**Problem statement 1:**

You are given the root of a binary tree that consists of exactly 3 nodes: the root, its left child, and its right child.

Return true if the value of the root is equal to the sum of the values of its two children, or false otherwise.

Example 1:

Input: root = [10,4,6]

Output: true

Explanation: The values of the root, its left child, and its right child are 10, 4, and 6, respectively.

10 is equal to 4 + 6, so we return true.

Example 2:

Input: root = [5,3,1]

Output: false

Explanation: The values of the root, its left child, and its right child are 5, 3, and 1, respectively.

5 is not equal to 3 + 1, so we return false.

Constraints:

The tree consists only of the root, its left child, and its right child.

-100 <= Node.val <= 100

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem Statement 2:**

Given the root of a binary tree, return the preorder traversal of its nodes' values.

Example 1:

Input: root = [1,null,2,3]

Output: [1,2,3]

Example 2:

Input: root = []

Output: []

Example 3:

Input: root = [1]

Output: [1]

Constraints:

The number of nodes in the tree is in the range [0, 100].

-100 <= Node.val <= 100

Follow up: Recursive solution is trivial, could you do it iteratively?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem Statement 3:**

You are given the root of a full binary tree with the following properties:

Leaf nodes have either the value 0 or 1, where 0 represents False and 1 represents True.

Non-leaf nodes have either the value 2 or 3, where 2 represents the boolean OR and 3 represents the boolean AND.

The evaluation of a node is as follows:

If the node is a leaf node, the evaluation is the value of the node, i.e. True or False.

Otherwise, evaluate the node's two children and apply the boolean operation of its value with the children's evaluations.

Return the boolean result of evaluating the root node.

A full binary tree is a binary tree where each node has either 0 or 2 children.

A leaf node is a node that has zero children.

Example 1:

Input: root = [2,1,3,null,null,0,1]

Output: true

Explanation: The above diagram illustrates the evaluation process.

The AND node evaluates to False AND True = False.

The OR node evaluates to True OR False = True.

The root node evaluates to True, so we return true.

Example 2:

Input: root = [0]

Output: false

Explanation: The root node is a leaf node and it evaluates to false, so we return false.

Constraints:

The number of nodes in the tree is in the range [1, 1000].

0 <= Node.val <= 3

Every node has either 0 or 2 children.

Leaf nodes have a value of 0 or 1.

Non-leaf nodes have a value of 2 or 3.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem Statement 4:**

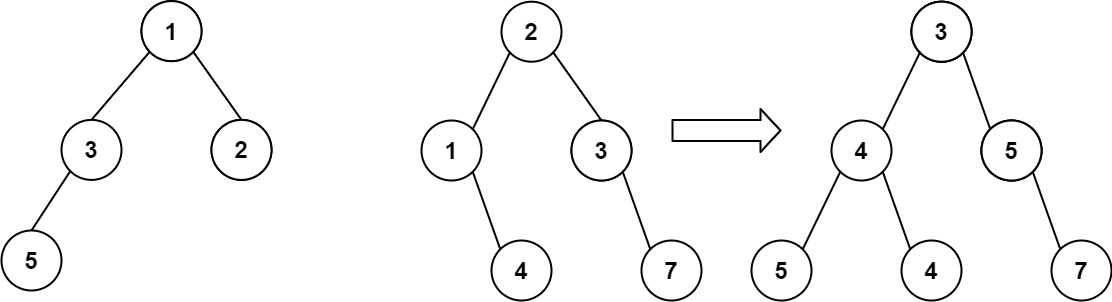
You are given two binary trees root1 and root2.

Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.

Return the merged tree.

Note: The merging process must start from the root nodes of both trees.

Example 1:



Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]

Output: [3,4,5,5,4,null,7]

Example 2:

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

Output: [2,2]

Constraints:

The number of nodes in both trees is in the range [0, 2000].

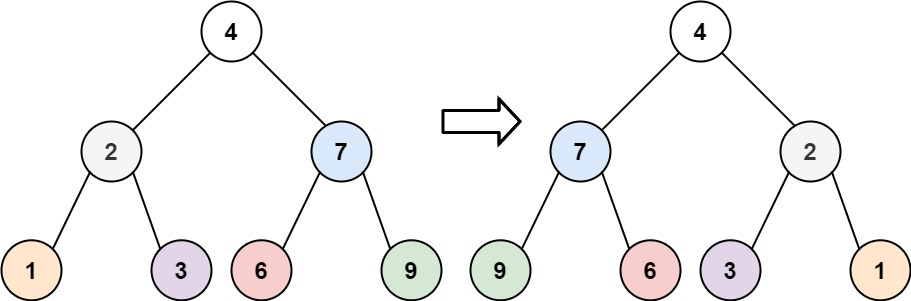
-104 <= Node.val <= 104

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem Statement 5:**

Given the root of a binary tree, invert the tree, and return its root.

