

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.

```
In [ ]: # Recording all possible paths in a binary tree given its root.
```

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

```
In [ ]: # New Exmaple:
# Input = [6,2,3,5,6,7,8,9]
# Output = [[6,2,5,9],[6,2,6],[6,3,7],[6,3,8]]

# Tracing a partner's example:
# Input: root = [2, 1, 3]
# Output: [[2, 1], [2, 3]]
# In this example there is a tree with 2 levels. Actually a node and two childs
# at the left and right sides. So,
# there are two paths which start by node and then right/left leafs as presented
# in paths list of lists in output.
# Trace with code:
# stack = [(TreeNode(1),[1])] # initial stack
```

```
# stack = [(TreeNode(3), [1, 3]), (TreeNode(2), [1, 2])] # after adding right
# and left child to the stack
# stack = [(TreeNode(3), [1, 3])] # after second pop
# stack = [] # after third pop
# paths= [[1,2],[1,3]]
```

- Copy the solution your partner wrote.

```
In [ ]: from typing import 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]]:
    if not root:
        return []

    paths = []
    stack = [(root, [root.val])]

    while stack:
        node, path = stack.pop()

        if node.left is None and node.right is None:
            paths.append(path) # Append completed path

        if node.right:
            stack.append((node.right, path + [node.right.val]))

        if node.left:
            stack.append((node.left, path + [node.left.val]))

    return paths
```

```
In [2]: # Testing new example root=[6,2,3,5,6,7,8,9]:
root3 = TreeNode(6)
root3.left = TreeNode(2)
root3.right = TreeNode(3)
root3.left.left = TreeNode(5)
root3.left.right = TreeNode(6)
root3.right.left = TreeNode(7)
root3.right.right = TreeNode(8)
root3.left.left.left = TreeNode(9)

print(bt_path(root3))
```

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

- Explain why their solution works in your own words.

```
In [ ]: # The main function of the solution (bt_path) is getting the root of the table as
# an input and giving a list of the lists of all possible paths as an output.
# In this function, an stack has been considered to keep track of the traversed
# nodes so that if there is a child it will be added to the stack in addition to
# its path and then popped to make the paths. Since the stack is FILO, first the
# right node has been considered and then the left node so that
# by doing pop preorder traversal is implemented.
```

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

```
In [ ]: # Time complexity:
# This can be measured by considering the traversed nodes and edges in the
# algorithm processes:
# 1-Traversal of Nodes:  $O(n)$  where  $n$  is the number of nodes as they are
# pushed and popped to the stack at least once.
# 2-Building Paths:  $O(n)$  for a balanced binary tree and  $O(n^2)$  for a
# skewed tree (the complexity of path creation will
# be  $O(n)$  for each node)
# Overall:  $O(n^2)$ 

# Space Complexity:
#1- Stack:  $O(n)$ : in worst case stack will hold all nodes( $n$ )
#2- Path's List:  $O(n^2)$ :  $n$  paths that each could take up to  $n$  space
# Overall:  $O(n^2)$ 
```

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

```
In [ ]: # The time and space complexity of both initial and modified solutions
# are  $O(n^2)$ . For the modified solution it is expected
# to suggest an algorithm which is more efficient and make the time/space
# complexity better.
# For adjustment, I think the time and complexity of the code needs
# modification since it is  $O(n^2)$  rather than  $O(n)$ .
```

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

```
In [ ]: # For assignment 1 I implemented the tree and then traversed by DFS to find
# duplicates and their depths. The overall time and space complexity of the
# code was  $O(n)$  but in order to optimize the solution, trying different types
# of traverse methods like BFS was suggested. Since the question is asking
# to consider depth as well of duplication, BFS solution may have better
# efficiency. Also, I think I could bypass the tree building step and just
```



```
# make it for the desired trees in examples.
# In my Reviewing experience in assignment 2, I found the solution was
# implemented without building the general tree and juts implemented for
# the trees on examples which was different from my approach. In the
# reviewing process, looking at the problem while not being involved
# in the details of implementation gave me the ability to look at the
# solution from a different perspective and focusing more on the big
# picture and overall effect which was a good experience. Comparing
# the suggested solutions, the time and space complexity of the suggested
# solution was  $O(n^2)$  which could be optimized to be more efficient I
# think. Also, in the assignment this complexity was considered  $O(n)$ 
# that I think it may need a revision.
```

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/
```

 - 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.