**Assignment 2:**

Based on the document's contents, let's break down the assignment into detailed explanations and a sample Python code to demonstrate the solution provided by Ashish Kadia. The task involves finding duplicates in a binary tree and returning the closest duplicate to the root or -1 if no duplicates exist.

1. **Paraphrase the Problem:**

The problem is to inspect a given binary tree to identify if any value occurs more than once. If so, return the duplicate value that is closest to the root node. In case there are multiple duplicates with the same proximity to the root, any one of them can be returned. If the tree contains no duplicates, the function should return -1.

1. **Create a New Example:**

Let's create a new example to ensure understanding:

5

/ \

3 5

/ \ \

2 4 6

/ \ \ 2 4 6

In this binary tree, the value **5** appears more than once, and since one of the **5**s is the root, it's the closest duplicate to itself.

1. **Trace/Walkthrough an Example:**

Let's walk through Example 5 from the document:

2

/ \

4 7

/ \ / \

1. 8 9 9

This tree has a duplicate value of **9**. Both duplicates of **9** are at the same level and equidistant from the root. According to the problem statement, returning any one of these **9**s would be correct. The output is **9**.

1. **The Solution:**

Here's the solution provided in the document:

from collections import deque

class TreeNode:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def list\_to\_tree(nodes):

root = TreeNode(nodes[0])

queue = deque([root])

i = 1

while queue and i < len(nodes):

current = queue.popleft()

if nodes[i] is not None:

current.left = TreeNode(nodes[i])

queue.append(current.left)

i += 1

if i < len(nodes) and nodes[i] is not None:

current.right = TreeNode(nodes[i])

queue.append(current.right)

i += 1

return root

def find\_closest\_duplicate(root):

queue = deque([(root, 0)])

duplicates = {}

while queue:

node, distance = queue.popleft()

if node.val in duplicates:

return node.val

duplicates[node.val] = distance

if node.left:

queue.append((node.left, distance + 1))

if node.right:

queue.append((node.right, distance + 1))

return -1

1. **Explain Why the Solution Works:**

The solution effectively employs a breadth-first search (BFS) approach, using a queue to traverse the tree level by level. This ensures that the first duplicate encountered is the one closest to the root since levels are explored from the top down. The **duplicates** dictionary tracks visited values and their distances from the root, enabling efficient duplicate detection. Upon finding a duplicate, the function immediately returns its value, ensuring minimal traversal.

1. **Explain the Problem’s Time and Space Complexity:**

The time complexity is O(n), where n is the number of nodes in the tree. This is because each node is visited exactly once during the BFS traversal. The space complexity is also O(n), primarily due to the queue used in BFS which, in the worst-case scenario, could contain all nodes at the tree's last level. This analysis assumes a complete or nearly complete binary tree.

1. **Critique the Solution:**

The solution is efficient and correctly implements the BFS approach to solve the problem. However, improvements can be made in terms of code readability and efficiency:

* **Error Handling**: The code assumes the tree has at least one node. It might be beneficial to add a check if the **nodes** list is empty to avoid potential errors.
* **Early Termination**: The function continues traversal even after finding a duplicate. An optimization could be implemented to stop the search once a duplicate is found, although the current logic ensures the closest duplicate is returned.
* **Code Readability**: Adding comments or more descriptive variable names could improve understanding, especially for someone unfamiliar with BFS.