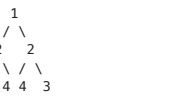
Create 1 new example that demonstrates you understand the problem:

Suppose we have the following binary tree:

In this tree, the closest duplicate value to the root is 2.

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

Your partner provided an example with the following binary tree:



Here, the tree is symmetric, so the function should return -1 indicating that no duplicate values were found.

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())
```

Explain why their solution works in your own words:

I believe the solution works by applying a depth-first search (DFS) traversal of the binary tree. It recursively traverses each node of the tree, keeping track of the values encountered so far. If a duplicate value is found during traversal, the function returns that value. Otherwise, it returns -1 indicating no duplicate values were found. The DFS approach ensures that all nodes of the tree are visited, allowing for the detection of duplicate values.

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

The time it takes to run the solution is directly proportional to the number of nodes in the binary tree. This is because the solution looks at each node once. Similarly, the amount of memory the solution uses is also directly proportional to the number of nodes in the tree. This is because the solution keeps track of visited nodes using a set, and it might need to remember all nodes if the tree is very deep. So, if there are n nodes in the tree, the time and memory needed are both about n, which we write as O(n).

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

The solution tries to solve the problem using a depth-first search (DFS) method, but it's not very clear and has some mistakes. Also, the function name "is_symmetric" is confusing because it suggests it's checking for symmetry, which isn't the problem. It's actually about finding duplicate values. Using a set to remember visited values is a good idea, but the part where it returns the closest duplicate value needs more explanation. In short, the solution has potential but needs to be clearer and more accurate.

Reflection:

Overall the solution is good and well written in my opinion, but adding some comments would make it muhc better. The solution tries to find matching numbers in a tree but it was a bit confusing for me about what the problem is asking. While the examples are right, the solution and explanations are a bit mixed up in my opinion. Also, the explanation of how the solution works is a bit tricky to understand, and the parts about how long it takes and how much space it needs could be clearer. It's like trying to explain how a game works but not being very clear about the rules. With some refinement and clarification, the solution could become even more effective.