Documentation

The problem involves finding the lowest common ancestor (LCA) of two given nodes in a Binary Search Tree (BST). A BST is a specialized binary tree structure in which every node satisfies a specific ordering property: all nodes in the left subtree have values less than the node's value, and all nodes in the right subtree have values greater than the node's value. This ordering property is central to solving the problem efficiently because it provides a clear and structured way to navigate the tree in search of the LCA, which is defined as the deepest node in the tree that has both of the target nodes as descendants (with the possibility that the LCA itself is one of the target nodes).

To address this problem, we will start the search at the root of the BST. From there, we assess whether the values of both target nodes are less than the current node's value or greater than it. This helps us decide in which direction to proceed: if both target nodes have values less than the current node's value, we know they must lie in the left subtree, so we move left; similarly, if both target nodes' values are greater, we proceed to the right subtree. This decision-making process uses the BST's ordering properties to reduce unnecessary comparisons and directs us toward the lowest ancestor efficiently without traversing the entire tree.

The traversal continues until we reach a node where the values of the target nodes diverge, meaning one target node's value is less than the current node's value and the other's is greater. This divergence indicates that we have found the lowest node where the paths to the two target nodes split, identifying it as the lowest common ancestor. Additionally, if we encounter a node whose value matches one of the target nodes, this node is also the LCA. This is because, in a BST, each node is a descendant of itself, so if the current node's value is equal to one of the targets and lies between the two target values, it confirms that it serves as their ancestor.

The advantage of this approach is its efficiency. Instead of exploring every node, which would be time-consuming for larger trees, we utilize the tree's inherent structure to limit our search path. Consequently, this solution operates with a time complexity of (O(h)), where (h) is the height of the tree. In balanced BSTs, (h) approximates (log(n)), where (n) is the number of nodes, making this method very efficient. Moreover, as the solution relies on modifying pointers rather than using additional data structures, its space complexity is (O(1)), ensuring it uses minimal memory.

Finally, the problem constraints guarantee that the BST is valid and that both target nodes are distinct and exist within the tree. This assurance allows us to proceed without additional checks for node presence, which simplifies the implementation. By using the BST's properties effectively, this solution can provide an optimal path to find the LCA, accommodating large trees with minimal computation and ensuring that the solution is both time-efficient and space-efficient. This method leverages the fundamental properties of the BST, making it a canonical example of problem-solving through efficient tree traversal and node comparison techniques.