Practice Interview

Objective

The partner assignment aims to provide participants with the opportunity to practice coding in an interview context. You will analyze your partner's Assignment 1. Moreover, code reviews are common practice in a software development team. This assignment should give you a taste of the code review process.

Group Size

Each group should have 2 people. You will be assigned a partner

Part 1:

You and your partner must share each other's Assignment 1 submission.

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.

```
# Your answer here
Review of >> https://github.com/nestorojas/algorithms_and_data_structures/blob/assignment-1/02_activities/assignments/assignment_1.ipynb

Given the root node of a binary tree.
The task is to determine if there is any duplicate value in the tree.
If a duplicate value exists, return the value of the duplicate that is closest to the root.
If multiple duplicates exist, return the one with the smallest distance to the root.
If no duplicates are found, return -1.
```

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

```
# Your answer here
    4
   / \
  4   6
 / \   \
7   8   9

Input: root = [4, 4, 6, 7, 8, null, 9]
Output: 4

In this example, the value 4 is duplicated, and it is the closest duplicate to the root.
```

- Copy the solution your partner wrote.

```
# Your answer here
from typing import List, Optional
from collections import deque

# TreeNode definition
class TreeNode(object):
    def __init__(self, val = 0, left = None, right = None):
        self.val = val
        self.left = left
        self.right = right

# Helper function to convert list to binary tree
def list_to_binary_tree(items: List[Optional[int]]) -> Optional[TreeNode]:
    if not items or items[0] is None:
        return None

    iter_items = iter(items)
    root = TreeNode(next(iter_items))
    queue = [root]

    for item in iter_items:
        node = queue.pop(0)
        if item is not None:
            node.left = TreeNode(item)
            queue.append(node.left)

        try:
            item = next(iter_items)
        except StopIteration:
            break

        if item is not None:
            node.right = TreeNode(item)
            queue.append(node.right)

    return root

# Function to check for duplicates in the binary tree
def is_duplicate(root: TreeNode) -> int:
    if not root:
        return -1

    seen = set()
    queue = deque([root])

    while queue:
        current = queue.popleft()
        if current.val in seen:
            return current.val
        seen.add(current.val)
        if current.left:
            queue.append(current.left)
        if current.right:
            queue.append(current.right)

    return -1
```

- Explain why their solution works in your own words.

```
# Your answer here
Why the Solution Works

    -Level-order Traversal:

        -The level-order traversal (BFS) ensures that nodes are checked level by level, starting from the root.
        This guarantees that the first duplicate encountered is the one closest to the root,
        meeting the requirement of finding the duplicate with the smallest distance from the root

    -Use of Set for Duplication Check:

        -The set seen is used for efficient look-up and insertion operations.
        Checking if a value is already in the set and adding a new value to the set both have average time complexity of O(1).

    -Queue for Managing Nodes:

        -The deque is used to manage the nodes during the level-order traversal.
        This allows efficient addition of nodes to the end of the queue
        and removal of nodes from the front, both with O(1) complexity.

        Overall, the combination of these techniques ensures that the function performs efficiently
        and correctly identifies the first duplicate value in the binary tree.
```

