space Complexity: O(H), where H is the height of the binary tree due to the function call stack.

Approach 2: Public function to check if a binary tree is balanced using an optimized recursive approach

- In the **isBalancedHelper** function, we check if a binary tree is balanced and compute its height simultaneously using a recursive approach.
- The function returns a pair with the first element representing whether the tree is balanced and the second element representing its height.
- At each node, we recursively calculate the left subtree's balance status and height and the right subtree's balance status and height.
- We check if the current tree is balanced based on the heights of its subtrees and the absolute difference between them.
- If both subtrees are balanced and the current tree is balanced, we return **true** and the maximum height of the subtrees plus 1; otherwise, we return **false** and the maximum height of the subtrees plus 1.
- The **isBalancedOptimized** function calls **isBalancedHelper** and returns the balance status part of the pair.

Time Complexity: O(N), where N is the number of nodes in the binary tree. We visit each node once.

Space Complexity: O(H), where H is the height of the binary tree due to the function call stack.