MIT World Peace University
Advanced Data Structures

Assignment 3

Naman Soni Roll No. 10

Contents

1 Problem Statement

Implement dictionary using binary search tree where dictionary stores keywords and its meanings.
Perform following operations:

- Insert a keyword
- Delete a keyword
- Create mirror image and display level wise
- Copy

2 Objectives

- To study data structure : Binary Search Tree
- To study breadth first traversal.
- To study different operations on Binary search Tree.

3 Theory

3.1 *Binary Search Tree*

A binary search tree is a type of data structure used to store a collection of elements in a sorted order. It is called a "binary" tree because each node in the tree has at most two children: a left child and a right child.

The structure of the tree is such that the value of any node in the left subtree is less than the value of its parent node, and the value of any node in the right subtree is greater than the value of its parent node. This property makes it possible to search for an element in the tree efficiently, by repeatedly comparing the target value with the value of the current node and then traversing either the left or right subtree accordingly.

Binary search trees can be used for a variety of operations, including insertion, deletion, and searching for elements. They are commonly used in computer science for implementing algorithms like binary search and quicksort. However, if the tree is not balanced (i.e. if one subtree is much deeper than the other), the performance of these operations can degrade and lead to inefficiencies.

3.2 *Breadth First Traversal*

Breadth First Traversal, also known as level order traversal, is a tree traversal algorithm that visits all the nodes of a tree in a breadth-first manner, i.e. it visits all the nodes at each level before moving on to the next level.

Starting from the root node, the algorithm visits all the nodes at level 1 (i.e. the immediate children of the root node) from left to right, then visits all the nodes at level 2 (i.e. the children of the nodes at level 1) from left to right, and so on until all the nodes in the tree have been visited.

The algorithm uses a queue data structure to keep track of the nodes to be visited, and a visited set to keep track of the nodes that have already been visited. Initially, the root node is added to the queue and marked as visited. Then, while the queue is not empty, the algorithm dequeues the next node in the queue, visits it, and enqueues all its unvisited children (if any). The algorithm continues until the queue is empty, at which point all nodes in the tree will have been visited in breadth-first order.

Breadth First Traversal has a time complexity of O(n), where n is the number of nodes in the tree, as it visits each node exactly once. It is commonly used in graph algorithms, such as finding the shortest path between two nodes in an unweighted graph.

3.3 *Different operations on binary search tree.(copy ,mirror image and delete)*

Here are brief descriptions of different operations on binary search trees:

1. Copy: To copy a binary search tree, we need to create a new tree and copy all the nodes of the original tree into the new tree. We can use either recursive or iterative approach for this operation. In the recursive approach, we start from the root node of the original tree and copy it into the new tree. Then, we recursively copy the left and right subtrees of the root node. In the iterative approach, we use a stack to traverse the original tree and create a new node for each visited node.

2. Mirror Image: To create a mirror image of a binary search tree, we need to swap the left and right subtrees of each node in the tree. This can be done recursively by traversing the tree and swapping the subtrees of each node.

3. Delete: To delete a node from a binary search tree, we first need to find the node to be deleted. We then need to handle three cases:

   - If the node has no children, we simply remove it from the tree.
   - If the node has one child, we replace it with its child.
   - If the node has two children, we find the node with the smallest value in its right subtree (or the node with the largest value in its left subtree), copy its value to the node to be deleted, and then delete the node with the smallest value (or largest value).

   We can use a recursive or iterative approach for this operation. In the recursive approach, we start from the root node and recursively search for the node to be deleted. In the iterative approach, we use a loop to traverse the tree and find the node to be deleted.

4 Implementation

4.1 *Platform*

Visual Studio Code
Mac OS 64 bit

5 Test Conditiions

- Input at least 10 nodes.
- Display binary search tree levelwise traversals of binary search tree with 10 nodes
- Display mirror image and copy operations on BST

6 Conclusion

Thus, implemented Dictionary using Binary search tree.

7 FAQ's

(a) Explain application of BST?

Ans. Binary Search Trees (BSTs) have a wide range of applications in computer science some of them are:

- Searching: BSTs are commonly used to perform searching operations efficiently. Because of the tree structure of BSTs and their ordering property, it is possible to search for a specific element in O(log n) time, where n is the number of elements in the tree.
- Sorting: BSTs can be used to sort a set of elements efficiently. By inserting elements into a BST in sorted order and then performing an in-order traversal of the tree, we can retrieve the elements in sorted order.
- Symbol table: A symbol table is a data structure that is used to store key-value pairs, and it is commonly used in compilers, interpreters, and other software systems. BSTs can be used to implement symbol tables efficiently, with operations such as insert, delete, and search running in O(log n) time.
- File system: BSTs can be used to implement file systems efficiently, with directories and files represented as nodes in the tree. By maintaining the ordering property of the BST, we can quickly search for files and navigate through the directory structure.
- Network routing: BSTs can be used to implement network routing tables efficiently, with IP addresses and routing information stored as key-value pairs in the tree. By searching the tree for a given IP address, we can quickly determine the appropriate route for a packet to take.

(b) Explain with example deletion of a node having two child.?

Ans. Deleting a node with two children from a binary search tree can be a bit more complex than deleting a node with no children or one child. Here is an example of how to delete a node with two children:

- Suppose we have the following binary search tree:

We want to delete the node with key value 10. This node has two children, so we need to find the node with the smallest key value in its right subtree to replace it. Here are the steps we can follow:

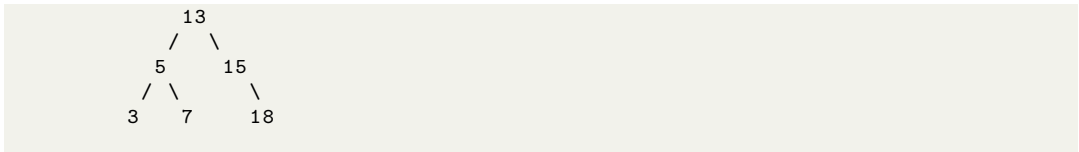
Find the node with the smallest key value in the right subtree of the node to be deleted. In this case, the right subtree of 10 is:

The smallest key value in this subtree is 13.

Copy the key value of the node we found in step 1 (13) to the node we want to delete (10)

Delete the node we found in step 1 (13). Because it has no children, we can simply remove it from the tree.

The resulting tree looks like this:



- Define skewed binary tree.?

  Ans. A skewed binary tree is a type of binary tree in which all of the nodes are either left or right children of their parent. In other words, the tree is "lopsided" in one direction, with all of the nodes either slanting to the left or slanting to the right.

  There are two types of skewed binary trees:

  – Left-skewed binary tree: In a left-skewed binary tree, all of the nodes have a left child, except for the leaf nodes, which have no children. An example of a left-skewed binary tree is:

```
1
 \
  2
   \
    3
     \
      4
       \
        5
```

  – Right-skewed binary tree: In a right-skewed binary tree, all of the nodes have a right child, except for the leaf nodes, which have no children. An example of a right-skewed binary tree is:

```
        1
       /
      2
     /
    3
   /
  4
 /
5
```

  Skewed binary trees can be unbalanced and can lead to inefficient operations such as searching and inserting, as the time complexity of these operations may be O(n) in the worst case, where n is the number of nodes in the tree. Therefore, balancing techniques such as rotation and rebalancing may be applied to improve the performance of skewed binary trees.