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

```
# Your answer here
        Time Complexity:

The time complexity of the is_duplicate function can be analyzed as follows:

    1. Tree Traversal:

        - The function performs a level-order traversal (BFS) of the binary tree, visiting each node exactly once.
        - If there are n nodes in the tree, the traversal requires O(n) time to visit all nodes.

    2. Duplicate Check and Set Operations:

        - For each node, the function checks if the node's value is in the set seen and possibly adds it to the set.
        - Checking for the existence of an element in a set and adding an element to a set both have an average time complexity of O(1).

Since both the traversal and the set operations are efficient, the overall time complexity
is dominated by the traversal, resulting in a time complexity of O(n).

        Space Complexity:

The space complexity of the is_duplicate function is determined by the following factors:

    1. Queue (for BFS traversal):

        - The queue is used to store nodes at each level during the traversal.
        - In the worst case, the queue can hold all the nodes at the maximum level of the tree.
        For a balanced binary tree, the maximum number of nodes at the last level is about n/2.
        - Therefore, the space required for the queue is O(n) in the worst case.

    2. Set (to track seen values):

        - The set seen stores the values of the nodes encountered during the traversal.
        - In the worst case, where all node values are unique, the set will store n values.
        - Thus, the space required for the set is O(n).

Combining these factors, the overall space complexity of the function is O(n), considering both the queue and the set.

        Summary

    - Time Complexity: O(n), where n is the number of nodes in the binary tree.
    This is due to the level-order traversal that visits each node once.
    - Space Complexity: O(n), accounting for the storage required by the queue and the set,
    both of which can grow to store up to n elements in the worst case.

These complexities ensure that the function is efficient in both time and space,
making it suitable for handling reasonably large binary trees.
```

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

```
# Your answer here
Strengths:

    1. Correctness:

        - The solution correctly implements a level-order traversal (BFS) of the binary tree.
        - It accurately identifies and returns the first duplicate value encountered, which is closest to the root.
        - The function handles edge cases such as an empty tree (returning -1).

    2. Efficiency:

        - The use of a set for tracking seen values ensures efficient O(1) average-time complexity for both look-up and insertion operations
        - The queue (implemented using deque) efficiently manages nodes for the level-order traversal.

    3. Clarity:

        - The code is well-structured and easy to understand.
        - Helper functions like list_to_binary_tree are used to convert the input list into a binary tree,
        making the main function more readable and focused on the core logic.

Areas for Improvement:

    1. Edge Cases:

        - The current solution does not explicitly handle cases where
        the input list contains None values (representing missing children).
        The list_to_binary_tree function does handle this correctly,
        but it would be good to ensure the main function explicitly deals with any potential None nodes in its logic.

    2. Documentation and Comments:

        - Adding more comments in the code, especially within the is_duplicate function,
        would improve readability and help others understand the code's flow more easily.

    3. Variable Naming:

        - Using more descriptive variable names could enhance clarity.
        For example, current could be renamed to current_node to explicitly indicate it is a node in the binary tree.

    4. Helper Function Robustness:

        - The list_to_binary_tree function could include more checks or handle more edge cases,
        such as lists with inconsistent lengths that do not fully represent a binary tree.
```

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

```
# Your answer here
The process of solving the problem of identifying duplicates in a binary tree involved several key steps:

    1. Understanding the Problem: I carefully read the problem statement and examples to fully grasp the requirements.
    The goal was to find and return the first duplicate value closest to the root in a binary tree,
    or return -1 if no duplicates exist.

    2. Planning the Solution: I decided to use a level-order traversal (BFS) to ensure we find the closest
    duplicate to the root. A set was used to track seen values, leveraging its O(1) average-time
    complexity for look-up and insertion.
```

Evaluation Criteria

We are looking for the similar points as Assignment 1

- Problem is accurately stated
- New example is correct and easily understandable
- Correctness, time, and space complexity of the coding solution
- Clarity in explaining why the solution works, its time and space complexity
- Quality of critique of your partner's assignment, if necessary

```
    3. Reflection: This process highlighted the importance of clear problem understanding,
```

Submission Information

🚨 **Please review our [Assignment Submission Guide](#)** 🚨 for detailed instructions on how to format, branch, and submit your work. Following these guidelines is crucial for your submissions to be evaluated correctly.

Submission Parameters:

- Submission Due Date: `HH:MM AM/PM - DD/MM/YYYY`
- The branch name for your repo should be: `assignment-2`
- What to submit for this assignment:
  - This Jupyter Notebook (assignment_2.ipynb) should be populated and should be the only change in your pull request.
- What the pull request link should look like for this assignment: `https://github.com/<your_github_username>/algorithms_and_data_structures/pull/<pr_id>`
  - Open a private window in your browser. Copy and paste the link to your pull request into the address bar. Make sure you can see