Part 2:

Create a Jupyter Notebook, create 6 of the following headings, and complete the following for your partner's assignment 1:

- Paraphrase the problem in your own words.

A function that checks a binary tree for duplicate values and returns the one closest to the root, or -1 if none exist.

Start from root node Check all nodes for repeating values Return nearest duplicate to root If tied distance, return first found If no duplicates, return -1

In [4]:
```
# Your answer here

# A function that checks a binary tree for duplicate values and returns the

# Start from root node
# Check all nodes for repeating values
# Return nearest duplicate to root
# If tied distance, return first found
# If no duplicates, return -1
```

- Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

In [15]:
```
# Your answer here

root = [4, 2, 6, 1, 2, 5, 7]

# In this tree, we find value 2 twice, at levels 1 and 2. Since level 1 is c
```

- Copy the solution your partner wrote.

In [ ]:
```
# Your answer here

from collections import deque

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

    @staticmethod
    def is_duplicate(root: TreeNode) -> int:
        if not root:
```

```
            return -1  # No nodes in the tree
        # Initialize a queue for BFS and a set to keep track of visited valu
        queue = deque([root])
        seen = set()

        # Perform BFS
        while queue:
            node = queue.popleft()

            # Check if the current node's value is a duplicate
            if node.val in seen:
                return node.val  # Return the first duplicate found

            # Mark the value as seen
            seen.add(node.val)

            # Add child nodes to the queue
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        # If no duplicate is found
        return -1

# given problem
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(2)
root.left.left = TreeNode(3)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)

print(TreeNode.is_duplicate(root))  # Output: 2

# New Example 1
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(3)
root.right.right = TreeNode(6)

print(TreeNode.is_duplicate(root))  # Output: 3

# New Example 2 (No Duplicates):
root = TreeNode(8)
root.left = TreeNode(3)
root.right = TreeNode(10)
root.left.left = TreeNode(1)
root.left.right = TreeNode(6)
root.right.left = TreeNode(9)
root.right.right = TreeNode(14)
```

```
print(TreeNode.is_duplicate(root))  # Output: -1
```

- Explain why their solution works in your own words.

In [10]:
```
# Your answer here

# Solution uses BFS level by level:
# Checks root first, then level 1, then level 2
# Finds closest duplicates first
# Returns first match found at any level

# Uses set for tracking:
# Stores each new value
# Quick duplicate check
# Returns match when found twice

# Queue for BFS:
# Processes nodes in order
# Level by level search

# Handles edge cases:
# Empty tree = -1
# No dupes = -1
# Multiple dupes = closest one
```

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

In [12]:
```
# Your answer here

## Time complexity is O(n):
# Visits each node once with BFS
# Queue = O(1)
# Set ops = O(1)

## Space complexity is O(n)
# Queue stores nodes = O(n)
# Set stores values = O(n)
```

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

In [ ]:
```
# Your answer here

# Good use of BFS for finding closest duplicates
# Clean code with clear variable names
# Handles edge cases well

# Could improve by including more test cases
# Add docstring for the function
```

Part 3:

Please write a 200 word reflection documenting your process from assignment 1, and your presentation and review experience with your partner at the bottom of the Jupyter Notebook under a new heading "Reflection." Again, export this Notebook as pdf.

Reflection

In [ ]:
```
# Your answer here

# This assignment gave me better understanding of code review. Looking at my
# I learned that reviewing code isn't just about finding mistakes. My partne
# Going through their solution helped improve my coding skills. Now I know g
# This was good interview practice too. I understand why clean, efficient co
# This project helped me become a better programmer and gave me real experie
```