

543. Diameter of Binary Tree

1007

58

[Description \(/problems/diameter-of-binary-tree/description/\)](/problems/diameter-of-binary-tree/description/)[Hints \(/problems/diameter-of-binary-tree/hints/\)](/problems/diameter-of-binary-tree/hints/)[Submissions \(/problems/diameter-of-binary-tree/submissions/\)](/problems/diameter-of-binary-tree/submissions/)

Quick Navigation ▾

[View in Article \(/articles/diameter-of-binary-tree/\)](/articles/diameter-of-binary-tree/)

Notes

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: $\max(\text{depth of node.left}, \text{depth of node.right}) + 1$. While we do, a path "through" this node uses $1 + (\text{depth of node.left}) + (\text{depth of node.right})$ 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

```
1 class Solution {
2     int ans;
3     public int diameterOfBinaryTree(TreeNode root) {
4         ans = 1;
5         depth(root);
6         return ans - 1;
7     }
8     public int depth(TreeNode node) {
9         if (node == null) return 0;
10        int L = depth(node.left);
11        int R = depth(node.right);
12        ans = Math.max(ans, L+R+1);
13        return Math.max(L, R) + 1;
14    }
15 }
```

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>).

Comments: 15

Sort By ▾



Type comment here... (Markdown is supported)

Preview

Post