Example 1:



Input: root = [4,2,7,1,3,6,9]

Output: [4,7,2,9,6,3,1]

Example 2:

Input: root = [2,1,3]

Output: [2,3,1]

Example 3:

Input: root = []

Output: []

Constraints:

The number of nodes in the tree is in the range [0, 100].

-100 <= Node.val <= 100

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 6:

Given the root of a binary tree, return the in-order traversal of its nodes' values.

Example 1:

Input: root = [1,null,2,3]

Output: [1,3,2]

Example 2:

Input: root = []

Output: []

Example 3:

Input: root = [1]

Output: [1]

Constraints:

The number of nodes in the tree is in the range [0, 100].

-100 <= Node.val <= 100

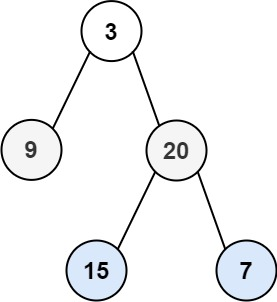
Follow up: Recursive solution is trivial, could you do it iteratively?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem Statement 7:**

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Example 1:



Input: root = [3,9,20,null,null,15,7]

Output: [[3],[9,20],[15,7]]

Example 2:

Input: root = [1]

Output: [[1]]

Example 3:

Input: root = []

Output: []

Constraints:

The number of nodes in the tree is in the range [0, 2000].

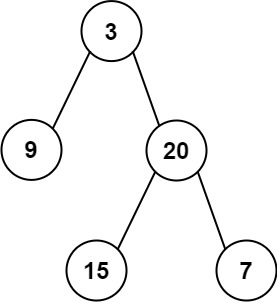
-1000 <= Node.val <= 1000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem Statement 8:**

Given the root of a binary tree, return the average value of the nodes on each level in the form of an array. Answers within 10-5 of the actual answer will be accepted.

Example 1:



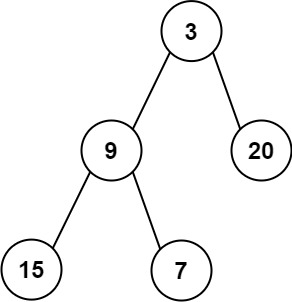
Input: root = [3,9,20,null,null,15,7]

Output: [3.00000,14.50000,11.00000]

Explanation: The average value of nodes on level 0 is 3, on level 1 is 14.5, and on level 2 is 11.

Hence return [3, 14.5, 11].

Example 2:



Input: root = [3,9,20,15,7]

Output: [3.00000,14.50000,11.00000]

Constraints:

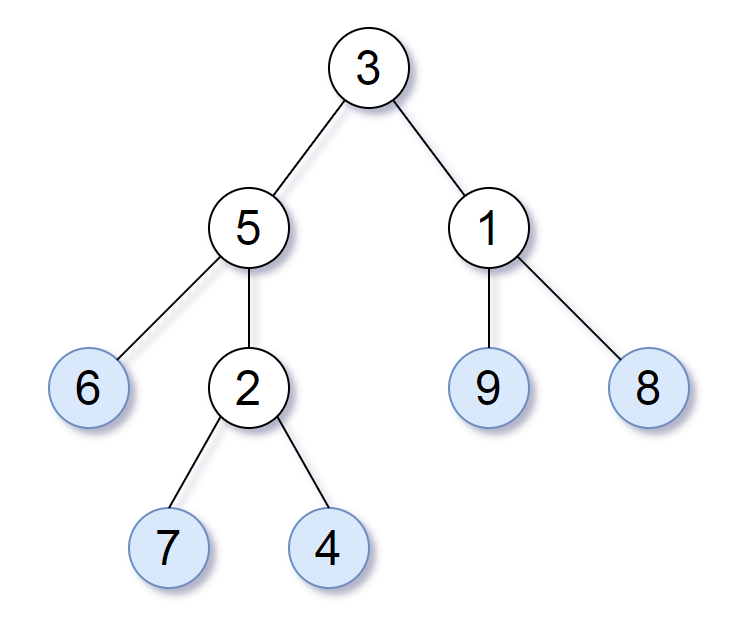
The number of nodes in the tree is in the range [1, 104].

