Reflection on Assignment 1: Missing Number(s) in Range

Working on Assignment 1 allowed me to explore multiple approaches to the "Missing Number(s) in Range" problem and refine my problem-solving process.

I began with Approach 1 (Set Difference), which was intuitive and straightforward. Using Python's set operations, I compared a complete range to the input. While it was clean and easy to implement, initial tests revealed mistakes, particularly with edge cases. Although fixing these helped deepen my understanding, I realized this approach becomes inefficient for large ranges due to its O(max_value + n + k log k) time complexity.

Next, I tackled Approach 2 (Boolean Array) to optimize performance. This approach marked seen numbers in a boolean array and identified missing values through a simple iteration. With a time complexity of O(max_value + n), this was the most efficient solution, particularly for large datasets. This process reinforced the importance of selecting the right data structure for performance gains.

Finally, I developed Approach 3 (Sort + Two Pointers) to explain an alternative. Sorting the input and comparing it to the range allowed me to explicitly handle duplicates, making it a practical middle-ground solution. This assignment enhanced my understanding of algorithms, trade-offs, and adapting solutions to meet different requirements.

Reflection on Assignment 2: Path to Leaves

Collaborating with my partner on this solution was a constructive experience. Their implementation effectively used Breadth-First Search (BFS) to systematically traverse the binary tree and capture all root-to-leaf paths. Their explanation of BFS mechanics and its iterative approach was clear and insightful. Additionally, their emphasis on correctness and efficient queue management provided a solid foundation for understanding the solution's strengths.

However, through reviewing the solution, I identified a few areas for optimization. The original code's path construction introduced overhead, especially for deep or unbalanced trees. Adjusting this to construct paths once per node and reusing them improved efficiency. A minor structural cleanup in queue initialization also made path handling more consistent and readable.

The weaknesses of the solution, such as space usage and scalability challenges for large or deep trees, were inherent to BFS but are worth noting. Including these considerations highlighted the importance of choosing the right algorithm based on the tree's characteristics. Furthermore, documenting edge cases and refining explanations strengthened the solution's presentation.

Overall, this collaboration emphasized the value of iterative feedback and showcased how constructive critique and small adjustments can enhance both the solution's clarity and performance.