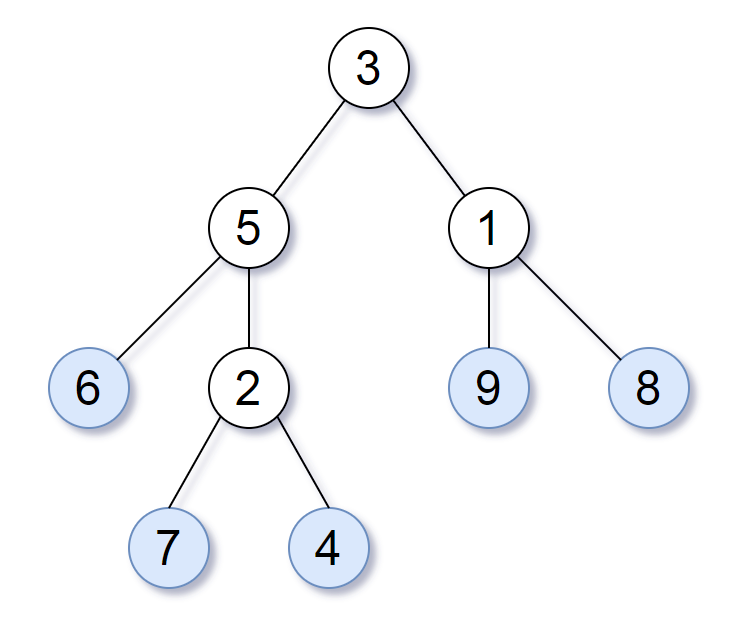
<https://leetcode.com/problems/leaf-similar-trees/description/>

Consider all the leaves of a binary tree, from left to right order, the values of those leaves form a **leaf value sequence.**

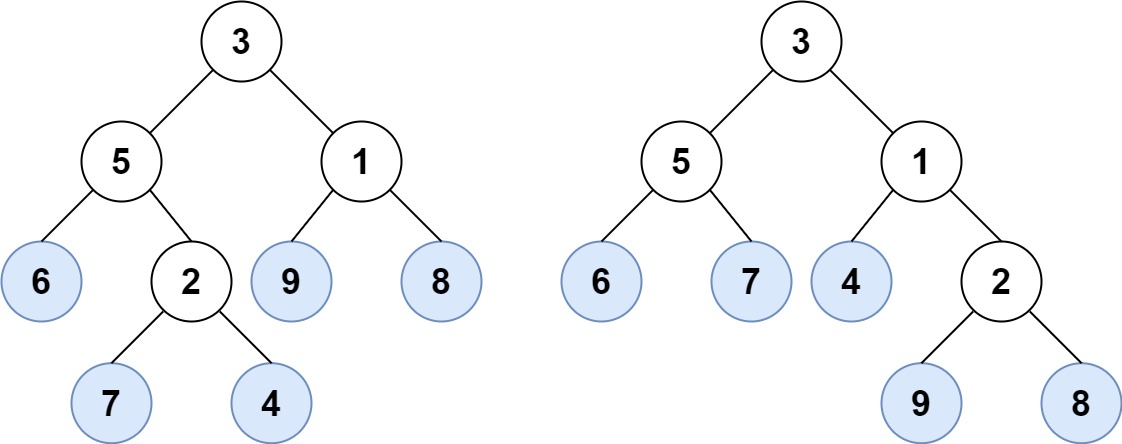


For example, in the given tree above, the leaf value sequence is (6,7,4,9,8)

Two binary trees are considered **leaf-similar** if their leaf value sequence is the same.

Return true if and only if the two given trees with head nodes root1 and root2 are leaf-similar.

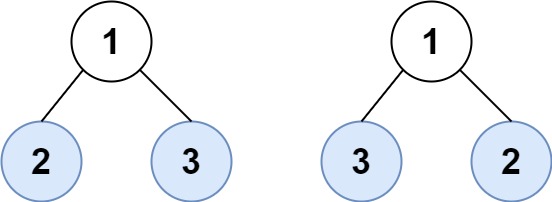
**Example 1:**



**Input:** root1 = [3,5,1,6,2,9,8,null,null,7,4], root2 = [3,5,1,6,7,4,2,null,null,null,null,null,null,9,8]

**Output:** true

**Example 2:**



**Input:** root1 = [1,2,3], root2 = [1,3,2]

**Output:** false

**Constraints:**

The number of nodes in each tree will be in the range [1,200].

Both of the given trees will have values in the range [0,200].

**Attempt 1: 2024-08-05**

**Solution 1: DFS (10 min)**

**和L366.Find Leaves of Binary Tree的剥洋葱解法一模一样，只是不需要把已经加入结果的节点变成null而已**

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public boolean leafSimilar(TreeNode root1, TreeNode root2) {

        List<Integer> list1 = new ArrayList<>();

        List<Integer> list2 = new ArrayList<>();

        helper(root1, list1);

        helper(root2, list2);

        return list1.equals(list2);

    }

    private void helper(TreeNode root, List<Integer> list) {

        if(root == null) {

            return;

        }

        if(root.left == null && root.right == null) {

            list.add(root.val);

            return;

        }

        helper(root.left, list);

        helper(root.right, list);

    }

}

**Refer to chatGPT**

// Definition for a binary tree node.

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode() {}

TreeNode(int val) {

this.val = val;

}

TreeNode(int val, TreeNode left, TreeNode right) {

this.val = val;

this.left = left;

this.right = right;

}

}

import java.util.ArrayList;

import java.util.List;

public class Solution {

public boolean leafSimilar(TreeNode root1, TreeNode root2) {

List<Integer> leaves1 = new ArrayList<>();

List<Integer> leaves2 = new ArrayList<>();

getLeaves(root1, leaves1);

getLeaves(root2, leaves2);

return leaves1.equals(leaves2);

}

private void getLeaves(TreeNode node, List<Integer> leaves) {

if (node == null) {

return;

}

if (node.left == null && node.right == null) {

leaves.add(node.val);

} else {

getLeaves(node.left, leaves);

getLeaves(node.right, leaves);

}

}

}

**Explanation:**

TreeNode Class:

This class defines the structure of a binary tree node with three constructors: a default constructor, a constructor that takes a value, and a constructor that takes a value and left and right children.

Solution Class:

Contains the leafSimilar method that takes two TreeNode objects (root1 and root2) as input.

Uses two ArrayList objects (leaves1 and leaves2) to store the leaf values of the two trees.

getLeaves Method:

This helper method traverses the tree and collects the leaf values.

If the node is null, it returns.

If the node is a leaf (both left and right children are null), it adds the node's value to the list.

Otherwise, it recursively calls itself on the left and right children.

leafSimilar Method:

Calls the getLeaves method for both trees to populate leaves1 and leaves2.

Compares the two lists using the equals method to check if the leaf sequences are identical.

Returns true if they are the same, false otherwise.

**Refer to**

[L366.Find Leaves of Binary Tree (Ref.L104)](note://WEBaec386e979a0193801a395a9a33ba86d)