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.

In [ ]:
```
# Your answer here

''' The task is to determine if a binary tree has any duplicate values among its nodes.
If duplicates are present, the goal is to identify and return the value of the first
duplicate encountered during a Breadth-First Search (BFS) traversal.
This ensures that the duplicate closest to the root is returned.
If no duplicates are found, the function should return -1.
'''
```

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

```
    4
   /  \
  5    6
 / \  / \
7   8 5  9
```

In this tree, the value 5 appears twice: once as the left child of 4 and once as the left child of 6. Expected Output: 5

1. Initialization:
    - Queue: [4]
    - Visited Set: {}
2. First Level:
    - Dequeue 4.
    - Check if 4 is in the visited set: No.
    - Add 4 to the visited set: {4}.
    - Enqueue its children 5 and 6.
    - Queue: [5, 6]
3. Second Level:
    - Dequeue 5.
    - Check if 5 is in the visited set: No.
    - Add 5 to the visited set: {4, 5}.
    - Enqueue its children 7 and 8.
    - Queue: [6, 7, 8]
    - Dequeue 6.
    - Check if 6 is in the visited set: No.
    - Add 6 to the visited set: {4, 5, 6}.
    - Enqueue its children 5 and 9.

   - Queue: [7, 8, 5, 9]

4. Third Level:

   - Dequeue 7.
   - Check if 7 is in the visited set: No.
   - Add 7 to the visited set: {4, 5, 6, 7}.
   - 7 has no children to enqueue.
   - Queue: [8, 5, 9]
   - Dequeue 8.
   - Check if 8 is in the visited set: No.
   - Add 8 to the visited set: {4, 5, 6, 7, 8}.
   - 8 has no children to enqueue.
   - Queue: [5, 9]
   - Dequeue 5.
   - Check if 5 is in the visited set: Yes.
   - Since 5 is already in the visited set, we have found our duplicate.
   - Return 5.

This demonstrates an understanding of how to traverse the binary tree using BFS and detect the first duplicate value.

Let's walk through **Example** 1 step-by-step to find the duplicate value in the given binary tree.

Example 1: [1, 2, 3, 4, 5, 6, 3, 2] The binary tree is structured as follows:

```
        1
       / \
      2   3
     / \   \
    4   5   6
   / \
```

3 2

Initialization

- Queue: [1]
- Visited Set: {}

Level-by-Level Walkthrough

1. First Level:
    - Dequeue 1.
    - Check if 1 is in the visited set: No.
    - Add 1 to the visited set: {1}.
    - Enqueue its children 2 and 3.
    - Queue: [2, 3]
2. Second Level:
    - Dequeue 2.
    - Check if 2 is in the visited set: No.
    - Add 2 to the visited set: {1, 2}.
    - Enqueue its children 4 and 5.
    - Queue: [3, 4, 5]
    - Dequeue 3.
    - Check if 3 is in the visited set: No.
    - Add 3 to the visited set: {1, 2, 3}.
    - Enqueue its child 6.
    - Queue: [4, 5, 6]
3. Third Level:
    - Dequeue 4.
    - Check if 4 is in the visited set: No.
    - Add 4 to the visited set: {1, 2, 3, 4}.
    - Enqueue its children 3 and 2.
    - Queue: [5, 6, 3, 2]
    - Dequeue 5.
    - Check if 5 is in the visited set: No.
    - Add 5 to the visited set: {1, 2, 3, 4, 5}.

- 5 has no children to enqueue.
- Queue: [6, 3, 2]
- Dequeue 6.
- Check if 6 is in the visited set: No.
- Add 6 to the visited set: {1, 2, 3, 4, 5, 6}.
- 6 has no children to enqueue.
- Queue: [3, 2]

4. Fourth Level:

- Dequeue 3.
- Check if 3 is in the visited set: Yes.
- Since 3 is already in the visited set, we have found our duplicate.
- Return 3.

The process finds that the value 3 is the first duplicate encountered during the BFS traversal. Therefore, the function correctly returns 3.

- Copy the solution your partner wrote.

In [ ]:
```
# Your answer here

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

def find_duplicate(root: TreeNode):
    def dfs(node, visited):
        if not node:
            return -1

        if node.val in visited:
            return node.val
        visited.add(node.val)
        left_result = dfs(node.left, visited)
        if left_result != -1:
            return left_result

        right_result = dfs(node.right, visited)
        return right_result

    visited = set()
    return dfs(root, visited)

root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(4)
root.left.left = TreeNode(3)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(1)

print(find_duplicate(root))
```

Explain why their solution works in your own words.

