543. Diameter of Binary Tree

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.

Java Python Copy

```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;
    }
}
```

Complexity Analysis

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

Analysis written by: @awice (https://leetcode.com/awice).