Zakeriya Muhumed
Program #4 Writeup
CS163

A potent data structure, binary search trees enable effective data searching, insertion, and deletion. It is crucial to comprehend the fundamental ideas and principles that underlie binary search trees in order to completely comprehend and utilize them.

A binary search tree is fundamentally a hierarchical data structure that is set up to facilitate effective data searching. Each node in the tree has a value and two pointers pointing to other nodes, one of which points to a left subtree and the other to a right subtree. While the right child's value exceeds that of its parent node, the left child's value is less than that of its parent node. The tree can be easily navigated and the data can be searched effectively thanks to its organization.

In order to add information to a binary search tree, you must navigate the tree in search of the ideal spot to add the new node. To do this, a new node's value is compared to the value of the current node, and based on the comparison, a left or right move is made. The new node is added to the tree as a leaf node once the proper placement has been determined.

It's a little trickier to delete data from a binary search tree than it is to insert it. Finding the node in the tree that holds the data that has to be erased is the first step. Upon the discovery of the node, there are three scenarios to take into account: There are no children of the deleted node. In this situation, it is simple to remove the node from the tree. There is only one child of the removed node: To replace the lost node in this scenario, the child node can be promoted.The node that has to be eliminated has two kids: The most challenging scenario is the one where the values of the node to be deleted and the node with the next-highest value in the tree (i.e., the node with the smallest value in the right subtree) must be switched. One of the first two scenarios can then be used to delete the node.

The effectiveness of binary search trees in finding data is one of its main advantages. Data searching can be completed in O(log n) time due to the tree's easy traversal due to its organization. Searching over an unsorted list or array would take O(n) time, which is much slower.

The ability of binary search trees to handle vast volumes of data is another advantage. The performance of the tree is unaffected by the size of the data set as long as it stays balanced (that is, the left and right subtrees are around the same size).The vulnerability of binary search trees to imbalance is one potential downside, though. The performance of the tree can suffer greatly if the data being added is not balanced, i.e., one side of the tree has noticeably more nodes than the other. The AVL tree and the red-black tree are two balancing methods that have been created to address this problem.

Working with binary search trees might be difficult, but it can also be rewarding. It takes a firm grasp of algorithms and data structures, as well as expertise in programming languages, to comprehend the principles underlying the data structure and implement it in a program. The end product, though, is a strong tool for quickly searching through and altering data, with a variety of possible real-world applications.