02

Date:- 1 1
Page No.:- 28

Title :- Implementation of a Dictionary using Binary Search Tree ~~of~~ 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 comparisions 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 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 right child
If we reach a null child node we insert that new keyword and its meaning at that position.
2. Deleting Keyword.
3. Updating values of any entry
4. Displaying whole day 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 diffrent types of BSTs that can be used depending on the specific requirements of the applications Some of the common types include AVL trees, Red-black trees and splay trees.

Date:- / /
Page No.:- 30

Algorithm :-

1. start
2. Define an empty tree
3. Define function insert (), to add a new keyword
4. Define function delete (), to deleting a keyword
5. Define function update () to updating the value of a keyword.
6. Define function as Inorder Traversal () to displaying the data in ascending order
7. Define the function as max comparisions (), for finding the maximum comparisions required for find any keyword
8. stop

In general, BSTs are efficient for searching and inserting element 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 comparisions required for finding any keyword is equal to the height of the tree, which is log(n) for the balanced BST