

D2-1

Title :- Implementation of a Dictionary using Binary Search Tree for sorting and searching.

Problem Statement :- A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data stored in ascending / Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.

Objectives :- To implement a Dictionary using binary search tree for sorting and searching.

Software Requirement :-

Theory :- A Binary Search Tree (BST) is a binary tree data structure where each node has at most two children, and the values of the left child are less than the parent, and the values of the right child are greater than the parent. The BST property allows for efficient searching, inserting and deleting of nodes.

In the case of a dictionary implementation, the keywords would be stored as the key values, and their corresponding meanings as the data values.

To implement the required functionalities, we can use the following operations.

1. Adding a new keywords : To add a new keyword we start at the root of the tree and compare the keyword with the value of the current node. If the keyword is less than the value of the node, we move to the left child. If the keyword is greater move to right child. If we reach a null child node, we insert the new keyword and its meaning at that position.
2. Deleting keywords.
3. Updating values of any entry.
4. Displaying whole data stored in ascending / descending order.
5. Finding the maximum comparisons required for finding any keyword.

- The maximum comparisons required for finding any keyword in a BST is equal to the height of the tree. A balanced BST has a height of $\log(n)$, where n is the number of nodes in the tree.

There are different types of BSTs that can be used depending on the specific requirements of the application. Some of the common types include AVL trees, Red-black trees and Splay trees.

02-2

Algorithm :-

1. Start
2. Define an empty tree.
3. Define Function $\text{insert}()$, to add a new keyword.
4. Define Function $\text{delete}()$, to deleting a keyword.
5. Define a Function $\text{update}()$, to updating the value of a keyword.
6. Define Function as $\text{inorderTraversal}()$, to displaying the data in ascending order.
7. Define the Function as $\text{maxComparisons}()$, for finding the maximum comparisons required for finding any keyword.
8. Stop.

In general, BSTs are efficient for searching and inserting elements in a sorted collection. However, the worst-case time complexity of $O(n)$ can occur when the BST is unbalanced leading to degraded performances. To avoid this, it is important to balance the BST after every insertion and deletion operation.

Conclusion :- we can use a BST to implement a dictionary with efficient add, delete and update operations and the ability to display the data in sorted order. The maximum comparisons required for finding any keyword is equal to the height of the tree, which is $\log(n)$ for the balanced BST.