JavaScript Leaf-Similar Trees

Challenge

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

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.

1st Example

```
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
```

2nd Example

```
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]`.

Solution

```
const leafSimilar = (root1, root2) => {
    let leaf1 = [],
        leaf2 = [];

    const dfs = (node, leaf) => {
        if (!node) return;

        if (!node.left && !node.right) {
            leaf.push(node.val);

            return;
        }

        dfs(node.left, leaf);

        dfs(node.right, leaf);
    };

    dfs(root1, leaf1);

    dfs(root2, leaf2);

    return leaf1.join('_') == leaf2.join('_');
};
```

Explanation

I've defined a function called `leafSimilar` that takes in two binary tree roots, `root1` and `root2`, as parameters. The purpose of this function is to check if the leaves of both trees are similar.

Inside the function, two empty arrays `leaf1` and `leaf2` are declared. These arrays will store the leaf values of `root1` and `root2`, respectively.

The function also defines a helper function called `dfs` (depth-first search) which takes in a `node` and a `leaf` array as parameters.

Within the `dfs` function, it first checks if the `node` is null. If it is, the function returns, as there are no more nodes to process.

Next, it checks if the `node` has no left and right children, indicating that it is a leaf node. If it is, the value of the `node` is pushed to the `leaf` array and the function returns.

If the `node` is not a leaf node, the `dfs` function is recursively called for its left child and right child, passing the same `leaf` array.

Once the `dfs` function is defined, it is called twice with `root1` and `root2` as the nodes, and `leaf1` and `leaf2` as the leaf arrays, respectively.

Finally, the function checks if the joined string representation of the `leaf1` array, with elements separated by `'_'`, is equal to the joined string representation of the `leaf2` array. If they are equal, it returns `true`, indicating that the leaves of both trees are similar. Otherwise, it returns `false`.

In summary, the `leafSimilar` function checks if the leaves of two binary trees are similar. It uses a depth-first search (DFS) approach to traverse the trees and store the leaf values in separate arrays. It then compares the arrays to determine if the leaves are similar.

Author: Trevor Morin