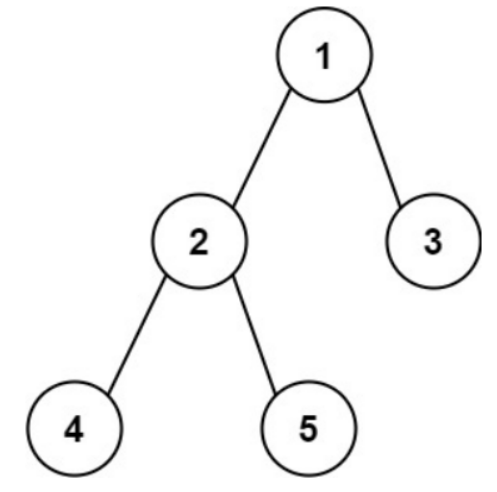
<https://leetcode.com/problems/diameter-of-binary-tree/>

Given the root of a binary tree, return *the length of the* ***diameter*** *of the tree*.

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The **length** of a path between two nodes is represented by the number of edges between them.

**Example 1:**



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

Output: 3

Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

**Example 2:**

Input: root = [1,2]

Output: 1

**Constraints:**

* The number of nodes in the tree is in the range [1, 104].
* -100 <= Node.val <= 100

**Attempt 1: 2023-01-12**

**Solution 1:   (30 min)**

/\*\*

\* 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 {

int maxDiameter = 0;

public int diameterOfBinaryTree(TreeNode root) {

if(root == null) {

return 0;

}

helper(root);

return maxDiameter;

}

private int helper(TreeNode root) {

if(root == null) {

return 0;

}

int left = helper(root.left);

int right = helper(root.right);

// The length of path between two nodes is represented by

// the number of edges between them, so no need 'left + right + 1'

maxDiameter = Math.max(maxDiameter, left + right);

return Math.max(left, right) + 1;

}

}

Time Complexity : O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/diameter-of-binary-tree/solutions/101132/java-solution-maxdepth/>

For every node, length of longest path which pass it = MaxDepth of its left subtree + MaxDepth of its right subtree.

public class Solution {

int max = 0;

public int diameterOfBinaryTree(TreeNode root) {

maxDepth(root);

return max;

}

private int maxDepth(TreeNode root) {

if (root == null) return 0;

int left = maxDepth(root.left);

int right = maxDepth(root.right);

max = Math.max(max, left + right);

return Math.max(left, right) + 1;

}

}