Path-from-root-to-node

May 4, 2020

0.1 Problem Statement

Given the root of a Binary Tree and a `data` value representing a node, return the path from the root to that node in the form of a list. You can assume that the binary tree has nodes with unique values.

```
In [38]: class BinaryTreeNode:

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

```
In [1]: def path_from_root_to_node(root, data):
            """
            :param: root - root of binary tree
            :param: data - value (representing a node)
            TODO: complete this method and return a list containing values of each node in the p
            from root to the data node
            """
            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` represnts lack of 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 [40]: 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 [1]: # Solution
        def path_from_root_to_node(root, data):
            """
            Assuming data as input to find the node
            The solution can be easily changed to find a node instead of data
            :param data:
            :return:
            """
            output = path_from_node_to_root(root, data)
            return list(reversed(output))

        def path_from_node_to_root(root, data):
            if root is None:
                return None
```

```
    elif root.data == data:
        return [data]

    left_answer = path_from_node_to_root(root.left, data)
    if left_answer is not None:
        left_answer.append(root.data)
        return left_answer

    right_answer = path_from_node_to_root(root.right, data)
    if right_answer is not None:
        right_answer.append(root.data)
        return right_answer
    return None
```

```
In [46]: def test_function(test_case):
             arr = test_case[0]
             data = test_case[1]
             solution = test_case[2]
             root = convert_arr_to_binary_tree(arr)
             output = path_from_root_to_node(root, data)
             if output == solution:
                 print("Pass")
             else:
                 print("Fail")
```

```
In [47]: arr = [1, 2, 3, 4, 5, None, None, None, None, None, None]
         data = 5
         solution = [1, 2, 5]

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

```
Pass
```

```
In [48]: arr = [1, 2, 3, 4, None, 5, None, None, None, None, None]
         data = 5
         solution = [1, 3, 5]

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

```
Pass
```

```
In [49]: arr = [1, 2, 3, None, None, 4, 5, 6, None, 7, 8, 9, 10, None, None, None, None, None, N
         data = 11
         solution = [1, 3, 4, 6, 10, 11]
         test_case = [arr, data, solution]
         test_function(test_case)

Pass


In [50]: arr = [1, 2, 3, None, None, 4, 5, 6, None, 7, 8, 9, 10, None, None, None, None, None, N
         data = 8
         solution = [1, 3, 5,8]

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

Pass
```