assignment_2

July 21, 2024

1 Practice Interview

1.1 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.

1.2 Group Size

Each group should have 2 people. You will be assigned a partner

1.3 Part 1:

You and your partner must share each other's Assignment 1 submission.

1.4 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.

```
[ ]: # Your answer here
```

Given a list of integers where the numbers range from 0 to n, including possible duplicates, the task is to identify the specific numbers missing within this range. The range starts at 0 and ends at n, where 0 is the smallest number and n is the largest. If all numbers within the range are present in the list, the function should return -1. If there are any missing numbers, the function should return a list of these absent numbers.

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

```
[ ]: # Your answer here
```

The New example. Inputs = [0, 2, 2, 1, 5] Output was [3]

My partner

Inputs = [1, 2, 3, 5] The proposed solution indetifies the following outputs [1, 2, 5] In this case, with a list of 4 numbers, we should see the numbers 1, 2, 3 and 5. However, since the list can only contain 3 numbers, there will always be a missing number. For instance, in this example, the number 3 is missing.

- Copy the solution your partner wrote.

[ ]:
```
# Example 1
#     1
#    / \
#   2   3
#    \
#     5
# There are two Root-to-leaf paths:

# Path 1: [1, 2, 5]
# Path 2: [1, 3]
# And the expected output of the problem should be [[1, 2, 5], [1, 3]]

# Example 2
#      10
#     /  \
#    5    15
#   / \     \
#  3   7    18


# There are two Root-to-leaf paths:

# Path 1: [10, 5, 3]
# Path 2: [10, 5, 7]
# Path 3: [10, 15, 18]

# And the expected output of the problem should be [[10, 5, 3], [10, 5, 7],␣
↪[10, 15, 18]]
```

- Explain why their solution works in your own words.

[ ]:
```
# Your answer here
```

This solution involves finding all root-to-leaf paths in a binary tree using a depth-first search (DFS) approach. The bt_path function initiates a DFS traversal from the root, constructing paths as it visits each node. When a leaf node is reached, the current path is saved. The helper function insertLevelOrder is used to construct the binary tree from an array representation, allowing us to test the bt_path function with different tree structures. The result is a list of lists, where each inner list represents a path from the root to a leaf node.

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

```
[ ]: # Your answer here
```

Time Complexity: During the depth-first search (DFS) traversal, each node and edge in the binary tree is visited and processed exactly once. Therefore, the time complexity of the DFS itself is O(n), where n is the number of nodes in the tree. Additionally, the total time spent on copying paths during this traversal is also bounded by O(n), as each node and edge is processed exactly once, and each path is copied exactly once.

Space Complexity: The space complexity is influenced by both the recursion stack and the storage required for the paths. The recursion stack's depth depends on the height of the tree, leading to a space complexity of O(h), where h is the height of the tree. Furthermore, since we store all root-to-leaf paths and each path can potentially contain up to O(n) nodes, the total space required for storing these paths is O(n). Therefore, combining both aspects, the overall space complexity is O(n) due to the recursion stack and the storage of paths.

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

```
[ ]: # Your answer here
```

The solution efficiently finds all root-to-leaf paths in a binary tree using a depth-first search (DFS) approach. The bt_path function explores each path from the root to the leaves, collecting all such paths. For example, in the tree [1, 2, 3, None, 5], it correctly identifies the paths [1, 2, 5] and [1, 3], while in the tree [10, 5, 15, 3, 7, None, 18], it finds the paths [10, 5, 3], [10, 5, 7], and [10, 15, 18]. The implementation handles these cases well, with a time complexity of O(n) and a space complexity of O(n). However, for very large trees, it might be beneficial to consider an iterative DFS approach to avoid potential stack overflow issues and to handle edge cases more robustly.

1.5 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.

1.5.1 Reflection

```
[ ]: # Your answer here
```

The assignment question initially posed a significant challenge, requiring multiple iterations to arrive at the final solution. My lack of experience with recursive functions became apparent, but this assignment provided valuable practice in developing and understanding recursive solutions. Through rewriting functions, creating new ones, and reorganizing code, I began to question whether professional developers can create optimized functions from the start or if they, too, make iterative changes during development.

In reviewing the partner assignments, I found it reassuring to see similar problem-solving approaches and function choices used by peers. This confirmed that my thought process was on the right track.

Overall, the course has sparked a deeper interest in developing complex functions and integrating various functions into my work. It highlighted the importance of clear problem definitions, iterative *refinement, and the value of peer feedback in improving coding skills and problem-solving strategies.*

1.6 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

1.7 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.

1.7.1 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.