Anwser

The provided solution utilizes a Depth-First Search (DFS) algorithm to detect duplicates in a binary tree. Let's break down the key components and logic of the solution:

Components

1. TreeNode Class:

   - This class defines the structure of a node in the binary tree, containing a value (val), and pointers to the left and right child nodes (left and right).

2. find_duplicate Function:

   - This function serves as the entry point and sets up the initial conditions for the DFS traversal.

3. dfs Helper Function:

   - A recursive function that performs the DFS traversal, checking for duplicates.

Logic

1. DFS Traversal:
    - The dfs function is defined to take a node and a set of visited values as arguments. The traversal starts from the root node and explores as far down a branch as possible before backtracking.
2. Base Case:
    - If the current node (node) is None, the function returns -1, indicating no duplicate has been found in this path.
3. Duplicate Check:
    - The function checks if the value of the current node (node.val) is already in the visited set. If it is, the function returns this value, as it is a duplicate.
    - If the value is not in the visited set, it is added to the set.
4. Recursive Calls:
    - The function recursively calls itself on the left child of the current node and stores the result in left_result.
    - If left_result is not -1, it indicates that a duplicate has been found in the left subtree, so the function returns this result immediately.
    - Otherwise, the function proceeds to recursively call itself on the right child of the current node and returns the result (right_result).
5. Set Initialization:
    - A set visited is initialized to keep track of the values of the nodes that have been visited during the traversal.
6. Initial Call:
    - The dfs function is initially called with the root node and the empty visited set.

Why It Works

- Correctness:
    - The solution correctly identifies duplicates by keeping track of visited node values in a set, ensuring that any value seen more than once is detected.
    - By using DFS, the solution explores each node systematically, checking for duplicates in both left and right subtrees.

- Efficiency:
  - The use of a set for tracking visited values ensures that the check for duplicates is efficient (O(1) average time complexity for lookups and inserts).
- Handling Edge Cases:
  - The base case where the node is None ensures that the function handles leaf nodes and empty trees gracefully.
  - The recursive structure ensures that the solution works for any binary tree, regardless of its shape or size.

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

- Time Complexity:
  - $O(n)$ because each node is visited once and set operations are $O(1)$
- Space Complexity:
  - $O(n)$ due to the storage of the visited set and the potential depth of the recursion stack in the worst case.

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

- Strengths:
  - The solution correctly identifies duplicates by using a set to track visited values, ensuring that any value seen more than once is detected.
  - The algorithm is simple and easy to understand, using straightforward DFS recursion.
- Weaknesses:
  - For very large trees, the recursive approach might lead to stack overflow errors due to deep recursion. An iterative approach using a stack could mitigate this issue and make the solution more robust for larger inputs.
- Adjustments:
  - To address the recursion depth issue and improve robustness for larger trees, we can convert the DFS recursion to an iterative approach using an explicit stack. This way, we avoid potential stack overflow errors and still maintain the same time and space complexity.

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

Assignment 1 Process:

- For Assignment 1, the task was to solve a given coding problem from LeetCode. I was assigned the problem of finding the missing numbers in a list. My approach involved leveraging Python’s set operations to identify and return the missing numbers efficiently. I first determined the range by finding the maximum value in the list and then created a set of all numbers within this range. By calculating the difference between this set and the set of numbers from the list, I was able to identify the missing numbers. The solution was efficient with a linear time complexity, making it suitable for large input sizes. This assignment reinforced my understanding of set operations and their applications in solving algorithmic problems.

Assignment 2 Presentation and Review Experience:

- In Assignment 2, I collaborated with a partner to review and critique each other's solutions from Assignment 1. This experience was invaluable as it provided a taste of the code review process common in software development teams. I analyzed my partner's solution for detecting duplicates in a binary tree using Depth-First Search (DFS). By paraphrasing the problem, creating new examples, and explaining the solution’s time and space complexities, I deepened my understanding of tree traversal algorithms. Additionally, providing constructive feedback on the strengths and potential improvements of my partner's solution helped me dive deep into the solution and understanding of trees and tree search.

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_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.