```
(hash('Ashish')%3) + 1
```

```
    1
```

Question One: Check Duplicates in Tree Given the root of a binary tree, check whether it is contains a duplicate value. If a duplicate exists, return the duplicate value. If there are multiple duplicates, return the one with the closest distance to the root. If no duplicate exists, return -1.

Q1: Paraphrase the problem in your own words.

Answer: For given binary tree structure, we are writing a code which can... (1) Identify and populate duplicate entry. (2) If there are more than one duplicate entries, only populate entry which is closest to the root. (3) if there are no duplicates in data structure populate '-1' as output.

Q2: In the .md file containing your problem, there are examples that illustrate how the code should work. Create 2 new examples that demonstrate you understand the problem.

Aswer:

Example 4:

```
          (1)
     (3)       (6)
(7)   (5) (10) (9)
```

For above binary data tree, (1) is root entry --> which has two child (3) on left and (6) on right. --> (3) have two child, (7) on left and (5) on right. --> (6) have two child (10) on left and (9) on right.

Data structure does not have any duplicates so output will be -1

input: root = [1,3,7,5,6,10,9] output: -1

Example 5:

```
          (2)
     (4)       (7)
(1)   (8) (9) (9)
```

For above binary data tree, (2) is root entry --> which has two child (4) on left and (7) on right. --> (4) have two child, (1) on left and (8) on right. --> (7) have two child (9) on left and (9) on right.

Data structure have '9' as duplicate entry. so output will be: 9

input: root = [2,4,1,8,7,9,9] output: 9

Q3: Code the solution to your assigned problem in Python (code chunk). Try to find the best time and space complexity solution!

Answer: The approch i have chosen is Breath-first search. where the code is going through every node at each level and is storing them in queue. and than this queue is used to find duplicates and the distance from root.

```
# importing necessary library
from collections import deque

# creating a template for each node which will have two child (left and right)
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

# defining a new function which can convert a list of input to a binary tree structure
def list_to_tree(nodes):

    # creating and storing first node as root
    root = TreeNode(nodes[0])
    # initiates level order traversal
    queue = deque([root])
    i = 1

    # creating a while loop which will keep stroing left and right child
    while queue and i < len(nodes):
        current = queue.popleft()  # Pop the leftmost node from the queue

        # creating left child if the value is not None
        if nodes[i] is not None:
            current.left = TreeNode(nodes[i])
            queue.append(current.left)

        i += 1

        # Creating right child if the value is not None
        if i < len(nodes) and nodes[i] is not None:
            current.right = TreeNode(nodes[i])
            queue.append(current.right)
```

```
        i += 1

    return root

# finding duplicates and storing them in dictionary
def find_closest_duplicate(root):

    queue = deque([(root, 0)])
    duplicates = {}

    while queue:
        # Pop the leftmost node from the queue
        node, distance = queue.popleft()
        # check for duplicate
        if node.val in duplicates:
            return node.val

        # update dictionary with current node's value and distance
        duplicates[node.val] = distance

        # add left child to the queue with an increased distance
        if node.left:
            queue.append((node.left, distance + 1))
        # add right child to the queue with an increased distance
        if node.right:
            queue.append((node.right, distance + 1))

    # returning -1 if no duplicates found
    return -1
```

```
def is_symmetric(root: TreeNode) -> int:
    return find_closest_duplicate(root)

# Demo usage for example 4:
# Convert the list to a binary tree and storing a root4
root4 = list_to_tree([1,3,7,5,6,10,9])
# expected outcome is -1
print(is_symmetric(root4))

# Demo usage for example 5:
# Convert the list to a binary tree and storing a root5
root5 = list_to_tree([2,4,1,8,7,9,9])
# expected outcome is 9
print(is_symmetric(root5))
```

```
-1
9
```

Q4: Explain why your solution works

Answer: I have used comments to explain code

Q5: Explain the problem’s time and space complexity.

Answer:

both space and time complexity would be O(n)

Time complexity: For each function in my code, while loop goes through each node once. hence time complexity is O(n)

Space complexity: For BFS approch. we care about howmuch nodes are being queued at once. i.e. last level of nodes. assuming that tree is balanced, usually total nodes in last level would be roughly half of total nodes in tree(n/2). so technically space complexity would be O(n/2). this is further simplified to O(n)

Q6: Explain the thinking to an alternative solution (no coding required, but a classmate reading this should be able to code it up based off your text)

Answer:

This problem can also be solved using recursive approch. where both functions: list_to_tree and find_closest_duplicate can be defined as recursive approch. As a recursive list_to_tree function, it can keep calling itself to creat left and right child untill node value is none. (use if condition, no need to run loop). similarly, find_closest_function can be defined