Coding Problems

Objective

This assignment aims to demonstrate how to study a data structures or algorithms question in depth to prepare for an industry coding interview. Leetcode is a popular coding practice site that many use to practice for technical interviews. Like behavioral interviews, it's important to practice and keep your skills sharp.

Group Size

Please complete this individually.

Part 1:

You will be assigned one of three problems based of your first name. Execute the code below, and that will tell you your assigned problem. Include the output as part of your submission (do not clear the output). The problems are based-off problems from Leetcode.

```
print((hash('Yuri') % 3) + 1)
```

```
1
```

Starter Code for Question 1

```
# class TreeNode(object):
#     def __init__(self, val = 0, left = None, right = None):
#         self.val = val
#         self.left = left
#         self.right = right
def is_duplicate(root: TreeNode) -> int:
      # TODO
```

Starter Code for Question 2

```
# 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
def bt_path(root: TreeNode) -> List[List[int]]:
  # TODO
```

Starter Code for Question 3

```
def missing_num(nums: List) -> int:
  # TODO
```

Part 2:

- Paraphrase the problem in your own words

```
# Your answer here

"""Find if any number in a binary tree appears more than once with the
following conditions: If a number is duplicated, return it. If there
are several duplicated numbers, return the one nearest to the root. If
no duplicates are found, return -1 to signify no duplicates."""
```

- 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.

```
# Input: root = [0, 3, 5, 7, 2, 7, 12];
# Output is 7 as it is the duplicate value;

# Input: root = [6, 2, 5, 1];
# Output is -1 as no duplicates were found;
```

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

```
from collections import deque
class TreeNode(object):
     def __init__(self, val = 0, left = None, right = None):
         self.val = val
         self.left = left
         self.right = right
def is_duplicate(root: TreeNode) -> int:
    if not root:
        return -1
    queue = deque([(root, 0)])
    value_distance = {}
    min_distance = float('inf')
    result = -1

    while queue:
        node, distance = queue.popleft()

        if node.val in value_distance and distance < min_distance:
            min_distance = distance
            result = node.val
        else:
            value_distance[node.val] = distance
        if node.left:
            queue.append((node.left, distance + 1))
        if node.right:
            queue.append((node.right, distance + 1))

    return result

# Creating nodes
root = TreeNode(0)
node1 = TreeNode(3)
node2 = TreeNode(5)
node3 = TreeNode(7)
node4 = TreeNode(2)
node5 = TreeNode(7)
node6 = TreeNode(12)


# Building the tree
root.left = node1
root.right = node2
node1.left = node3
node1.right = node4
node2.left = node5
node2.right = node6

# Calling the function
print(is_duplicate(root))
```

```
7
```

- Explain why your solution works

```
"""The is_duplicate function searches the first duplicate value in a
binary tree by performing the "breadth-first search (BFS)". At the
initialization phase the function checks if the root of the tree is
None. If it is, the function returns -1 (no duplicates).

Next it initializes a queue with the root node and its distance from
the root ("0"), a dictionary value_distance to store the distances of
the node values from the root, and two variables min_distance and
result. Thereby it keeps track of the minimum distance of a duplicate
value from the root and the corresponding value. The BFS (while) loop
runs until there are no more unchecked nodes in the queue.

Subsequently the function checks if the value of the dequeued node is
already in the value_distance dictionary (signals presence of a
duplicate). If the distance of a duplicate is less than min_distance,
the function updates min_distance and result with the current distance
and value.

The function then checks if the dequeued node has a left or right
```

```
child and enqueues the child and its distance from the root (= the
parent’s distance + 1).

Finally, the function returns result: the value of the first duplicate
node, or -1 if no duplicates were found.

To sum it up, this solution works, because it systematically visits
all the nodes of the tree in order of their distance from the root,
and keeps track of the first duplicate value it encounters.



 However, please note that this function assumes that all node values
are integers and that the tree is a binary tree. If these assumptions
are not met, the function may not work as expected. Please make sure
to adjust the function as needed to match your specific requirements
and constraints."""
```

- Explain the problem’s time and space complexity

```
"""The time complexity of this function is O(n) (where "n" is the
number of nodes), because the function visits each node once.
The space complexity is also O(n), because in the worst case, it needs
to store all the nodes in the queue or the dictionary. This makes the
function efficient for large trees."""
```

- 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)

```
"""One such alternative is the depth-first search (DFS) with
recursion. In this approach the function goes as deep as possible
along each branch before backtracking.

Step 1: Create an empty set to keep track of the values of the nodes
that have been visited.

Step 2: Define a recursive function that performs a DFS on the tree.
The function should take a node and its depth as parameters.

Step 3: If the node is 'None', return '-1'.

Step 4: If the value of the node is already in the set, return the
value.

Step 5: Add the value of the node to the set.

Step 6: Recursively call the DFS function on the left and right
children of the node. If either call returns a value other than '-1',
return that value.
```

```
Step 7: If the DFS function has visited all the nodes and hasn't found
a duplicate, return '-1'."""
```

Evaluation Criteria

- Problem is accurately stated
- Two examples are 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
- Clarity in the proposal to the alternative solution

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-1`
- What to submit for this assignment:
  - This Jupyter Notebook (assignment_1.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:

- ☐ Create a branch called `assignment-1`.
- ☐ Ensure that the repository is public.
- ☐ Review the PR description guidelines and adhere 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.