assignment_2

October 16, 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 Parts:

- Part 1: Complete 1 of 3 questions
- Part 2: Review your partner's Assignment 1 submission
- Part 3: Perform code review of your partner's assignment 1 by answering the questions below
- Part 3: Reflect on Assignment 1 and Assignment 2

1.4 Part 1:

You will be assigned one of three problems based of your first name. Enter your first name, in all lower case, execute the code below, and that will tell you your assigned problem. Include the output as part of your submission (do not clear the output). The problems are based-off problems from Leetcode.

[1]:
```python
import hashlib

def hash_to_range(input_string: str) -> int:
    hash_object = hashlib.sha256(input_string.encode())
    hash_int = int(hash_object.hexdigest(), 16)
    return (hash_int % 3) + 1
input_string = "Andrew"
result = hash_to_range(input_string)
print(result)

from typing import List
import random
```

```
# 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 __repr__(self):
    tree_str = self._visualize()
    paths = self.get_paths()
    return f"{tree_str}\nPaths to leaves: {paths} \n\n\n"

  def _visualize(self, prefix="", is_left=True):
    result = ""
    if self.right:
      result += self.right._visualize(prefix + ("    " if is_left else "    "),␣
↪False)
    result += prefix + ("   " if is_left else "   ") + str(self.val) + "\n"
    if self.left:
      result += self.left._visualize(prefix + ("    " if is_left else "    "),␣
↪True)
    return result

  def add_left(self, node):
    self.left = node
    return self.left

  def add_right(self, node):
    self.right = node
    return self.right

  def search(self, val):
    if self.val == val:
      return self
    if self.left:
      found = self.left.search(val)
      if found:
        return found
    if self.right:
      found = self.right.search(val)
      if found:
        return found
    return None

  def get_paths(self):
    if self is None:
      return []
    if self.left is None and self.right is None:
      return [[self.val]]
    paths = []
    if self.left:
      for path in self.left.get_paths():
        paths.append([self.val] + path)
    if self.right:
      for path in self.right.get_paths():
        paths.append([self.val] + path)
    return paths

  def find_duplicates(self):
      """Return a list of duplicate values in the tree."""
      counts = {}

      def traverse(node):
        if not node:
          return
        counts[node.val] = counts.get(node.val, 0) + 1
        traverse(node.left)
        traverse(node.right)

      traverse(self)
      return [val for val, count in counts.items() if count > 1]
```

2

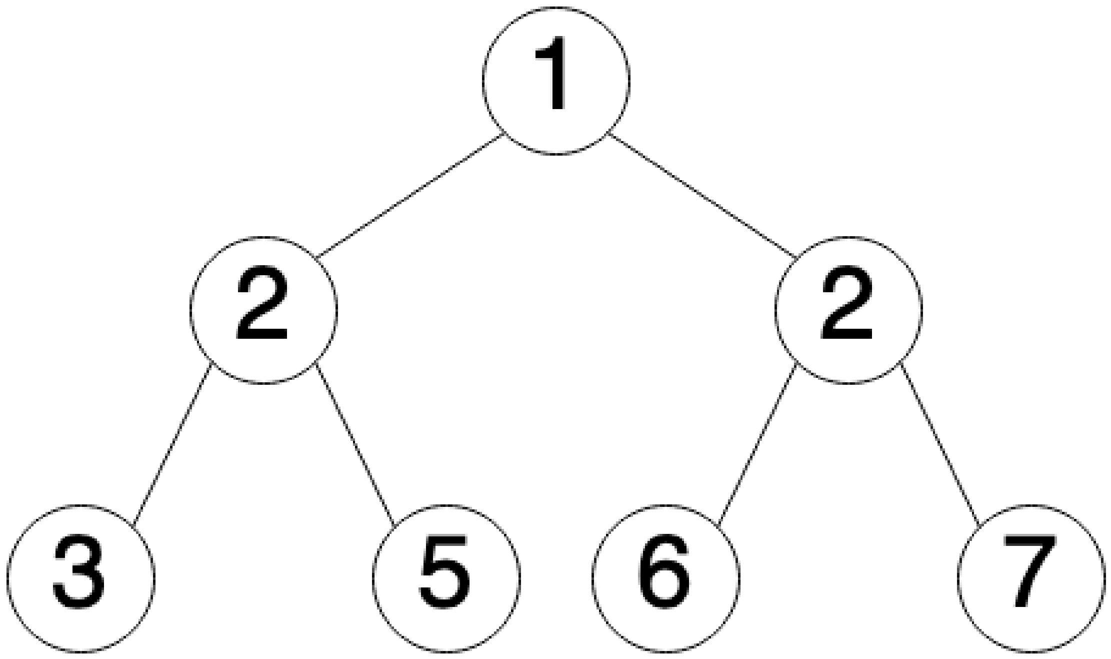
Question 1

2 Question One: Check Duplicates in Tree

Given the `root` of a binary tree, check whether it is contains a duplicate value. If a duplicate exists, return the duplicate value. If there are multiple duplicates, return the one with the closest distance to the root. If no duplicate exists, return -1.

