Assignment No - 4 (Group B)

Problem statement:

Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree –

i. Insert new node ii. Find number of nodes in longest path iii.Minimum data value found in the tree iv. Change a tree so that the roles of the left and right pointers are swapped at every node v. Search a value

Pre-requisite

Knowledge of C++ programming

Knowledge of binary search tree (BST) data structure

Objective

- Implement various operations of a binary search tree, such as insertion, finding the smallest and largest element, mirror, etc.
- Derive the time complexities for the above operations on a binary search tree.
- Give that advantages and disadvantages of a binary search tree over other data structures
- Identify applications where a binary search tree will be useful.

Input

Sequence of input data in different orders like increasing order, decreasing order, random order

Output

Display result of each operation with error checking

Outcome

Student will be able to create a binary search tree and perform various operations on it for solving various problems.

Software and Hardware requirements:-

1. **Operating system:** Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10,
2. **RAM**- 2GB RAM (4GB preferable)
3. C++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Theory-

A binary tree is called a binary search tree (BST) if for every node in the tree the data elements in the node is greater than the data elements in all the nodes in the left subtree and is less than or equal to the data elements in all the nodes in the right subtree.

Consider a binary tree T. T is a binary search tree, that is, the value of the data elements in every node N in T greater than to the data elements in every node in left subtree and is less than or equal to the data elements in every node in right subtree.

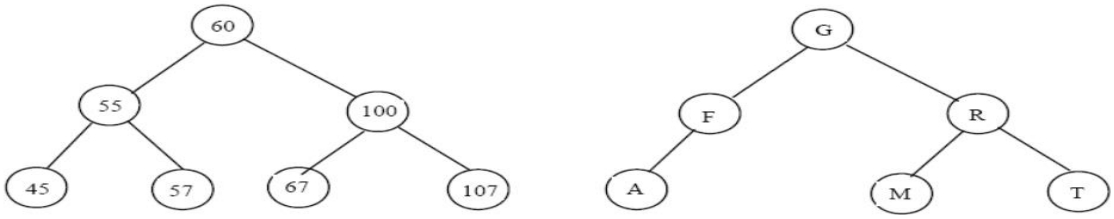


Figure 2. Binary Search Tree (BST)

Linked Representation of Binary Search Tree

A binary search tree can be represented using a set of linked nodes. Each node contains a value and two links named left and right that reference the left child and right child, respectively

Pointer to Left Subtree	Data	Pointer to Right Subtree

Figure 3. Node

The variable *root* refers to the root node of the tree. If the tree is *empty*, root is NULL.

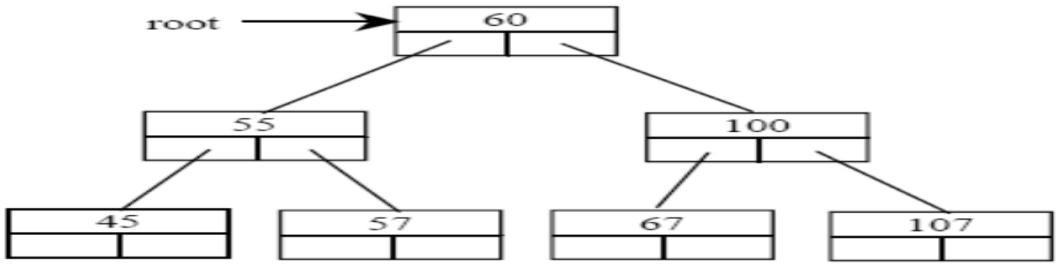


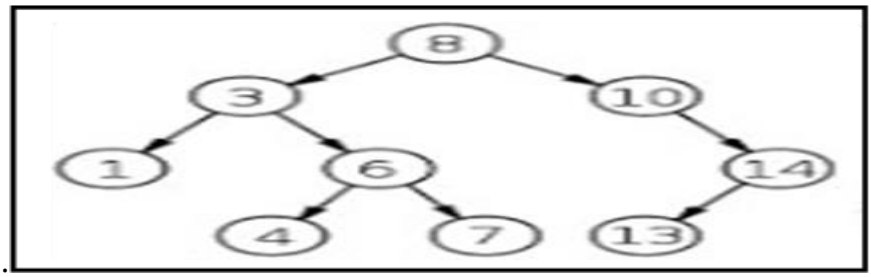
Figure 4. Linked Representation of Binary Search Tree

Operations on Binary search Tree :

1. Insertion

Insertion begins as a search would begin; if the root is not equal to the value, we search the left orright subtrees as before.

Eventually, we will reach an external node and add the value as its right or left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or theright subtree if the new value is greater than or equal to the root



2. Searching :

Searching a binary tree for a specific value can be a recursive or iterative process. This explanation covers a recursive method.