-231 <= Node.val <= 231 - 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 9:

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

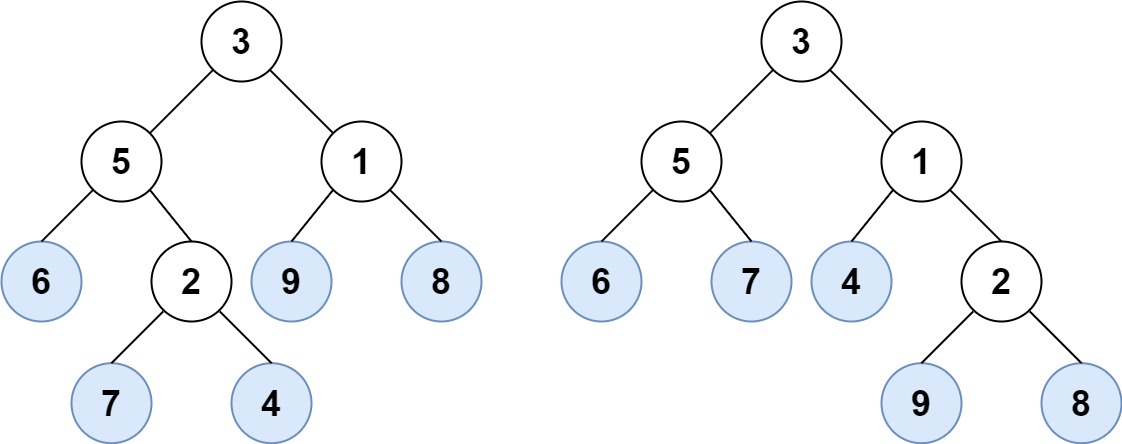


For example, in the given tree above, the leaf value sequence is (6, 7, 4, 9, 8).

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.

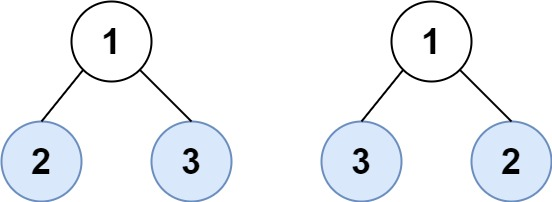
Example 1:



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].

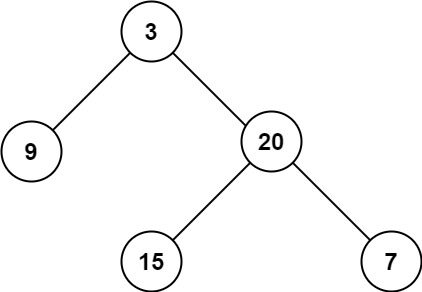
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 10:

Given the root of a binary tree, return its maximum depth.

A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:



Input: root = [3,9,20,null,null,15,7]

Output: 3

Example 2:

Input: root = [1,null,2]

Output: 2

Constraints:

The number of nodes in the tree is in the range [0, 104].

-100 <= Node.val <= 100

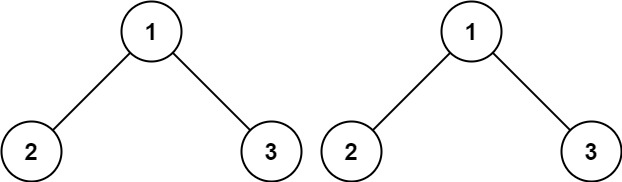
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 11:

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

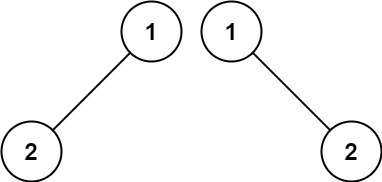
Example 1:



Input: p = [1,2,3], q = [1,2,3]

Output: true

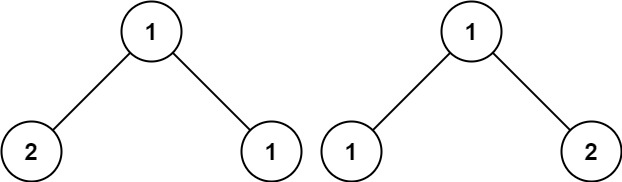
Example 2:



Input: p = [1,2], q = [1,null,2]

Output: false

Example 3:



Input: p = [1,2,1], q = [1,1,2]

Output: false

Constraints:

The number of nodes in both trees is in the range [0, 100].

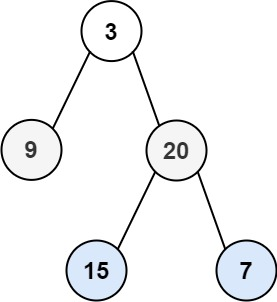
-104 <= Node.val <= 104

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 12:

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

Example 1:



Input: root = [3,9,20,null,null,15,7]

Output: [[3],[20,9],[15,7]]

Example 2:

Input: root = [1]

Output: [[1]]

Example 3:

Input: root = []

Output: []

Constraints:

The number of nodes in the tree is in the range [0, 2000].

