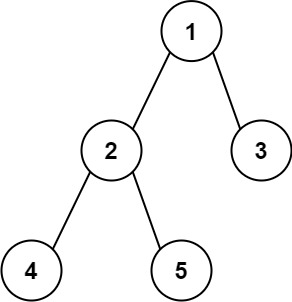
Given the root of a binary tree, return *the length of the****diameter****of the tree*.

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

Solution:

/\*\*

\* 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 {

public int diameterOfBinaryTree(TreeNode root) {

// base case if tree is empty

if (root == null)

return 0;

return diameter(root)-1;

}

public int diameter(TreeNode root) {

if (root == null)

return 0;

// get the height of left and right sub-trees

int lheight = height(root.left);

int rheight = height(root.right);

// get the diameter of left and right sub-trees

int ldiameter = diameter(root.left);

int rdiameter = diameter(root.right);

/\* Return max of following three

1) Diameter of left subtree

2) Diameter of right subtree

3) Height of left subtree + height of right subtree + 1

\*/

return Math.max(lheight + rheight + 1,

Math.max(ldiameter, rdiameter));

}

static int height(TreeNode node)

{

// base case tree is empty

if (node == null)

return 0;

// If tree is not empty then height = 1 + max of

// left height and right heights

return (1

+ Math.max(height(node.left),

height(node.right)));

}

}