We begin by examining the root node. If the tree is null, the value we are searching for does not exist in the tree. Otherwise, if the value equals the root, the search is successful. If the value is less than the root, searchthe left subtree.

Similarly, if it is greater than the root, search the right subtree. This process is repeated until the value is found or the indicated subtree is null. If the searchedvalue is not found before a null subtree is reached, then the item must not be present in the tree.

3. Deletion :

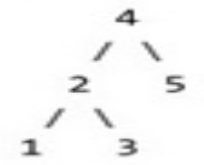
There are three possible cases to consider:

1. Deleting a leaf (node with no children): Deleting a leaf is easy, as we can simply remove it from the tree.
2. Deleting a node with one child: Remove the node and replace it with its child.
3. Deleting a node with two children: Call the node to be deleted N. Do not delete N. Instead, choose either its in-order successor node or its in-order predecessor node, R.Replace the value of N with the value of R, then delete R.

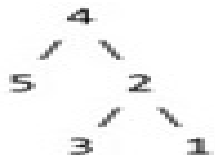
As with all binary trees, a node's in-order successor is the left-most child of its right subtree, and a node's in-order predecessor is the right-most child of its left subtree. In either case, this node will have zero or one children. Delete it according to one of the twosimpler cases above.

Mirror()-

Change a tree so that the roles of the left and right pointers are swapped at every node. So the tree...



is changed to...



The solution is short, but very recursive. As it happens, this can be accomplished without changing the root node pointer, so the return-the-new-root construct is not necessary.

Alternately, if you do not want to change the tree nodes, you may construct and return a new mirror tree based on the original tree.

Algorithms for Operations on binary search tree:

1. Create ( ) // Create binary search tree

Step 1: Allocate the memory by using new keyword for a new node. Make its left and right node Pointer NULL.

Step 2: Accept the data from user and store it in the data part of new node.

Step 3: Check whether the root is pointing to NULL , if it is then assign the value of new node Pointer to the root node pointer.

Step 4: If the root node pointer is not NULL, then define a node pointer (temp) for traversing and store Root node address in it.

Step 5: Compare the values of new node and the node pointed by temp. Step 6: If value in new node is greater than the value in temp, then perform temp=temp->right otherwise temp=temp->left

Step 7: Repeat the step 5 and 6 until temp encounters the NULL value.

Step 8: Link the new node to the parent node of temp properly according to its value.

// Traverse Binary search tree using Recursive Inorder Traversal

2. Inorder (node *temp)

Step 1: Check whether temp==NULL

Step 2: If not then call function inorder (temp->left).

Step 3: Display the data and in the temp node.

Step 4: Call the function inorder (temp->right).

//Traverse Binary search tree using Recursive Preorder Traversal

3. preorder (node *temp)

Step 1: Check whether temp==NULL

Step 2: If not then Display the data in the temp node.

Step 3: Call function preorder (temp->left).

Step 4: Call the function preorder (temp->right).

//Traverse Binary search tree using Recursive Postorder Traversal

4. postorder (node *temp)

Step 1: Check whether temp==NULL

Step 2: If not then Call function postorder(temp->left).

Step 3: Call the function postorder (temp->right).

Step 4: Display the data in the temp node.

// Search an element from binary search Tree

5. search ( )

Step 1: Accept the data which is to be searched from the user.

Step 2: Store the address of the root node in the current code pointer (cn).

Step 3: Traverse the tree until the required data and the data in the cn matches.

If the required data is greater than the word in the cn then perform

cn=cn->right

else

Perform cn=cn->left

If cn encounters the NULL value

then come out of the loop statement by using break statement.

Step 4: Check whether the cn is NULL, if it is then required data is not found in the tree else the required data is present in the tree.

Time complexity:

Time complexity of the various functions of the binary tree are as follows:

Sr.No.	Function	Time Complexity
1	create ( )	O(nlogn)
2	inorder ( )	O(n)
3	preorder ( )	O(n)
4	postorder ( )	O(n)
5	search ( )	O(log n)
6	display ( )	O(n)
7	delete_node ( )	O(log n)
8	Insert()	O(log n)

The overall time complexity of the program is O(n log n).

Conclusion:

Thus we studied the binary search tree and its operations and successfully implemented it for solving the given problem

Sample Oral Questions :

1. What is Binary search tree?
2. What are the members of structure of tree & what is the size of structure?
3. What are rules to construct binary search tree?
4. How general tree is converted into binary tree?
5. What is time complexity for search operation in BST?
6. What is preorder, postorder, in order traversal?
7. Traverse tree using BFS & DFS.