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

Parts:

- Part 1: Complete 1 of 3 questions
- Part 2: Review your partner's Assignment 1 submission
- Part 3: Perform code review of your partner's assignment 1 by answering the questions below
- Part 3: Reflect on Assignment 1 and Assignment 2

Part 1:

You will be assigned one of three problems based of your first name. Enter your first name, in all lower case, 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.

```python
import hashlib

def hash_to_range(input_string: str) -> int:
     hash_object = hashlib.sha256(input_string.encode())
     hash_int = int(hash_object.hexdigest(), 16)
     return (hash_int % 3) + 1
input_string = "pulkit"
result = hash_to_range(input_string)
print(result)
```

```
2
```

▶ Question 1

Starter Code for Question 1

```python
# 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 is_duplicate(root: TreeNode) -> int:
  # TODO
```

```
  File "/tmp/ipython-input-2-2850872121.py", line 8
    # TODO
          ^
SyntaxError: incomplete input
```

Next steps: Explain error

▶ Question 2

Starter Code for Question 2

```python
from typing import Optional, List


# 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]]:
  # Handling extreme edge case, in case, TreeNode is null or empty
  if not root:
    return []

  paths = []

  def dfs(node, path):
    # Add the current node's value to the path
    path.append(node.val)

    # If it's a leaf node, append the path to paths
    if not node.left and not node.right:
        paths.append(path[:])  # Make a copy of the current path
    else:
        # Otherwise, go deeper
        if node.left:
            dfs(node.left, path)
        if node.right:
            dfs(node.right, path)

    # Backtrack for other paths
    path.pop()

  dfs(root, [])
  return paths

def build_tree(arr: List[int]) -> Optional[TreeNode]:
    if not arr:
        return None
    nodes = [TreeNode(val) for val in arr]
    sizeOfArray = len(arr)

    for i in range(sizeOfArray):
        if nodes[i]:

            left_idx = 2 * i + 1
            right_idx = 2 * i + 2

            if left_idx < sizeOfArray:
                #Assigning left TreeNode
                nodes[i].left = nodes[left_idx]
            if right_idx < sizeOfArray:
                #Assigning right TreeNode
                nodes[i].right = nodes[right_idx]

    #Returning the root node
    return nodes[0]

arr = [1, 2, 2, 3, 5, 6, 7]
root = build_tree(arr)
print(bt_path(root))


arr_second = [10, 9, 7, 8]
root_second = build_tree(arr_second)
print(bt_path(root_second))
```

```
[[1, 2, 3], [1, 2, 5], [1, 2, 6], [1, 2, 7]]
[[10, 9, 8], [10, 7]]
```

▶ Question 3

Starter Code for Question 3

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

Part 2:

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

Part 3:

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.

> We are given a list of integers, and we have to go through them, and find the first duplicate element we found.
> And in case, there is no duplicates, we return -1.

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

```
input = [1, 2, 3, 4, 5]
output = -1
#We didn't find any duplicates

input_second = [11, 53, 52, 63, 13, 64, 11, 35, 64, 52]
output_second = 11
#We found the duplicate, first as 11
```

- Copy the solution your partner wrote.

```
from typing import List

def first_duplicate(nums: List[int]) -> int:
    seen_elements = set()
    for item in nums:
        if item in seen_elements:
            return item
        else:
            seen_elements.add(item)
    return -1

# Show output results with two input examples
list_1 = [34, 11, 41, 12, 52]
list_2 = [10, 30, 20, 40, 20, 50, 50, 10]

print(f"Input: nums = {list_1}")
print(f"Output: {first_duplicate(list_1)}")

print(f"Input: nums = {list_2}")
print(f"Output: {first_duplicate(list_2)}")
```

```
Input: nums = [34, 11, 41, 12, 52]
Output: -1
Input: nums = [10, 30, 20, 40, 20, 50, 50, 10]
Output: 20
```

- Explain why their solution works in your own words.

> Initially an empty set is created. And we iterate through the array, and check if item is in set or not. It should take O(1) time.
>
> If item is not in set, we add it to this. In case we reach the end of array, we return -1.

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

> **Time Complexity: O(n)**
>
> Time Complexity is O(n), as in case there is no duplicates, we need to reach the end of the array. That is the worst case scenario here.

> **Space Complexity: O(n)**
>
> Space complexity is also, O(n), due to the fact, that when we don't have any duplicates, we still need to store, all those items in set, to check for duplicates.

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

> Their solution works perfectly, and fulfills the requirement to find the first duplicate number in the array, or return -1 if no duplicates are found.
>
> Only other thing can think of adjusting, is that we can handle the edge cases like empty and only 1 item in array, and return -1 beforehand.

Part 4:

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, was fun and challenging in it's own way. The challenging part in question 2, was to figure out, to match the sequence in which bracket close. Luckly stack data structure is there for this. It got a bit confusing, as I wrote my first brute force solution in Part 1 only, along with examples, and when I saw more examples defined in the question above, I realized my mistake, and started thinking of correct solution.

> Assignment 2, code review is fun too, it teaches us, how to understand the problem, read the code, understand the proposed solution, and come up with ways, in which the code is perfect.

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-6-help`. Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.