BINARY TREE (DFS)

Problem – 872. Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Problem Statement

- You are given two trees
- The goal is to compare if they have the same leaves
- The leaves should be in the same order
- Example:

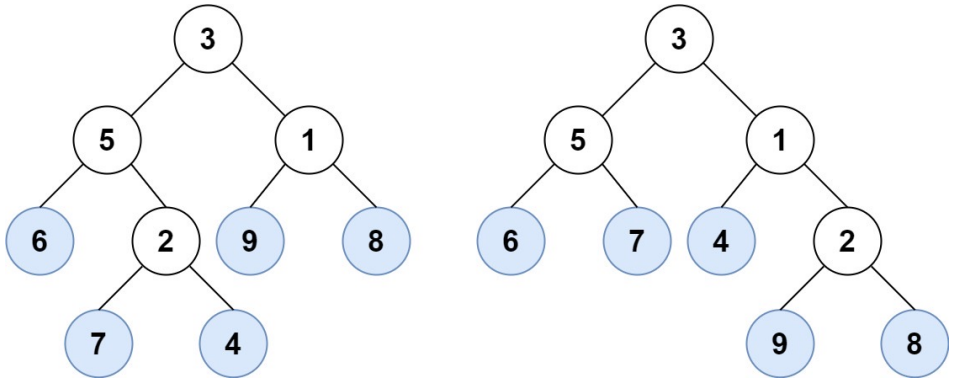
  First tree:

  leaves = 6,7,4,9,8 (blue nodes)

  Second tree:

  leaves = 6,7,4,9,8

- Return true if the leaves are the same



Solution – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Solution

- Get the first leaf value from tree 1
- Get the first leaf value from tree 2
- Compare, if they are different, return false immediately
- Otherwise, continue finding the next leaf value for tree 1 and 2

Implementation

- Create two stacks **stack<TreeNode*> left** and **stack<TreeNode*> right**
- Add the

Code – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Code

Time: **O(n + m)** where n and m are the numbers of nodes for trees 1 and 2 Space: **O(h1 + h2)** where h1 and h2 represents the height of the tree

```
// returns the value of the leaf, or -1 if empty
int getLeaf(stack<TreeNode*>& tree) {
    // tree is a reference, we will always pop an element from it
    while(!tree.empty()) {
        // get the top element from the stack
        TreeNode* node = tree.top();
        // already visited, so remove from stack
        tree.pop();
        // is this a leaf?
        if (!node->left && !node->right) {
            // yes, return the value
            return node->val;
        }
        // push the right FIRST to the stack
        if (node->right) tree.push(node->right);
        // left should be on top of the stack
        if (node->left) tree.push(node->left);
    }
    return -1;
}
```

```
bool leafSimilar(TreeNode* root1, TreeNode* root2) {
    // initialize the stacks, add root1 and root2
    std::stack<TreeNode*> leftTree, rightTree;
    leftTree.push(root1);
    rightTree.push(root2);

    while(true) {
        // get the leaves to compare
        int leaf1 = getLeaf(leftTree);
        int leaf2 = getLeaf(rightTree);
        // exit immediately if one leaf is different
        if (leaf1 != leaf2) return false;
        // stop when there are no leaves left
        if (leaf1 == -1 || leaf2 == -1) break;
    }
    return true;
}
```

Problem – 1448. Count Good Nodes in Binary Tree

Easy

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Problem Statement

- You are given a binary tree and have to find "**good**" **nodes**
- A **good node** is a **node** where the values in the path are always than the **node**
- The **root node** is always a **good node**
- Example:

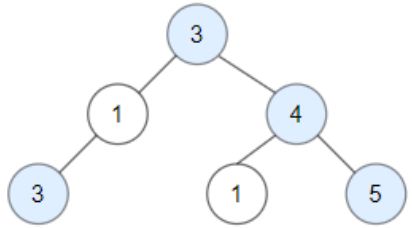
  - root 3 is a good node

  left side:

  - left **leaf 1** is not a good node because **1 < 3**
  - **leaf 3** is a good node because **3 > 1** and **3 == 3**

  right side:

  - **leaf 4** is a good node because **4 > 3**
  - **leaf 1** is not a good node because **1 < 4**
  - **leaf 5** is a good node because **5 > 4 > 3**



Solution – Count Good Nodes in Binary Tree

Easy

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Solution

- ...

Code – Count Good Nodes in Binary Tree

Easy

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Code

Time: **O(n + m)** where n and m are the numbers of nodes for trees 1 and 2 Space: **O(h1 + h2)** where h1 and h2 represents the height of the tree

```
// returns the value of the leaf, or -1 if empty
int getLeaf(stack<TreeNode*>& tree) {
    // tree is a reference, we will always pop an element from it
    while(!tree.empty()) {
        // get the top element from the stack
        TreeNode* node = tree.top();
        // already visited, so remove from stack
        tree.pop();
        // is this a leaf?
        if (!node->left && !node->right) {
            // yes, return the value
            return node->val;
        }
        // push the right FIRST to the stack
        if (node->right) tree.push(node->right);
        // left should be on top of the stack
        if (node->left) tree.push(node->left);
    }
    return -1;
}
```

```
bool leafSimilar(TreeNode* root1, TreeNode* root2) {
    // initialize the stacks, add root1 and root2
    std::stack<TreeNode*> leftTree, rightTree;
    leftTree.push(root1);
    rightTree.push(root2);

    while(true) {
        // get the leaves to compare
        int leaf1 = getLeaf(leftTree);
        int leaf2 = getLeaf(rightTree);
        // exit immediately if one leaf is different
        if (leaf1 != leaf2) return false;
        // stop when there are no leaves left
        if (leaf1 == -1 || leaf2 == -1) break;
    }
    return true;
}
```