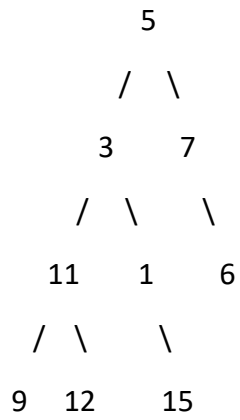
The sum of Deepest Leaves of Binary Tree LeetCode

Given the root of a binary tree, return *the sum of values of its deepest leaves*.

Example:



Output: The Sum of Deepest Leaves: 36

Approach 1: Function to calculate the sum of deepest leaves using recursive approach

- The recursive approach first calculates the height of the binary tree, which is the maximum depth.
- It then calculates the sum of nodes at the deepest level.
- The recursion starts at the root, and at each level, it checks if the current level is equal to the maximum depth. If so, it adds the value of the node to the sum.
- The process continues recursively for left and right subtrees.
- The sum of values at the deepest level is returned as the result.

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, due to the recursive call stack.**

Approach 2: Function to calculate the sum of deepest leaves using optimized recursive approach

- This optimized recursive approach eliminates the need to calculate the height separately.

- It maintains two variables: **sum** to store the sum of values at the deepest level and **maxHeight** to keep track of the maximum height seen during traversal.
- As the recursion progresses, it checks if the current node is at a greater height than **maxHeight**. If so, it resets **sum** and updates **maxHeight**.
- When the current node is at the deepest level (height equals **maxHeight**), its value is added to **sum**.
- The process continues recursively for left and right subtrees.
- The final **sum** represents the sum of values at the deepest leaves.

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 traversal.**

Space Complexity:

- **The space complexity is O(H), where H is the height of the tree, due to the recursive call stack.**

Approach 3: Function to calculate the sum of deepest leaves using iterative approach

- The iterative approach utilizes a level-order traversal using a queue.
- It starts at the root and iteratively processes nodes at each level.
- At each level, it calculates the sum of values of all nodes at that level.
- The process continues until all levels have been traversed.
- The final sum represents the sum of values at the deepest leaves.

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 traversal.**

Space Complexity:

- **The space complexity is O(N) in the worst case due to the queue storing all nodes at a level.**

Conclusion:

All three approaches effectively calculate the sum of values of the deepest leaves 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).
- The optimized recursive approach also has a space complexity of O(H).
- The iterative approach has a space complexity of O(N).

In terms of memory efficiency, the recursive approach and the optimized recursive approach are preferable when memory is a concern, as they have lower space complexities. The iterative approach, while straightforward, consumes more memory as it stores all nodes at each level. Therefore, the recursive or optimized recursive approach may be better choices, depending on your memory constraints.