Assignment 1 Review

Paraphrase the problem in your own words.

The task is to create a Python function that takes the root of a binary tree as input and generates all possible paths from the root to the leaves of the tree. The order of these paths in the output is flexible.

Create 1 new example that demonstrates you understand the problem.

New Example

10

/ \

5 15

/ \ \

3 7 20

new_root = TreeNode(10) new_root.left = TreeNode(5) new_root.right = TreeNode(15) new_root.left.left = TreeNode(3) new_root.left.right = TreeNode(7) new_root.right.right = TreeNode(20) new_result = bt_path(new_root) print(new_result)

Expected Output: [[10, 5, 3], [10, 5, 7], [10, 15, 20]]

Trace/walkthrough 1 example that your

partner made and explain it.

1

/ \

2 3

/ \

4 5

```
root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root.left.right = TreeNode(5)

result = bt_path(root)

print(result)
```

Expected Output: [[1, 2, 4], [1, 2, 5], [1, 3]]

Explanation: The function starts traversing the tree from the root (1). It appends the current node's value to the current_path list. When it reaches the leaf

nodes (4, 5), it appends the path to the paths list. The function then backtracks to explore other paths.

Copy of Solution

In [ ]:
```python
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]]:
    paths = []  # To store the result

    def dfs(node, current_path):
        if not node:
            return
        current_path.append(node.val)
        if not node.left and not node.right:  # Leaf node
            paths.append(list(current_path))
        else:
            dfs(node.left, current_path)
            dfs(node.right, current_path)
        current_path.pop()  # Backtrack

    dfs(root, [])
    return paths
```

Explain why their solution works in your own words.

The solution uses depth-first search (DFS) to traverse the binary tree, maintaining the current path. When a leaf node is reached, the current path is added to the result. The backtracking step is crucial to explore all possible paths.

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

Time Complexity: O(N) The function traverses each node once. Space Complexity: O(H) The space required for the recursive call stack, where H is the height of the binary tree.

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

The solution is correct and efficiently utilizes DFS for path generation. However, it could be enhanced by handling edge cases, such as an empty tree, to make it more robust.

Reflection

In completing review of Assignment 1, I deepened my understanding of binary tree traversal and recursion. Collaborating with my partner during the presentation and review process allowed me to gain insights into different problem-solving approaches. Discussing the examples and solutions strengthened my comprehension and honed my ability to articulate solutions effectively. Moving forward, I aim to apply these collaborative learning experiences to tackle more complex problems and improve my programming skills.