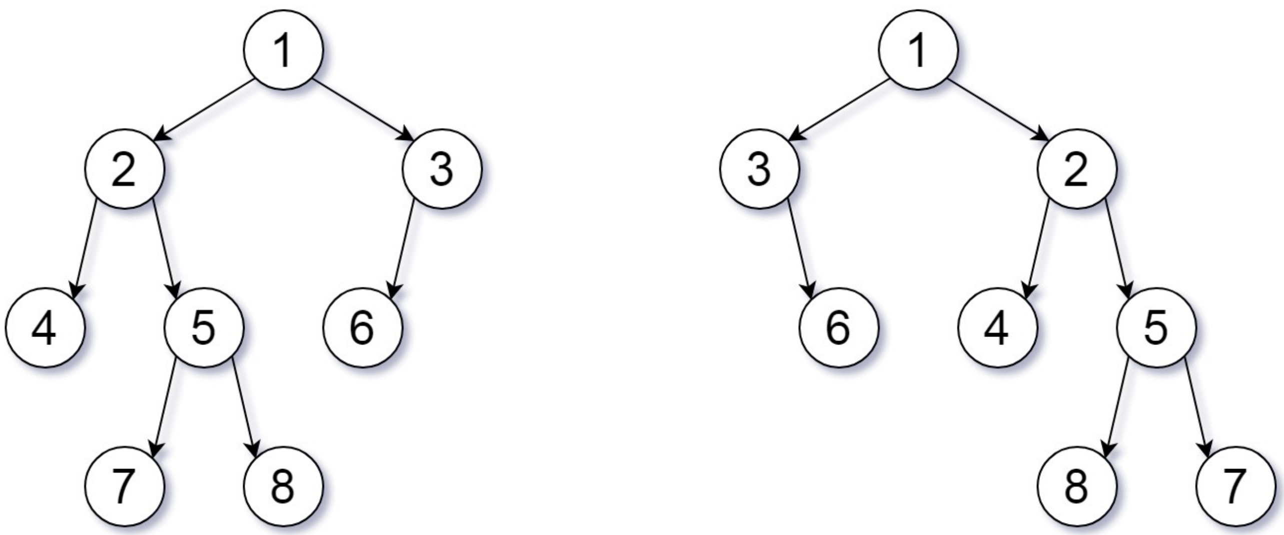
951. Flip Equivalent Binary Trees

For a binary tree **T**, we can define a **flip operation** as follows: choose any node, and swap the left and right child subtrees.

A binary tree **X** is *flip equivalent* to a binary tree **Y** if and only if we can make **X** equal to **Y** after some number of flip operations.

Given the roots of two binary trees `root1` and `root2`, return `true` if the two trees are flip equivalent or `false` otherwise.

Example 1:




```
Input: root1 = [1,2,3,4,5,6,null,null,null,7,8], root2 =
[1,3,2,null,6,4,5,null,null,null,null,8,7]
Output: true
Explanation: We flipped at nodes with values 1, 3, and 5.
```

Example 2:

```
Input: root1 = [], root2 = []
Output: true
```

Example 3:

```
Input: root1 = [], root2 = [1]
Output: false
```

Constraints:

- The number of nodes in each tree is in the range `[0, 100]`.
- Each tree will have **unique node values** in the range `[0, 99]`.

951. Flip Equivalent Binary Trees

```
/*
    DFS: Recursion
    Time complexity: O(n)
    Space complexity: O(n)
    n: number of nodes
*/
class Solution{
    public:
        bool flipEquiv(TreeNode* root1, TreeNode* root2){
            if(!root1 || !root2) return root1==root2;

            if(root1->val==root2->val)
                return flipEquiv(root1->left,root2->left) && flipEquiv(root1->right,root2->right)
                    || flipEquiv(root1->left,root2->right) && flipEquiv(root1->right,root2->left);

            return false;
        }
};
```

951. Flip Equivalent Binary Trees

```
/*
   DFS: Iterative
   Time complexity: O(n)
   Space complexity: O(n)
   n: number of nodes
*/
class Solution{
   public:
      bool check(TreeNode* node1,TreeNode* node2){
         if(!node1 || !node2) return node1==node2;
         return node1->val == node2->val;
      }

      bool flipEquiv(TreeNode* root1, TreeNode* root2){
         std::stack<std::pair<TreeNode*,TreeNode*>> st;
         st.push({root1,root2});

         while(!st.empty()){
            auto[node1,node2]=st.top();
            st.pop();

            // If both nodes are null, go the next level
            if(!node1 && !node2) continue;

            // If one of the nodes is null and the other is not null
            // means the current subtrees have not same structure
            // so the trees are not the same
            if(!node1 || !node2) return false;

            // If both nodes are not null, check their values
            // This done in the function `check(TreeNode*,TreeNode*)`
            // return `false`, if the have node the same values
            if(!check(node1,node2)) return false;
```

```
        // Check if the left and right children are the same, il we leave
        // the them as they are (no swap)
        else if(check(node1->left,node2->left) && check(node1->right,node2->right)){
            st.push({node1->left,node2->left});
            st.push({node1->right,node2->right});
        }
        // Check if the left and right children are the same, il we swap them
        else if(check(node1->left,node2->right) && check(node1->right,node2->left)) {
            st.push({node1->left,node2->right});
            st.push({node1->right,node2->left});
        }
        // If they are the same in return false
        else return false;
    }

    // If no false is returned, means:
    // Both given binary trees are equivalent, either with swap ot not
    return true;
  }
};
```

951. Flip Equivalent Binary Trees

```
/*
    Canonical forms
    Time complexity: O(n)
    Space complexity: O(n)
    n: number of nodes
*/
class Solution {
    public:
        void canonical_form(TreeNode*& root){
            if(!root) return;

            // DFS preorder
            canonical_form(root->left);
            canonical_form(root->right);

            // If no right child, back to parent
            if(!root->right) return;

            // If no left child
            if(!root->left){
                root->left=root->right; // right child become left child
                root->right=nullptr; // No right child
                return;
            }

            // If both of left and right child exist
            TreeNode* left=root->left;
            TreeNode* right=root->right;

            // If left child's value is greater than right child's value
            if(left->val<right->val){
                root->left=right; // parent's left child points to the parent's right child
                root->right=left; // parent's right child points to the parent's left child
            }
        }
```

```
    bool flipEquiv(TreeNode* root1, TreeNode* root2){
        canonical_form(root1);
        canonical_form(root2);

        if(!root1 || !root2) return root1==root2;
        if(root1->val==root2->val)
            return flipEquiv(root1->left,root2->left) && flipEquiv(root1->right,root2->right);
        return false;
    }
};
```