2.1 Examples

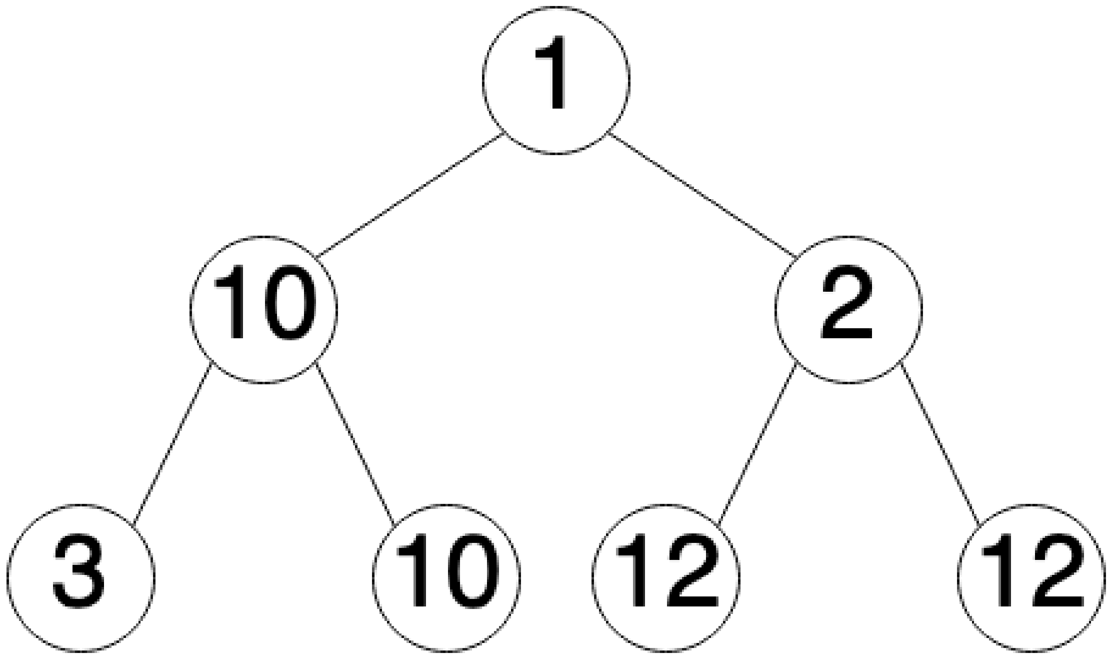
2.1.1 Example 1



Input: `root = [1, 2, 2, 3, 5, 6, 7]` *What traversal method is this?*

Output: 2

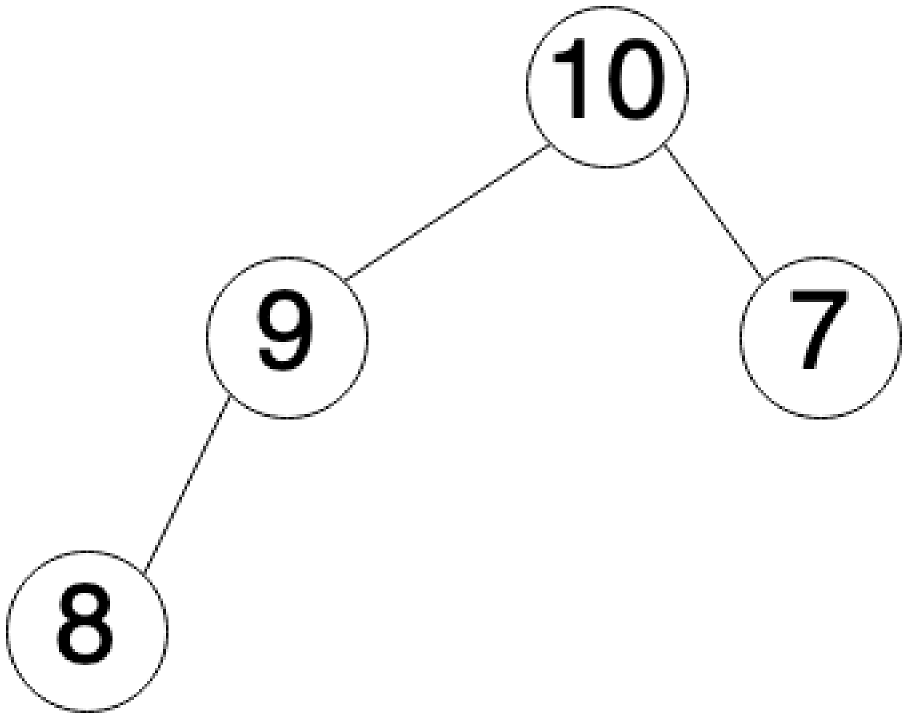
2.1.2 Example 2



Input: `root = [1, 10, 2, 3, 10, 12, 12]`

Output: 10

2.1.3 Example 3



Input: `root = [10, 9, 8, 7]`

Output: -1

Starter Code for Question 1

[2]:
```python
from collections import deque

def is_duplicate(root: TreeNode) -> int:
  duplicates = root.find_duplicates()
  if not duplicates:
    return -1
  # Find the duplicate closest to the root using BFS
  queue = deque([root])
  while queue:
    node = queue.popleft()
    if node.val in duplicates:
      return node.val
    if node.left:
      queue.append(node.left)
    if node.right:
      queue.append(node.right)
  return -1
```