-100 <= Node.val <= 100

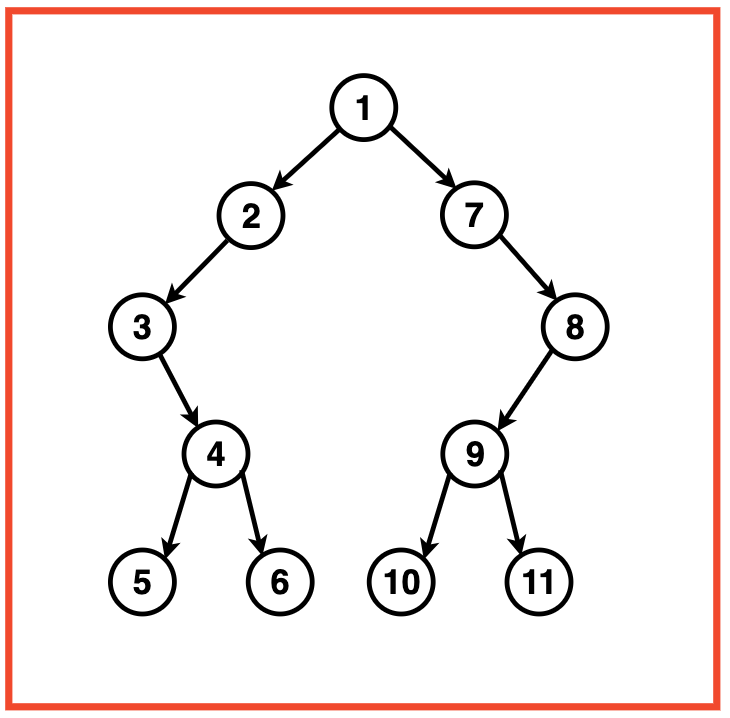
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 13:

Given a Binary Tree, perform the boundary traversal of the tree. The boundary traversal is the process of visiting the boundary nodes of the binary tree in the anticlockwise direction, starting from the root.

Example 1:

Input: 1 2 7 3 -1 -1 8 -1 4 9 -1 5 6 10 11



Output: [1, 2, 3, 4, 5, 6, 10, 11, 9, 8, 7]

Explanation: The boundary traversal of a binary tree involves visiting its boundary nodes in an anticlockwise direction:

Starting from the root, we traverse from: 1

The left side traversal includes the nodes: 2, 3, 4

The bottom traversal include the leaf nodes: 5, 6, 10, 11

The right side traversal includes the nodes: 9, 8, 7

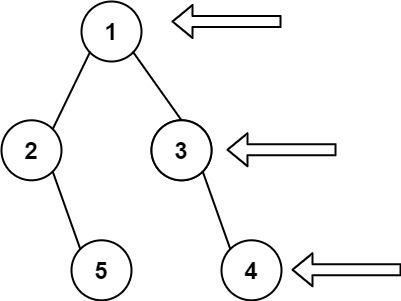
We return to the root and the boundary traversal is complete.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 14:

Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Example 1:



Input: root = [1,2,3,null,5,null,4]

Output: [1,3,4]

Example 2:

Input: root = [1,null,3]

Output: [1,3]

Example 3:

Input: root = []

Output: []

Constraints:

The number of nodes in the tree is in the range [0, 100].

-100 <= Node.val <= 100

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Problem Statement 15:

You are given the root of a binary tree containing digits from 0 to 9 only.

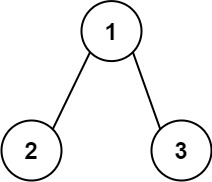
Each root-to-leaf path in the tree represents a number.

For example, the root-to-leaf path 1 -> 2 -> 3 represents the number 123.

Return the total sum of all root-to-leaf numbers. Test cases are generated so that the answer will fit in a 32-bit integer.

A leaf node is a node with no children.

Example 1:



Input: root = [1,2,3]

Output: 25

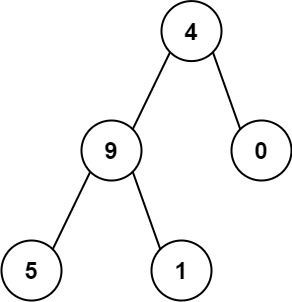
Explanation:

The root-to-leaf path 1->2 represents the number 12.

The root-to-leaf path 1->3 represents the number 13.

Therefore, sum = 12 + 13 = 25.

Example 2:



Input: root = [4,9,0,5,1]

Output: 1026

Explanation:

The root-to-leaf path 4->9->5 represents the number 495.

The root-to-leaf path 4->9->1 represents the number 491.

The root-to-leaf path 4->0 represents the number 40.

Therefore, sum = 495 + 491 + 40 = 1026.

Constraints:

The number of nodes in the tree is in the range [1, 1000].

0 <= Node.val <= 9

The depth of the tree will not exceed 10.