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.

```
""" The Question Two "Path to Leaves" is to find all the unique paths
from the root to the leaves in a binary tree. The binary tree is given
by its root node. The paths should be returned in any sequence.

The task is to identify what kind of traversal method this is and to
return all the paths from the root to the leaves. In the first example
the expected output for the binary tree [1, 2, 2, 3, 5, 6, 7] would be
[[1, 2, 3], [1, 2, 5], [1, 2, 6], [1, 2, 7]]. In another example, if
the binary tree has root node values as [10, 9, 7, 8], the expected
output would be [[10, 7], [10, 9, 8]]."""
```

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

```
""" Let the new example have root node 5 with two branches: 4 and 8.
The branch 4 nodes into 11, splitting into 7 and 2. The branch 8
splits into 13 and 4, and 4 leads to 1.

        5
       / \
      4   8
     /   / \
    11  13  4
```

```
   / \     /
  7   2   1

In other words input is root = [5, 4, 8, 11, 13, 4, 7, 2, 1]  The
paths from the root to the leaves are [[5, 4, 11, 7], [5, 4, 11, 2],
[5, 8, 13], [5, 8, 4, 1]]."""
```

- Copy the solution your partner wrote.

```
# 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

root = TreeNode(val = 1)
root.left = TreeNode(val = 2)
root.right = TreeNode(val = 2)
root.left.left = TreeNode(val = 3)
root.left.right = TreeNode(val = 5)
root.right.left = TreeNode(val = 6)
root.right.right = TreeNode(val = 7)

# preorder traversal

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

   def dfs (node:TreeNode, path, allpaths):

      if node:
         path.append(node.val)
         # remove the comments to see what is happening on each
recursive call
         #print ("node",node.val)
         #print ("path:",path)
         #print ("all paths:",allpaths)

      if node.left:
         dfs(node.left, path, allpaths)
         path.pop() #remove the leaf number

      if node.right:
         dfs(node.right, path, allpaths)
         path.pop() #remove the leaf number

      if not node.right and not node.left:
         allpaths.append (path[:])
```

```
        return allpaths

    path=[]
    allpaths=[]
    path = dfs (root, path, allpaths)

    return allpaths


btree_paths = []
btree_paths = bt_path(root)
print ("final list of all paths:", btree_paths)
```

- Explain why their solution works in your own words.

```
"""This solution works by using a Depth-First Search (DFS) algorithm.
The DFS algorithm starts at the root of the tree and explores as far
as possible along each branch before backtracking.

The main f(x) (bt_path) initializes two empty lists (path and
allpaths). "path" stores the current path from the root to a node, and
"allpaths" accumulates all such paths. The DFS f(x) takes three
arguments: the current "node" "path" and the list of "allpaths". If
the "node" is not "None", its value is appended to the current "path".
If the current `node` has a right child, the DFS f(x) is recursively
called removing the last node from the "path" (path.pop()). The same
applies for a right child in the node. If the "node" ends (no
children), the current "path" is added to "allpaths". Finally, the DFS
f(x) returns "allpaths" with all the paths in the tree.

This solution works because DFS systematically explores all paths down
the binary tree, and keeps track of all paths."""
```

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

```
"""Each node is visited once, therefore the time complexity of this
solution is O(N) (where N is the number of nodes in the tree).
The space complexity is also O(N), because in the worst case scenario
(the tree is a straight line), the maximum length of the path could be
N."""
```

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

```
"""As as in anything else there are stronger and weaker sides to this
solution.

- The DFS used in this solution is an efficient way to queue a binary
tree and explore all paths.
- Also the use of path and allpaths lists is a common practice to keep
```

```
track of the current path and all paths respectively.
- In addition the use of path.pop() after each recursive call ensures
that the path list only contains nodes from the current path.

- On the other hand, the comments could be more descriptive to better
explain what each part of the code does. For example, it would be
helpful to explain why path.pop() is used and why a copy of path is
added to allpaths.
- The function name "dfs" can be misleading. A more descriptive name
e.g. "preorder_traversal" or "explore_paths" might be more
informative.
- It is a good practice to handle the case where the root node is
None. And it is (excuse my tautology) not the case.

To sum it up the areas of improvement are optional and do not affect
the efficient and correct solution provided by Mario."""
```

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

```
"""The assignment at the algorithms and data structures module were
surprising in several ways, but overall, I enjoyed the idea creators'
had when developing this assignment.
The element of game in playing the lottery to be allotted a task, and
reviewing the peer work added the element of excitement to the whole
process. However, even more important was a step-by-step unrolling of
the assignment, which made me conceptualize the problem first, without
jumping into the code headlong. Although as a “visual” person I would
draw the algorithm diagram or sketch pseudo-code, describing the task
in my own words allowed me to comprehend the task better.
The review of my partner's work was quite interesting. It showed a
different approach to traverse the binary tree, and raised question on
efficiency, use cases and differences in implementation of the BFS and
DFS algorithms. It also made me reflect on my own coding habits. I
must admit that some points of improvement to Mario's code (not enough
comments, confusing choice of f(x)/variables names) applies to me in a
greater extent. The only upsetting experience was the prolonged
struggle to convert ipynb to pdf. To sum it up, I think it was a great
learning experience."""
```

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.