Lab 10 Binary Search Tree Implementation

Learning Outcomes:

After successfully completing this lab the students will be able to:

1. Understand the properties of Binary Search Trees and their associated functions.
2. Develop C programs for implementing Binary Search Trees and their associated functions.
3. Use BSTs to load/store data to/from a file on the hard disk.

Pre-Lab Reading Task:

Binary Search Trees:

Binary Search Tree is a node-based binary tree data structure which has the following properties:

1. The left subtree of a node contains only nodes with keys lesser than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. The left and right subtree each must also be a binary search tree.

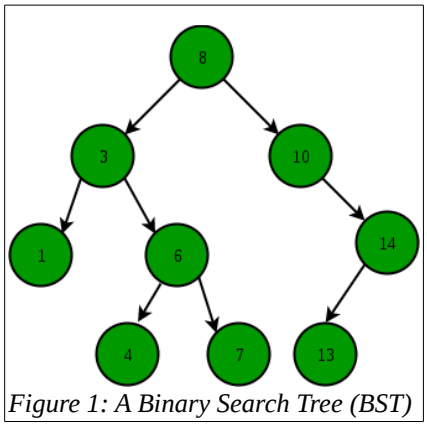

Figure 1: A Binary Search Tree (BST)

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise we recur for left subtree.

Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

Time Complexity:

The worst case time complexity of search and insert operations is **O(h)** where **h** is height of Binary Search Tree. In worst case, we may have to travel from root to the deepest leaf node. The height of a skewed tree may become **n** and the time complexity of search and insert operation may become **O(n).**

Deleting a Node from the tree:

When we delete a node, three possibilities arise.

1. Node to be deleted is leaf.
    - Simply remove from the tree.
2. Node to be deleted has only one child
    - Copy the child to the node and delete the child
3. Node to be deleted has two children
    - Find in-order successor of the node. Copy contents of the in-order successor to the node and delete the in-order successor. Note that in-order predecessor can also be used.

The important thing to note is, in-order successor is needed only when right child is not empty. In this particular case, in-order successor can be obtained by finding the minimum value in right child of the node.

Time Complexity

The worst case time complexity of delete operation is **O(h)** where **h** is height of Binary Search Tree. In worst case, we may have to travel from root to the deepest leaf node. The height of a skewed tree may become **n** and the time complexity of delete operation may become **O(n)**

For more information read **Chapter 10** from the book: ***"Data Structures using C"*** by Reema Thareja.

In-Lab Tasks:

You are provided with skeleton code that builds a Binary Search Tree by adding 10 nodes to it. Functions for node insertion and printing the tree (in-order traversal only) are already implemented. Your task is to complete the following functions.

1. Node deletion
2. Node search
3. Pre-Order and Post-Order printing

Post-Lab Tasks:

Complete the following functions for the BST:

1. Save the Tree data to a file (In-Order, Pre-Order and Post-Order)
2. Load tree from a file containing numbers.