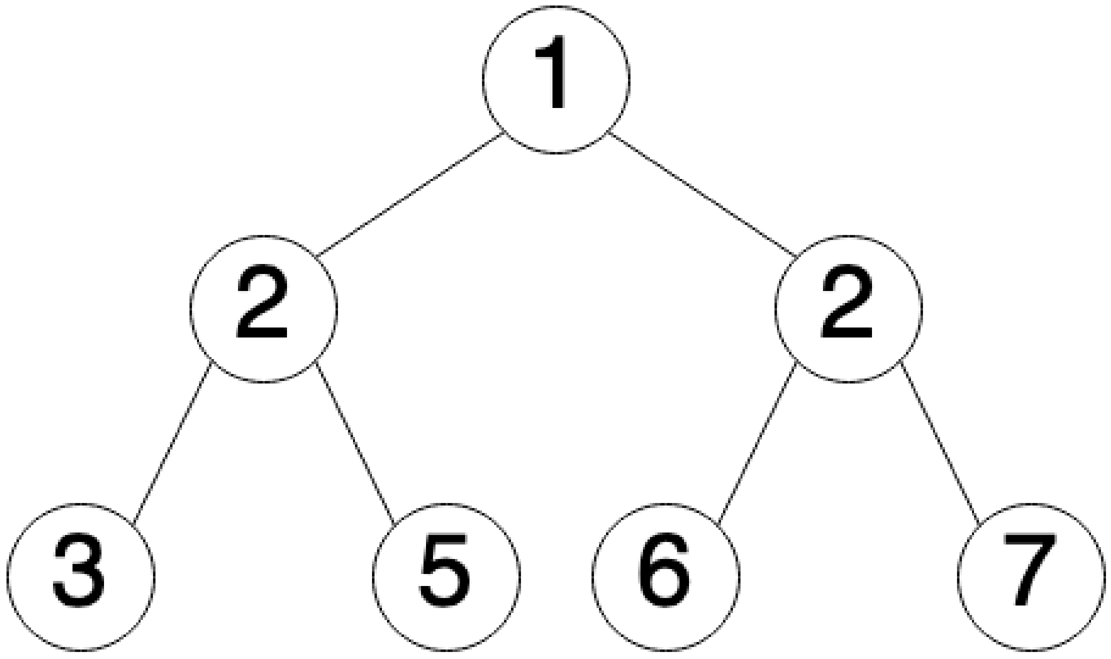
Question 2

3 Question Two: Path to Leaves

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

3.1 Examples

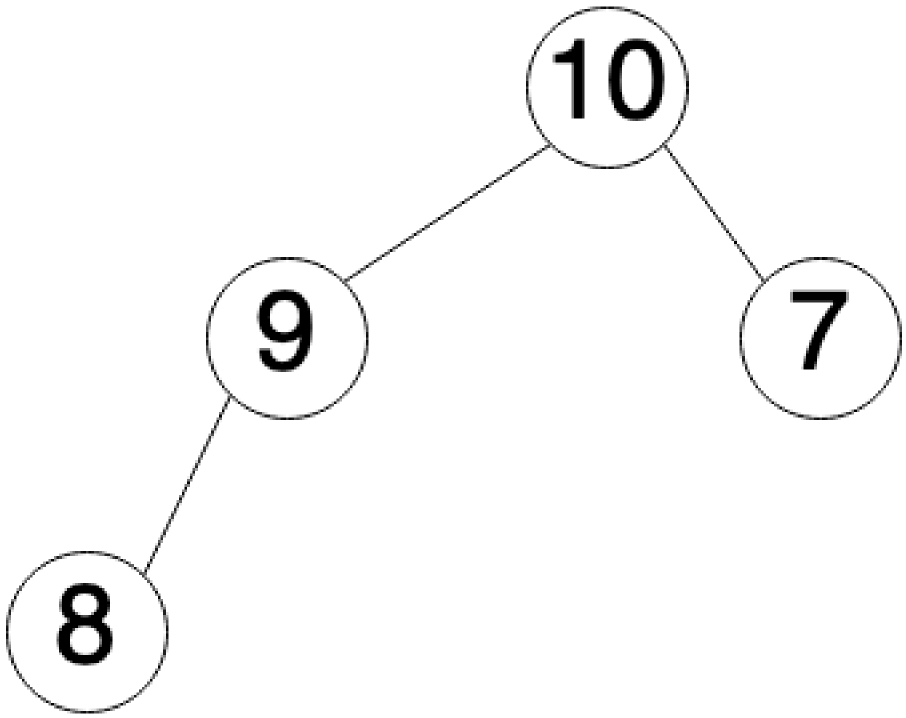
3.1.1 Example 1



Input: `root = [1, 2, 2, 3, 5, 6, 7]` *What traversal method is this?* Level Order or Breadth First Search

Output: [[1, 2, 3], [1, 2, 5], [1, 2, 6], [1, 2, 7]]

3.1.2 Example 2



Input: `root = [10, 9, 7, 8]`

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

Starter Code for Question 2

[3]:
```python
def bt_path(root: TreeNode) -> List[List[int]]:
  return root.get_paths()

def random_tree_node(min_val=0, max_val=10):
  return TreeNode(random.randint(min_val, max_val))

# Example 1: Balanced tree with random values
root1 = random_tree_node()
root1.add_left(random_tree_node()).add_left(random_tree_node())
root1.left.add_right(random_tree_node())
root1.add_right(random_tree_node()).add_left(random_tree_node())
root1.right.add_right(random_tree_node())

# Example 2: Tree with random values
root2 = random_tree_node()
root2.add_left(random_tree_node()).add_left(random_tree_node())
root2.left.add_right(random_tree_node())
root2.add_right(random_tree_node()).add_left(random_tree_node())
root2.right.add_right(random_tree_node())
```

```
# Example 3: Tree with random values
root3 = random_tree_node()
root3.add_left(random_tree_node()).add_left(random_tree_node())
