Assignment 1 - Eliezer Molina Mello

Hashed first name to determine which problem to work on.

```
import timeit
import random
from typing import List
```

```
(hash("Eliezer")%3)+1
```

```
2
```

Question Two: Path to Leaves

Given the `root` of a binary tree, return all root to leaf paths in any order.

```
# Starter code
# 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
```

Paraphrase the problem in your own words

The problem to solve is to write a piece of code that can give us the complete path (from root to leaves) of any binary tree. As asked during office hours, this algorithm needs to be able to return a path of a list that returns all paths in any order.

This problem also involves analyzing the tree in such a way that every path is identified and recorded, regardless of the order in which these paths are listed. The objective is to ensure that for any given binary tree structure, we can display all the unique routes that link the tree's origin (root) to its terminal points (leaves).

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.

```
# Example 1:

    # Input: root = [2, 5, 5, 6, 7, 7, 8]

    # Output: [[2, 5, 6], [2, 5, 7], [2, 5, 7], [2, 5, 8]]

# Example 2:

    # Input: root = [2, 3, 4, 6, 9]

    # Output: [[2, 3, 6], [2, 3, 9], [2, 4]]
```

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

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

# Create an extra function in order to distribute the root values
def insertLevelOrder(arr, root, i, n):
    # Base case for recursion
    if i < n:
        temp = TreeNode(arr[i])
        root = temp

        # insert left child
        root.left = insertLevelOrder(arr, root.left, 2 * i + 1, n)

        # insert right child
        root.right = insertLevelOrder(arr, root.right, 2 * i + 2, n)
    return root

# Create the function use the started code provided as base
def bt_path(root: TreeNode) -> List[List[int]]:
    if not root:
        return []

    paths = []

    def find_paths(node, current_path):
        if node:
            # Append the current node's value to the path
            current_path.append(node.val)

            # If it's a leaf node, add the path to the paths list
            if not node.left and not node.right:
                paths.append(list(current_path))
            else:
                find_paths(node.left, current_path)
                find_paths(node.right, current_path)

            # Backtrack to explore other paths
            current_path.pop()

    find_paths(root, [])
    return paths

# Insert the root provided in the q2.md file to confirm output
arr = [1, 2, 2, 3, 5, 6, 7]
n = len(arr)

# Configure insertLevelOrder function
root = None
root = insertLevelOrder(arr, root, 0, n)

# Call the bt_path function and print paths
print(bt_path(root))
```

```
     [[1, 2, 3], [1, 2, 5], [1, 2, 6], [1, 2, 7]]
```

```
# Check to see if code runs successfully

# Given another root, which is my own example provided above
arr = [2, 3, 4, 6, 9]
n = len(arr)
root = None
root = insertLevelOrder(arr, root, 0, n)

# Call the bt_path function and print paths
print(bt_path(root))
```

```
     [[2, 3, 6], [2, 3, 9], [2, 4]]
```

Explain why your solution works

My solution works because, after analyzing the sample root and output (Example 1 in q2.md), I understood that it represents a level order traversal of a binary tree, so I can construct the tree accordingly and then implement a function to find all root-to-leaf paths. To build the tree from its level order traversal, I iteratively inserted nodes into the tree following the order of the list. My solution ensures that every possible path from the root to the leaves is considered.

For added context, my solution starts at the root and ensures that both left and right branches are explored if they exist, covering all possible

Explain the problem's and time complexity and space complexity

Analyzing the problem's complexity:

In terms of time complexity:

- Constructing the tree from a level order traversal has a time complexity of `O(n)`, where `n` is the number of nodes in the tree. This is because we iterate through the list once to create all nodes and set their left and right children.
- Finding all root-to-leaf paths involves a Depth-First Search (DFS) traversal of the tree, as discussed in class. In the worst case, we visit each node exactly once, which also gives us a time complexity of `O(n)` for this part of the task.
- Therefore, I believe the overall time complexity of the solution is `O(n) + O(n) = O(n)`

In terms of space complexity:

- The space complexity for constructing the tree is `O(n)` for storing the tree itself.
- For finding all paths, the maximum amount of space is used by the call stack during the recursive DFS, which in the worst case (such as a skewed tree) can be `O(n)`.
- Therefore, the overall space complexity of the solution is `O(n)` for the tree construction and `O(n)` for the DFS traversal and path storage, making it `O(n)`.

In summary, based on my understanding of the problems complexity and the topics learned I can say that both time and space complexity is `O(n)`, where `n` is the number of nodes in the tree.

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)

I believe that regarding an alternative solution to find all `root to leaf` paths in a binary tree, using sorting algorithms like insertion sort or merge sort isn't applicable, these were discussed in class. Also, these sorting algorithms are designed to organize a linear sequence of elements into a specified order and don't deal with tree structures or pathfinding within trees.

However, an alternative approach to explore would be the Breadth-First Search (BFS) method. Instead of exploring a particular branch, BFS explores the tree level by level. This method involves keeping track of paths in a sequence, where each entry in the sequence represents a path from the root to the current node. For each node visited, you would add two new paths to the queue (if the node has branches): one path including the left branch and another including the right branch. Finally, when a leaf node is reached, the path leading to it is a complete `root to leaf` path and can be apppended to the path list.

End of assignment 1.