Notebook

February 26, 2025

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.

[36]:
```
# We need to print out all possible paths starting from root to leaves. In a␣
 ↪graph, paths are any sequence of nodes connected by edges.
# The order of the paths is not important.
```

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

[ ]:
```
# new example: in this three: 1(2(4,5),3) all path from root to leaves are␣
 ↪1-2-4, 1-2-5, 1-3
# this is one example that my partner provided:
# root3= TreeNode(1)
# root3.left = TreeNode(2)
# root3.right = TreeNode(3)
# root3.left.left = TreeNode(5)
# root3.left.right = TreeNode(8)
# root3.right.left = TreeNode(13)
```

```
# root3.right.right = TreeNode(21)
# print(bt_path(root3))

# The algorithm start with root, node 1. Then following DFS, it traverse the␣
 ↪tree and keep each path by adding newly visited node to the
# current path, till rech a leaf.
```

- Copy the solution your partner wrote.

[38]:
```
from typing import List

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]]:
    paths = []

    if not root:
        return []

    def dfs(node, path):
        path.append(node.val)
        if not node.left and not node.right:
            paths.append(path)
        if node.left:
            dfs(node.left, path.copy())
        if node.right:
            dfs(node.right, path.copy())


    dfs(root, [])
    return paths

root1 = TreeNode(1)
root1.left = TreeNode(2)
root1.right = TreeNode(2)
root1.left.left = TreeNode(3)
root1.left.right = TreeNode(5)
root1.right.left = TreeNode(6)
root1.right.right = TreeNode(7)

root2 = TreeNode(10)
root2.left = TreeNode(9)
```

```
root2.right = TreeNode(7)
root2.left.left = TreeNode(8)

print(bt_path(root1))
print(bt_path(root2))
```

```
[[1, 2, 3], [1, 2, 5], [1, 2, 6], [1, 2, 7]]
[[10, 9, 8], [10, 7]]
```

- Explain why their solution works in your own words.

[39]:
```
# This method works properly because it uses a depth-first search (DFS) approach to traverse the tree.
# The method starts from the root node and explores each path by recursively visiting the left and right child nodes.
# When it reaches a leaf node (a node with no children), it adds the current path to the list of paths. The algorithm explores
# every possible path from the root to all the leaf nodes because all the following premises are true: 1. all pathe starts with
# root and ends with a node, 2. all nodes will be visited using DFS search so all path are listed.
```

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

[ ]:
```
# The time complexity of the problem is O(N), where N is the number of nodes in the tree, because we visit each node once.
# The space complexity is O(D * N), where D is the depth of the tree and N is the number of leaves.
```

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

[ ]:
```
# The are two issue in his responces to the questions:
# First, he said that the space complexity is O(N), but as in his code all path is kept till the end of the method, nodes are stored
# multiple times depending to the depth of each node. I guess the space complexity is O(D * n), that D is the depth of the tree and n
# is the number of leaves.
# Second, the last question asked for alternative solution for the problem, but my partner explained about alternative solution for
# creating the binary tree. For example, he can use bfs instead of dfs, or any other traverse algorithm.
```

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

```
[ ]: # For the first assignment, I was tasked with solving a problem related to␣
      ↪detecting duplicate values in a binary tree.
     # The problem asks for a method to check if a tree contains duplicate values.␣
      ↪The solution needs to traverse the tree, for example
     # using DFS search, and identify duplicate values by keeping the set of all␣
      ↪visited values and searching for the value of each newly
     # visited node in this set. This approach ensures that I check every node in␣
      ↪the tree using dfs, while seeking for duplicates.
     # The DFS allows me to explore each branch fully before moving on to the next␣
      ↪one. The time complexity of this approach is O(N),
     # where N is the number of nodes in the tree, as we visit each node once. The␣
      ↪space complexity is also O(N) because of the need to
     # store the visited values in a set of size N.

     # In the second assignment, I reviewed my partner's solutions for finding all␣
      ↪paths in a binary tree problem. This review includes
     # paraphrasing the problem, creating examples, and analyzing their approach.␣
      ↪For example, in my partner's assignment, they work on
     # paths in a binary tree and use depth-first search (DFS) to find all possible␣
      ↪paths from the root to the leaves, and an alternative
     # could be using BFS search. The time complexity of this solution is O(N),␣
      ↪since each node is visited once. The space complexity is
     # O(D * n), where D is the depth of the tree and n is the number of leaves.
```

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.

This notebook was converted with convert.ploomber.io