Diameter of a Binary Tree

May 28, 2020

0.1 Problem statement

Given the root of a binary tree, find the diameter.

Note: Diameter of a Binary Tree is the maximum distance between any two nodes

```
In [1]: class BinaryTreeNode:

            def __init__(self, data):
                self.left = None
                self.right = None
                self.data = data
```

```
In [1]: def diameter_of_binary_tree(root):
            """
            :param: root - Root of binary tree
            TODO: Complete this method and return diameter (int) of binary tree
            """
            pass
```

You can use the following function to test your code with custom test cases. The function `convert_arr_to_binary_tree` takes an array input representing level-order traversal of the binary tree.

The above tree would be represented as `arr = [1, 2, 3, 4, None, 5, None, None, None, None, None]`

Notice that the level order traversal of the above tree would be `[1, 2, 3, 4, 5]`.

Note the following points about this tree: * `None` represents the lack of a node. For example, `2` only has a left node; therefore, the next node after `4` (in level order) is represented as `None` * Similarly, `3` only has a left node; hence, the next node after `5` (in level order) is represted as `None`. * Also, `4` and `5` don't have any children. Therefore, the spots for their children in level order are represented by four `None` values (for each child of `4` and `5`).

```
In [86]: from queue import Queue

         def convert_arr_to_binary_tree(arr):
             """
             Takes arr representing level-order traversal of Binary Tree
             """
             index = 0
```

```
    length = len(arr)

    if length <= 0 or arr[0] == -1:
        return None

    root = BinaryTreeNode(arr[index])
    index += 1
    queue = Queue()
    queue.put(root)

    while not queue.empty():
        current_node = queue.get()
        left_child = arr[index]
        index += 1

        if left_child is not None:
            left_node = BinaryTreeNode(left_child)
            current_node.left = left_node
            queue.put(left_node)

        right_child = arr[index]
        index += 1

        if right_child is not None:
            right_node = BinaryTreeNode(right_child)
            current_node.right = right_node
            queue.put(right_node)
    return root
```

```
In [87]: # Solution
         def diameter_of_binary_tree(root):
             return diameter_of_binary_tree_func(root)[1]

         def diameter_of_binary_tree_func(root):
             """
             Diameter for a particular BinaryTree Node will be:
                 1. Either diameter of left subtree
                 2. Or diameter of a right subtree
                 3. Sum of left-height and right-height
             :param root:
             :return: [height, diameter]
             """
             if root is None:
                 return 0, 0
```

```
        left_height, left_diameter = diameter_of_binary_tree_func(root.left)
        right_height, right_diameter = diameter_of_binary_tree_func(root.right)

        current_height = max(left_height, right_height) + 1
        height_diameter = left_height + right_height
        current_diameter = max(left_diameter, right_diameter, height_diameter)

        return current_height, current_diameter
```

In [88]:
```
def test_function(test_case):
    arr = test_case[0]
    solution = test_case[1]
    root = convert_arr_to_binary_tree(arr)
    output = diameter_of_binary_tree(root)
    print(output)
    if output == solution:
        print("Pass")
    else:
        print("Fail")
```

In [89]:
```
arr = [1, 2, 3, 4, 5, None, None, None, None, None, None]
solution = 3

test_case = [arr, solution]
test_function(test_case)
```

```
3
Pass
```

In [90]:
```
arr = [1, 2, 3, 4, None, 5, None, None, None, None, None]
solution = 4

test_case = [arr, solution]
test_function(test_case)
```

```
4
Pass
```

In [91]:
```
arr = [1, 2, 3, None, None, 4, 5, 6, None, 7, 8, 9, 10, None, None, None, None, None, N
solution = 6

test_case = [arr, solution]
test_function(test_case)
```

```
6
Pass
```

In [ ]: