# 298. Binary Tree Longest Consecutive Sequence



## Problem Description

The problem specifies a binary tree and requires us to find the length of the longest consecutive sequence path within it. A consecutive sequence path is defined as a path in which adjacent nodes in the path have values that differ by exactly one. An important aspect of this problem is that the path does not need to begin at the root of the tree; it can start at any node. Additionally, the path is uni-directional, meaning once you move from a parent node to a child node, you cannot move back up to the parent.

## Intuition

To solve this problem, we can use a Depth-First Search (DFS) approach. The idea is to explore each node starting from the root and recursively check its children to find the longest increasing consecutive sequence that includes the node itself as either the start or some part of the sequence. The DFS approach allows us to go as deep as possible down each path before backtracking, which is necessary to examine all potential paths.

Specifically, at each node, we will perform the following actions:

1. We check if the node is `None`. If it is, there's no sequence to be found, so we return `0`.
2. We recursively call the `dfs` function on the left and right children. The returned values represent the lengths of the consecutive sequences that start with the children.
3. We increment the lengths by one to include the current node. At this point, the length assumes that the current node is part of the sequence.
4. We need to check if the child node actually follows the consecutive sequence. We do this by ensuring that the difference between the value of the child node and the current node is exactly one. If not, we reset the length to `1` as the sequence can only start from the current node.
5. We calculate the temporary longest sequence `t` that can be created by the current node with its children by taking the maximum of the lengths from the left and right children.
6. We maintain a global variable `ans` that keeps track of the longest sequence found so far. After processing each node, we update `ans` with the maximum value between `ans` and `t`.

In the end, we return `ans`, which will contain the maximum length of the longest consecutive sequence path found anywhere in the binary tree.

## Solution Approach

To implement the solution approach for finding the longest consecutive sequence path in a binary tree, the Depth-First Search (DFS) algorithm is used, utilizing recursion. The code defines a helper function `dfs` within the `Solution` class's `longestConsecutive` method to perform the DFS.

Here's how the implementation works step by step:

1. The `dfs` function is defined to take a `TreeNode` as an input and returns an integer which is the length of the longest consecutive path that can be formed including the given node. If the node is `None`, the function returns `0`, indicating that a null node cannot contribute to the length of a sequence.
2. Invoking `dfs` recursively, we analyze the left and right subtrees by calling `dfs(root.left)` and `dfs(root.right)`. The returned values from both calls represent the lengths of consecutive sequence paths starting at the left and right children, respectively.
3. A value of `1` is added to both lengths to include the current `root` in the sequence. This is based on the assumption that the current `root` extends the paths from its children by one.
4. Two `if` conditions ensure that the sequence is indeed consecutive. If the left child's value is not one more than the current node's value or if the right child's value is not one more than the current node's value, the corresponding sequence length is reset to `1`, indicating that a new sequence can start at `root`.
5. The temporary maximum length `t` is determined by taking the maximum of the adjusted values from the left and right sequences.
6. The non-local variable `ans` is used to keep track of the global maximum. This variable is updated (if necessary) every time a new local maximum is found, ensuring it always holds the length of the longest consecutive sequence seen so far.
7. After updating `ans`, `t` is returned, referring to the longest sequence that includes the current `root`.

The recursive DFS strategy allows each node to contribute to the sequential path in a bottom-up approach, combining individual paths from leaf nodes to the root (or any other starting node), updating the global maximum along the way.

In the main function `longestConsecutive`, `ans` is initialized to 0, then the recursive `dfs` process is kickstarted by passing the `root` node. After exploring all possible nodes, the final `ans` represents the solution and is returned.

### Example Walkthrough

Let's assume we have the following binary tree structure to help illustrate the solution approach:

```
    1
   / \
  3   2
 / \
4   2
   /
  3
```

We need to find the length of the longest consecutive sequence path in this tree.

Here's a step-by-step walkthrough with our sample tree:

1. Start with the root node (1). The `dfs` function is called with node `1` as input.
2. Call `dfs` recursively for the left child (`dfs(2)`) and right child (`dfs(3)`).
3. In `dfs(2)`, it recursive calls `dfs(4)`.
   - `dfs(4)` returns `1` because it is a leaf node and has no children.
   - Since `4` is one more than `2`, we increase the sequence length to `2` (initial `1`, plus `1` since it's consecutive).
   - This sequence length is returned to the `dfs` call on node `2`.
4. For the right child `dfs(3)` of root node `1`:
   - `dfs(3)` calls `dfs(2)`, which then calls `dfs(3)` on its own left child.
   - The left child `3` is a leaf and returns `1`.
   - The node `2` has a child with a value of one less than itself, so this is not an incrementing consecutive sequence.
   - `dfs(2)` resets the sequence length to `1`.
5. Now we compare the temporary lengths from left and right children of the root. For the left side, we got a length of `2` (from `1->2->4` sequence), and from the right side, we have a length of `1` (since the `1->3->2` sequence isn't increasing consecutively).
6. The temporary maximum `t` at root node `1` is `2` (the maximum of `2` and `1`). However, root `1` does not increment either sequence, so we do not add `1` to it.
7. We update the global maximum `ans` if `t` is greater than the current `ans`. For the root node, `ans` is updated to `2`.
8. `dfs(1)` returns `t`, which is `2`.

Hence after running our `dfs` algorithm on the binary tree, we find that the longest consecutive sequence path has a length of `2` from the sequence `1->2->4`.

## Python Solution

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

class Solution:
    def longestConsecutive(self, root: Optional[TreeNode]) -> int:
        # Helper function that uses depth-first search to find the longest consecutive sequence.
        def dfs(node: Optional[TreeNode]) -> int:
            # If node is None, return 0
            if not node:
                return 0

            # Recursively find the longest consecutive sequence in the left and right subtrees.
            left_length = dfs(node.left) + 1  # Length of the sequence from the left child
            right_length = dfs(node.right) + 1  # Length of the sequence from the right child

            # If the left child is not consecutive, reset the length for that path
            if node.left and node.left.val - node.val != 1:
                left_length = 1

            # If the right child is not consecutive, reset the length for that path
            if node.right and node.right.val - node.val != 1:
                right_length = 1

            # The longest path through the current node is the larger of its two children's paths
            current_max_length = max(left_length, right_length)

            # Update the global answer if the current path is longer than the previously found paths
            nonlocal max_length
            max_length = max(max_length, current_max_length)

            # Return the length of the longest path passing through this node
            return current_max_length

        max_length = 0  # Initialize the maximum length variable
        dfs(root)  # Start DFS from the root of the tree
        return max_length  # Return the longest consecutive sequence length found
```

## Java Solution

```java
/**
 * Definition for a binary tree node.
 */
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode() {}
    TreeNode(int val) { this.val = val; }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}

class Solution {
    private int longestStreak;

    /**
     * Finds the length of the longest consecutive sequence path in a binary tree.
     * @param root The root of the binary tree.
     * @return The length of the longest consecutive sequence.
     */
    public int longestConsecutive(TreeNode root) {
        dfs(root);
        return longestStreak;
    }

    /**
     * Depth-first search helper method that traverses the tree and finds the longest consecutive path.
     * The returned value is used by the parent node to check if it can form a consecutive sequence.
     * @param node The current node in the DFS traversal.
     * @return The length of the longest path ending at the current node.
     */
    private int dfs(TreeNode node) {
        if (node == null) {
            return 0;
        }

        int leftLength = dfs(node.left) + 1; // Length of left child sequence + current node
        int rightLength = dfs(node.right) + 1; // Length of right child sequence + current node

        // If the left child's value is not consecutive, reset left sequence length
        if (node.left != null && node.left.val - node.val != 1) {
            leftLength = 1;
        }

        // If the right child's value is not consecutive, reset right sequence length
        if (node.right != null && node.right.val - node.val != 1) {
            rightLength = 1;
        }

        // The maximum length of the consecutive sequence at the current node
        int currentMax = Math.max(leftLength, rightLength);

        // Update the longest streak if the current max is greater
        longestStreak = Math.max(longestStreak, currentMax);

        return currentMax;
    }
}
```

## C++ Solution

```cpp
#include <functional> // We should include the necessary header for std::function

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
};

class Solution {
public:
    // Function to find the length of the longest consecutive sequence in a binary tree.
    int longestConsecutive(TreeNode *root) {
        int maxLength = 0; // Initialize the maximum length to keep track of the longest sequence.

        // Recursive lambda function to perform a depth-first search on the tree.
        // It returns the length of the longest consecutive sequence starting at 'node'.
        std::function<int(TreeNode*, int)> dfs = [&](TreeNode* node) {
            if (!node) {
                // If the node is null, return 0 since there is no sequence.
                return 0;
            }
            // Recursively find the length of the longest consecutive sequences in the left and right subtrees.
            // We start by assuming both sequences include the current node, hence +1.
            int leftLength = dfs(node->left) + 1;
            int rightLength = dfs(node->right) + 1;

            // If the left child exists and doesn't follow the consecutive property, reset the leftLength.
            if (node->left && node->left->val - node->val != 1) {
                leftLength = 1;
            }

            // If the right child exists and doesn't follow the consecutive property, reset the rightLength.
            if (node->right && node->right->val - node->val != 1) {
                rightLength = 1;
            }

            // The longest sequence at the current node is the max of the left and right sequences.
            int currentLength = std::max(leftLength, rightLength);

            // Update the maxLength if the sequence at the current node is the longest one found so far.
            maxLength = std::max(maxLength, currentLength);

            // Return the length of the longest sequence at the current node.
            return currentLength;
        };

        // Perform the depth-first search starting from the root.
        dfs(root);

        // After the DFS, maxLength contains the length of the longest consecutive sequence in the tree.
        return maxLength;
    }
};
```

## Typescript Solution

```typescript
// Type definition for a binary tree node.
class TreeNode {
    val: number;
    left: TreeNode | null;
    right: TreeNode | null;
}

// Function that returns the length of the longest consecutive sequence in the binary tree.
function longestConsecutive(root: TreeNode | null): number {
    let longestSequence = 0; // This variable will hold the longest sequence found.

    // Helper function that performs depth-first search to find the longest consecutive sequence.
    const findLongestConsecutive = (currentNode: TreeNode | null): number => {
        if (currentNode === null) {
            return 0;
        }

        // Recursively find the lengths for left and right subtrees.
        let leftLength = findLongestConsecutive(currentNode.left) + 1;
        let rightLength = findLongestConsecutive(currentNode.right) + 1;

        // If the left child is not consecutively increasing, reset the left sequence length.
        if (currentNode.left && currentNode.left.val - currentNode.val !== 1) {
            leftLength = 1;
        }

        // If the right child is not consecutively increasing, reset the right sequence length.
        if (currentNode.right && currentNode.right.val - currentNode.val !== 1) {
            rightLength = 1;
        }

        // Take the longer sequence of the two.
        const currentMax = Math.max(leftLength, rightLength);

        // Update the longest sequence found so far.
        longestSequence = Math.max(longestSequence, currentMax);

        // Return the longer sequence found at the current node.
        return currentMax;
    };

    // Start the search from the root of the binary tree.
    findLongestConsecutive(root);

    // Return the longest consecutive sequence length found.
    return longestSequence;
}
```

## Time and Space Complexity

The given code performs a depth-first search (DFS) on a binary tree to find the length of the longest consecutive sequence.

### Time Complexity:

The time complexity of the DFS function is $O(N)$, where $N$ is the total number of nodes in the binary tree. This is because the algorithm visits each node exactly once to calculate the longest consecutive path that includes that node as the sequence start.

### Space Complexity:

The space complexity of the DFS function is $O(H)$, where $H$ is the height of the binary tree. The reason for this is the recursive stack space which, in the worst-case scenario (a completely unbalanced tree), could be equal to the number of nodes $N$ (thus $O(N)$), but for a balanced tree, it will be $O(\log N)$. $H$ captures this as it ranges from $\log N$ to $N$ depending on the balance of the tree.