root3.add_right(random_tree_node())

# Example 4: Single node tree with random value
root4 = random_tree_node()

# Example 5: Unbalanced tree with random values
root5 = random_tree_node()
root5.add_left(random_tree_node()).add_left(random_tree_node())
root5.left.add_right(random_tree_node())
root5.add_right(random_tree_node()).add_right(random_tree_node())

print("Example 1:\n", root1)
print("Example 2:\n", root2)
print("Example 3:\n", root3)
print("Example 4:\n", root4)
print("Example 5:\n", root5)
```

```
Example 1:
            4
       2
           10
   6
            10
        9
            3

Paths to leaves: [[6, 9, 3], [6, 9, 10], [6, 2, 10], [6, 2, 4]]



Example 2:
            8
       2
           5
   3
            6
        2
            0

Paths to leaves: [[3, 2, 0], [3, 2, 6], [3, 2, 5], [3, 2, 8]]



Example 3:
         1
```

```
  9
      4
          6

Paths to leaves: [[9, 4, 6], [9, 1]]



Example 4:
    5

Paths to leaves: [[5]]



Example 5:
           6
       0
   9
          6
       5
           3

Paths to leaves: [[9, 5, 3], [9, 5, 6], [9, 0, 6]]
```

Question 3

4 Question Three: Missing Number in Range

You are given a list containing `n` integers in the range `[0, n]`. Return a list of numbers that are missing from the range `[0, n]` of the array. If there is no missing number, return -1. Note, all the integers in the list may not be unique.

