AlgoExpert | Python | 00:00:00

Prompt | Scratchpad | Solutions | Video Explanation

**Difficulty:** Easy **Category:** Binary Trees **Successful Submissions:** 55,756+

# Node Depths

The distance between a node in a Binary Tree and the tree's root is called the node's depth.

Write a function that takes in a Binary Tree and returns the sum of its nodes' depths.

Each `BinaryTree` node has an integer `value`, a `left` child node, and a `right` child node. Children nodes can either be `BinaryTree` nodes themselves or `None` / `null`.

### Sample Input

```
tree =    1
        /     \
       2       3
     /   \   /   \
    4     5 6     7
  /   \
 8     9
```

### Sample Output

```
16
// The depth of the node with value 2 is 1.
// The depth of the node with value 3 is 1.
// The depth of the node with value 4 is 2.
// The depth of the node with value 5 is 2.
// Etc..
// Summing all of these depths yields 16.
```

## Hints

**Hint 1**

As obvious as it may seem, to solve this question, you'll have to figure out how to compute the depth of any given node; once you know how to do that, you can compute all of the depths and add them up to obtain the desired output.

**Hint 2**

To compute the depth of a given node, you need information about its position in the tree. Can you pass this information down from the node's parent?

**Hint 3**

The depth of any node in the tree is equal to the depth of its parent node plus 1. By starting at the root node whose depth is 0, you can pass down to every node in the tree its respective depth, and you can implement the algorithm that does this and that sums up all of the depths either recursively or iteratively.

**Optimal Space & Time Complexity**

Average case: when the tree is balanced O(n) time | O(h) space - where n is the number of nodes in the Binary Tree and h is the height of the Binary Tree

Your Solutions | Run Code

Solution 1 | Solution 2 | Solution 3

```python
'''Recursive approach'''
def nodeDepths(root, depth=0):
    if root is None: return 0

    return depth + nodeDepths(root.left, depth+1) + nodeDepths(root.right, depth+1)
    #return node_depths_iterative(root)

'''Iterative approach'''
def node_depths_iterative(root):
    sum_dep = 0
    stack = [{'node': root, 'depth': 0}]
    while len(stack):
        node_info = stack.pop()
        node, depth = node_info['node'], node_info['depth']
        if node is None: continue

        sum_dep += depth
        stack.append({'node': node.left, 'depth': depth+1})
        stack.append({'node': node.right, 'depth': depth+1})
    return sum_dep;


class BinaryTree:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


# Kunal Wadhwa

# https://github.com/kunal5042
# https://leetcode.com/kunal5042/
# https://www.hackerrank.com/kunalwadhwa_cs
# https://www.linkedin.com/in/kunal5042/

```

Tests | Quick Test | Sandbox

**Test Case 1**

```
{
  "tree": {
    "nodes": [
      {"id": "1", "left": "2", "right": "3", "value
      {"id": "2", "left": "4", "right": "5", "value
      {"id": "3", "left": "6", "right": "7", "value
      {"id": "4", "left": "8", "right": "9", "value
      {"id": "5", "left": null, "right": null, "val
      {"id": "6", "left": null, "right": null, "val
      {"id": "7", "left": null, "right": null, "val
      {"id": "8", "left": null, "right": null, "val
      {"id": "9", "left": null, "right": null, "val
    ],
    "root": "1"
  }
}
```

**Test Case 2**

```
{
  "tree": {
    "nodes": [
      {"id": "1", "left": null, "right": null, "val
    ],
    "root": "1"
  }
}
```

**Test Case 3**

```
{
```

Custom Output | Raw Output | Submit Code

**Yay, your code passed all the test cases!**

9 / 9 test cases passed.

- Test Case 1 passed!
- Test Case 2 passed!
- Test Case 3 passed!
- Test Case 4 passed!
- Test Case 5 passed!
- Test Case 6 passed!
- Test Case 7 passed!
- Test Case 8 passed!