Find Binary Tree Paths LeetCode

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

Example:

```
              5
           /    \
         3        7
       /    \       \
     11       1       6
  /    \        \
9     12         15
```

Output: The Binary Tree Paths: [[5→3→11→9], [5→3→11→12], [5→3→1→15], [5→7→6]]

Approach 1: Function to find binary tree paths using a recursive approach

- The recursive approach starts at the root node and explores each branch of the tree individually.
- It maintains a temporary string to represent the current path.
- As it traverses the tree recursively, it appends each node's value to the path.
- When it reaches a leaf node, it adds the complete path to the answer.
- The paths are stored in a vector.

Time Complexity:

- **The time complexity is O(N), where N is the number of nodes in the tree, as each node is visited once.**

Space Complexity:

- **The space complexity is O(H), where H is the height of the tree, as each recursive call consumes space on the call stack.**

Approach 2: Function to find binary tree paths using an iterative approach

- The iterative approach uses a level-order traversal, starting from the root node.
- It maintains a queue to keep track of nodes and a corresponding string to store the path from the root to each node.

- As it dequeues nodes, it appends their values to their respective paths.
- When it reaches a leaf node, it adds the path to the answer.
- The paths are stored in a vector.

Time Complexity:

- **The time complexity is O(N), where N is the number of nodes in the tree, as each node is visited once during the traversal.**

Space Complexity:

- **The space complexity is O(N) in the worst case because all paths are stored in memory.**

Conclusion:

Both the recursive and iterative approaches effectively find all paths from the root to leaf nodes in a binary tree. They share the same time complexity of O(N), ensuring efficient traversal of all nodes. However, their space complexities differ. The recursive approach has a space complexity of O(H), making it memory-efficient, while the iterative approach consumes O(N) space since it stores all paths in memory.