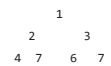
PART 2

1. Paraphrase the problem in your own words:

Look for duplicates in a binary tree, return the duplicate value closest to the tree's root, or return -1 if there are no duplicates.

2. Create 1 new example that demonstrates you understand the problem. Trace/walk through 1 example that your partner made and explain it:



Input: [4, 2, 7, 1, 6, 3, 7] *using DFS inorder traversal

Output: 7

Input: [1, 2, 2, 3, 4, 4, 3]

Output: 2

Explanation: This is a Breadth-First Search (BFS) which means the solution involves traversing all nodes at the same level before traversing the next level.

`[1]` is the root and duplicate values `[2, 2]` are the closest to the root.

3. Copy the solution your partner wrote:

```
# 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())

# Example 1:
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(2)
root.left.left = TreeNode(3)
root.left.right = TreeNode(7)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)

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
```

4. Explain why their solution works in your own words:

The code executes DFS inorder traversal and keeps track of values of visited nodes in the `seen` set.

It then returns the first duplicate value encountered, which would also be the duplicate value closest to the root.

`-1` is returned if no duplicates exist.

5. Explain the problem's time and space complexity in your own words:

Time complexity: `O(n)` Since each node is visited at least once.

Space complexity: `O(h)` Equal to the height of the tree.

6. Critique your partner's solution, including explanation, if there is anything that should be adjusted:

Analyzing the code to understand both the traversal method and logic/rationale behind the solution was time-consuming. Detailed comments would have been helpful.

PART 3

Reflection:

My studying process for Assignment 1 involved reviewing material on array-based data structures as well as recursion.

I had some issues with an index error in my code, but ChatGPT proposed a viable solution to that error.

Assessing my partner's binary tree problem was more time-consuming than expected.

I reviewed the appropriate class material before attempting to tackle the questions, but it still took effort to make sense of the code and determine exactly how DFS inorder traversal was being implemented.

Detailed comments would have made the process faster and easier.

Overall, the challenges deepened my understanding of both concepts.