Practical Assignment no : 9.

Title : Implementation of a dictionary using binary search tree for sorting & searching.

Problem statement : A dictionary store keywords & its meanings provide facility for adding new keywords updating values of any entry 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 & searching.

software Requirements :

Theory : A binary search tree (BST) is a binary tree data structure where each node has at most two children & the values of left child are less than the parent & the value of the right child are greter than present & values of The BST property allows for efficient searching, inserting & deleting of nodes.

In the case of dictionary, implementation the keywords would be stored as the key; values and their corresponding meaning 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 tree and compare the keyword with value of the current. If the keyword is less than the value of the node we move to the left child. If the keyword is greater move for right child. If we reach a null child node.

we insert the new keyword & its meaning at that
2) Deleting keywords.
3) Updating values of any entry
4) Displaying whole data stored in ascending/descending
5) finding the maximum comparisons required for finding any keyword.
- The maximum comparisions required for finding any key in a BST is equal to the height of the tree. A balanced has a height of log(n) where n is the number of node the tree.
- There are different types of BST's that can be used depending on the specific requirements of the application some of the common types include AVL Trees. Red Black trees & splay tree to implement the required functionalities we can use the following operations.
1) Adding a new keywords: To add a new keyword we ~~want~~ start at the root of tree and compare the key with the value of the current node. If the keyword is less than the value of the node we have 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.
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 a function update() to updating the value of a keyword.
6) Define functions as inorder transversal () to displaying the data in ascending order.

DATE

7) Define the function as max comparisions() for finding the maximum comparisions required for finding any keyword.

8) stop.

In General BSTs are efficient for search & inserting element in a sorted collection. However the worst case time complexity of o(n) can occur when the BST is unbalance leading to degraded performance to avoid this, it is important to balance the BST after every insertion & deletion operation.

Conclusion :

We can use a BST to implement a dictionary with efficient add, delete & update operations & 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.