

Assignment 2 – Algorithm

Defining problem:

From a binary tree, if there is any duplicate in the root, it will return the duplicate value that is closest to the root. If there is no duplicate value, it will return -1.

Code provided:

```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def is_symmetric(root: TreeNode) -> int:
    if not root:
        return -1

    def dfs(node, seen):
        if not node:
            return -1

        if node.val in seen:
            return node.val
        seen.add(node.val)

        left_result = dfs(node.left, seen)
        right_result = dfs(node.right, seen)

        if left_result != -1:
            return left_result
        if right_result != -1:
            return right_result

        return -1

    return dfs(root, set())
```

Create example that demonstrates you understand the problem:


Example 2 below shows that the code does not provide the duplicate value that is closest to the root because it should give answer of 1 instead of 2

```
[47] # ihayun Example 1:
      root = TreeNode(1)
      root.left = TreeNode(2)
      root.left.left = TreeNode(5)
      root.left.left.left = TreeNode(5)

      root.right = TreeNode(8)
      root.right.right = TreeNode(8)

      result = is_symmetric(root)
      if result != -1:
          print("The closest duplicate value to the root is:", result)
      else:
          print("No duplicate values found.")
```

The closest duplicate value to the root is: 5

```
 # ihayun Example 2:
      root = TreeNode(1)
      root.left = TreeNode(2)
      root.left.left = TreeNode(2)

      root.right = TreeNode(1)

      result = is_symmetric(root)
      if result != -1:
          print("The closest duplicate value to the root is:", result)
      else:
          print("No duplicate values found.")
```

The closest duplicate value to the root is: 2

Trace/Walkthrough 1 example that your partner made and explain it:

From the New Example 1 below, the left of root 1 would be 2 and it has left node 3 and right node 4. The right of root 1 would be 2 and it has left node of 4 and right node of 3. When it goes to the left side, it doesn't find any duplicate value (at this point, the seen consists of 1, 2, 3, 4), and when it goes to the right, it finds the first duplicate value, which is 2.

```
# New Example 1:
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(2)
root.left.left = TreeNode(3)
root.left.right = TreeNode(4)
root.right.left = TreeNode(4)
root.right.right = TreeNode(3)

result = is_symmetric(root)
if result != -1:
    print("The closest duplicate value to the root is:", result)
else:
    print("No duplicate values found.")

    The closest duplicate value to the root is: 2
```

Solution Explanation:

The solution that was provided is first, it will return -1 if there is no root. Then it will go into dfs function, in which first, it will return -1 if there is no node. If there is node, it will store the value of the node into seen.

To check the node value, it will first go down the left side of the root, then store the value of the node on the left side, and return the value if the same value is in the seen, otherwise, it will store the value in the seen. Once the left side has no more node, it will return -1 for left result, and it will go to the right node and also store the value of the node on the right side, and return the value if the same value is in the seen; otherwise, it will also return -1 if no more node. If both left result and right result have no more node and no duplicate value, it will return -1 which will show as "No duplicate values found".

Complexity:

The time complexity seems to be $O(n)$ since it only goes through each node once and the space complexity seems to depend on the height of the tree which affects the space stored in the seen, so might be $O(h)$.

Alternative Solution:

The code should reflect BFS instead of DFS. The code does not give the duplicate value of the closest distance because it checks all the values of the left of the root first then go to the side of the root once finished. It should instead check both the right and left side of the same height or level first and return the value of the duplicate.

Part 3:

There was a lot of research when doing the first part of assignment as I will need to understand the question, write the solution, test the code, and figure out the most efficient algorithm for a given question. Learning how to find good sources and apply it is definitely a must-have skill, and it is a skill needed for assignment 1. It took a lot of time to research and learning from the sources, but this is something that will need to be continuously worked and improved on.

For the second part of assignment, reviewing a code is definitely challenging. However, I definitely learn a lot about trouble shooting a code. It is actually not easy to understand and test a code that is written by another person. After reviewing and testing the code, the time and space complexity will also need to be optimized as much as possible. After doing both the assignments, it provides more clear understanding on how to approach technical question and providing feedback.