**Diameter of Binary Tree**

Question

Given a binary tree, you need to compute 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.

**Example:**  
Given a binary tree

1

/ \

2 3

/ \

4 5

Return **3**, which is the length of the path [4,2,1,3] or [5,2,1,3].

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

#### **Solution Approach #1: Depth-First Search [Accepted]**

**Intuition**

Any path can be written as two arrows (in different directions) from some node, where an arrow is a path that starts at some node and only travels down to child nodes.

If we knew the maximum length arrows L, R for each child, then the best path touches L + R + 1 nodes.

**Algorithm**

Let's calculate the depth of a node in the usual way: max(depth of node.left, depth of node.right) + 1. While we do, a path "through" this node uses 1 + (depth of node.left) + (depth of node.right) nodes. Let's search each node and remember the highest number of nodes used in some path. The desired length is 1 minus this number.

#### Coding Solution

Java

|  |
| --- |
| class Solution {  int ans;  public int diameterOfBinaryTree(TreeNode root) {  ans = 1;  depth(root);  return ans - 1;  }  public int depth(TreeNode node) {  if (node == null) return 0;  int L = depth(node.left);  int R = depth(node.right);  ans = Math.max(ans, L+R+1);  return Math.max(L, R) + 1;  }  } |

Python

|  |
| --- |
| class Solution(object):  def diameterOfBinaryTree(self, root):  self.ans = 1  def depth(node):  if not node: return 0  L = depth(node.left)  R = depth(node.right)  self.ans = max(self.ans, L+R+1)  return max(L, R) + 1  depth(root)  return self.ans - 1 |

**Complexity Analysis**

* Time Complexity: O(N)*O*(*N*). We visit every node once.
* Space Complexity: O(N)*O*(*N*), the size of our implicit call stack during our depth-first search.