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 goal is to find all paths in a binary tree starting from the root and ending at each leaf node. A leaf node is defined as a node with no left or right child. The solution should return a list of lists, where each inner list represents a unique root-to-leaf path.

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

New Example Input: Tree: [5, 3, 8, 1, 4]

Output: Paths: [[5, 3, 1], [5, 3, 4], [5, 8]]

Explanation:

Root node 5 has left child 3 and right child 8. Node 3 has children 1 and 4, which are leaf nodes. Node 8 is a leaf node.

Trace One Partner Example:

Input: [10, 9, 7, 8]

Steps:

Start at root node 10. Add 10 to the path.

Traverse left to node 9. Add 9 to the path.

Traverse left to node 8. Add 8 to the path.

Node 8 is a leaf node. Save path [10, 9, 8].

Backtrack to node 10 and traverse right to node 7.

Node 7 is a leaf node. Save path [10, 7].

Output: [[10, 9, 8], [10, 7]]

- Copy the solution your partner wrote.

The partner wrote two codes! I got confused. I selected one of them for here!

In [2]:
```python
from typing import List
```

In [3]:
```python
class TreeNode:
    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]]:
    def dfs(node, path, result):
        if not node:
            return

        # Add the current node's value to the path
        path.append(node.val)

        # If leaf node, add to the result
        if not node.left and not node.right:
            result.append(path[:])      # Use slicing to copy the path
        else:
            # Recurse on left and right children
            dfs(node.left, path, result)
            dfs(node.right, path, result)

        # Backtrack to explore other paths
        path.pop()

    result = []
    dfs(root, [], result)
    return result
```

```
# Test
def create_tree(values):
    if not values:
        return None
    nodes = [TreeNode(val) if val is not None else None for val in values]
    for i, node in enumerate(nodes):
        if node:
            left_index = 2 * i + 1
            right_index = 2 * i + 2
            if left_index < len(nodes):
                node.left = nodes[left_index]
            if right_index < len(nodes):
                node.right = nodes[right_index]
    return nodes[0]
```

- Explain why their solution works in your own words.

Approach: This approach to explore all paths from the root to each leaf node.

Path Construction: It appends the current node’s value to a path list during traversal.

Leaf Node Detection: If a node has no left or right children, it is identified as a leaf node, and the current path is added to the result list.

Backtracking: After processing a node, it removes the node from the path to backtrack and explore other potential paths.

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

Time Complexity:

The function visits each node exactly once. For n nodes, the time complexity is linear, O(n).

Space Complexity:

The space complexity is determined by the height of the tree (h), which represents the maximum recursion depth. In a skewed tree, h could be n (worst case -> O(n)).

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

The solution is well-structured and uses recursion effectively. Backtracking is implemented correctly to ensure paths are not overwritten or mixed up. Edge cases (e.g., empty trees) are handled well.

However, I think the following items could be usefull for improvement:

Code Readability: Add comments to explain each step of the DFS process for clarity.

Alternative Approach Mention: The explanation could benefit from briefly discussing the BFS approach in addition to the DFS method.

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

I found this review process insightful, as it gave me a deeper understanding of solving problems in a binary tree context. Reviewing my partner’s solution helped me see how others approach recursion and DFS in Python. It also emphasized the importance of clear code documentation and handling edge cases in real-world coding scenarios. I appreciated the opportunity to learn from their thought process. This practice has improved my ability to analyze code critically and reinforced concepts like DFS, recursion, and tree traversal.

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_struc`
  - 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.