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.

Partner's assignment: https://github.com/mehulptech/algorithms_and_data_structures/blob/assignment-1/02_activities/assignments/assignment_1.ipynb

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.

Find duplicate values in the binary tree and returns the value that's closest to the root of the tree.

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

New example:

```
        1
      / \
     2   2
    /
   1
```

The above tree has duplicates of 1s and 2s. Since one of the 1 is the root, 1 should be returned as the answer.

My partner's example (guessing of tree structure since only an array was provided):

```
      1
     / \
    2   3
   / \ / \
  4  5 3  6
```

With the above example the only duplicates in the tree is the value 3. Since it is the only duplicate so it should be the returned value.

- Copy the solution your partner wrote.

```python
# 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 values
        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.

The solution does not work because it returns the first duplicate that is found but the first duplicate may not be the closest to the root. Using my example as an example:

```
        1
       / \
      2   2
     /
    1
```

The code will return 2 since that's the first duplicate that is found. The correct answer should be 1 but the duplicate would not be found until the entire tree has been traversed.

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

Time complexity:

Partner's function's time complexity is **O(n)** since it traverse every nodes in the tree once.

Space complexity:

For space complexity, there are two data structures that should be considered.

First is the queue of nodes which at worst case is when we have a full tree and have finished traversing the seond to last level and have queued all the leaf nodes to the queue. In this case we have O(n) since the space required is linear with the input.

The second data structure to be considered is the list of seen values. The worst case is when there are no duplicate values and the complexity is therefore O(n).

Since both are O(n) therefore the space complexity of the algorithm is **O(n)**.

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

The biggest issue is the problem stated above, which is not returning the duplicate that is closest to the root. Beside the issue above and assume the logic as intended, the rest of the code are correct, clearly commented, and does not require any modification.

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

Assignment 1 for me requires design and implementation of question 2 which is to return all root to leaf paths in a binary tree. The first step was to think about how to do it methodically. The most simple way is to walk from the root to each leaf node and writing down each path as we do it. For example, using the first example given, we will find the path between 1-->3, and then 1-->5, 1-->6, and finally 1-->7. Next step was to translate that into detail step-by-step in terms of using a treenode data structure. And finally code it and test it. Finally we go thru the code and determine the time and space complexity and check if there are any optimization that can be done.

For this assignment, the approach I've taken is first determine what a solution would most likely look like, and then compare it to my partner's design and implementation. In order to find the duplicate closest to the root, it is necessary to know all the duplicate values first before determining which is the closest. That means either to walk the tree twice (once to get all the values in the tree, and second time to find the closest to the root), or alternatively to do one walk but needs to remember all the values and their distance from the root. After determining what a solution should look like, first look at my partner's code quickly revealed one major issue with the design which is returning the first found duplicate. However looking past the major issue with the code, the rest of the coding does appear to be correct and reflect the design as shown by the comment.

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

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 your pull request properly. This helps the technical facilitator and learning support staff review your submission easily.

Checklist:

- ☐ Created a branch with the correct naming convention.
- ☐ Ensured that the repository is public.
- ☐ Reviewed the PR description guidelines and adhered to them.
- ☐ Verify that the link is accessible in a private browser window.

If you encounter any difficulties or have questions, please don't hesitate to reach out to our team via our Slack at `#cohort-3-help`. Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.