4.1 Examples

4.1.1 Example 1

Input: `lst = [0, 2]`

Output: [1]

4.1.2 Example 2

Input: `lst = [5, 0, 1]`

Output: [2, 3, 4]

4.1.3 Example 3

Input: `lst = [6, 8, 2, 3, 5, 7, 0, 1, 10]`

Output: [4, 9]

Starter Code for Question 3

```
[4]: def missing_num(nums: List) -> int:
       n = max(nums)
       missing = [i for i in range(min(nums), n + 1) if i not in nums]
       return missing if missing else -1

     # Test cases for missing_num function
     print(missing_num([0, 2]))              # Output: [1]
     print(missing_num([5, 0, 1]))           # Output: [2, 3, 4]
     print(missing_num([6, 8, 2, 3, 5, 7, 0, 1, 10]))  # Output: [4, 9]
     print(missing_num([0, 1, 2, 3]))        # Output: -1
     print(missing_num([3, 1, 4, 2, 0]))     # Output: -1
     print(missing_num([1]))                 # Output: -1
     print(missing_num([0]))                 # Output: -1
```

```
[1]
[2, 3, 4]
[4, 9]
-1
-1
-1
-1
```

4.2 Part 2:

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

4.3 Part 3:

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.

Given a list of integers, find and move all zeroes to the end of the list while maintaining the relative order of non-zero elements. Return the modified list.

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

```
[5]: input = [7, 4, 0, 3, 7, 3, 0, 2, 0, 4, 0] # output = [7, 4, 3, 7, 3, 2, 4, 0,␣
       ↪0, 0, 0]

     xList_input = [4, 0, 5, 0, 0, 6, 12 ,33, 56, 12, 99] # note the duplicate 12 #␣
       ↪output = [4, 5, 6, 12, 33, 56, 12, 99, 0, 0, 0]
```

When the variable xList_input is used as a parameter for the move_zeros_to_end function, it first goes through the list and records all the nonzero elements into a new list. It then finds out how many zeros exist in the list, and adds them to the end of the new list. Finally, it returns the resulting list. I fail to see how this question differs from the questions that follow.

- Copy the solution your partner wrote.

[6]:
```
def move_zeros_to_end(nums: List[int]) -> List[int]:
    #first step, collect the non-zeros in the list by iterating through the␣
 ↪original
    nonZeros = [number for number in nums if number != 0]
    #figure out how many zeros there should be by subracting the lengths of the␣
 ↪two lists
    numZeros = len(nums)- len(nonZeros)
    #now regenerate the original list by putting the correct number of zeros at␣
 ↪the end
    nums[:] = nonZeros + [0] * numZeros
    return nums
    #####
#####
```

- Explain why their solution works in your own words.

The function first finds all the non-zero elements in the input list and stores them in a new list. It then calculates the number of zeroes by subtracting the length of the non-zero list from the length of the original list. Finally, it appends the calculated number of zeroes to the end of the non-zero list and returns the modified list.

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

Worst case time complexity is O(n) because the list has to be traversed once to separate non-zero elements and count zeroes. The space complexity is also O(n) in the worst case, as a new list is created to store non-zero elements and then append the zeroes, so the output list is necessarily the same size as the input list.

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

It's good, it avoids the necessity to swap elements in place which can be more complex and error-prone. The approach is straightforward and easy to understand, making it a good solution for the problem at hand. I would have used a numpy array since we're dealing with integers and numpy is optimized for numerical operations, and sorting by vectorization is going to be faster on large lists. However, it's possible the need to keep the order of numbers that are not sequentially descending to zero would be too complicated for a numpy approach, I would have to spend some time with the documentation to see if it's feasible.

4.4 Part 4:

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.

4.4.1 Reflection

In assignment 1, I defined the is_valid_brackets function first with a map to define which pairs of brackets are valid. I used the values of the map to load a set of open brackets and the keys to load a set of closing brackets. I did a simple test to find out if, logically, the brackets can't be paired, if there are an odd number of brackets, the string starts with a closing bracket, or it ends with an opening bracket. I did this just to take care of a few really simple edge cases where I could make the order of the function O(1) instead of O(n). Then I defined a stack and iterated through the string, pushing opening brackets onto the stack, and popping them off when I found a closing bracket that matched the last opening bracket. If I found a closing bracket that didn't match, I returned false. At the end of the function, if the stack was empty, I returned true, otherwise false.

For the second assignment, I implemented a BST from scratch. I moved the definition to the first code cell of this document so that some of the functionality required could be implemented as class methods rather than procedural code. I implemented a repr method that includes both the paths to leaves of the tree and a display of the tree itself. To test the insertion, duplicate detection, and path finding methods, I created a random tree generator function that creates a binary tree of random integers. I created three different trees to test the functionality, one balanced, one unbalanced, and one with duplicates. I printed the trees and the results of the duplicate detection and path finding methods to verify that they were working correctly. I thought about writing rotations to balance the tree, but I think that would be a bit beyond the scope of this assignment.

4.5 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

4.6 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.

4.6.1 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-6-help`. Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.

[ ]:
```

```