Assignment No.-1

Title: - Binary Search Tree

Problem Statement: - Beginning with an empty binary search tree(BST), 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

Learning Objectives:-

1. To understand the binary search tree data structure.
2. To understand the various operations on binary search tree.
3. To implement binary search tree using object oriented concepts.

Learning Outcomes: - After successfully completing this assignment you should be able to:

1. Define the classes required to implement binary search tree using object oriented features in C++.
2. Analyze and trace the working of recursive functions used in implementation.
3. Understand and implement various operations on BST such as Insert, Search, Finding maximum, Finding minimum, mirror image etc.

Software Tools Used: - linux GCC compiler, Eclipse Editor on 64-bit Ubuntu 14.04 operating system.

Hardware Used: - PC configuration- Intel Core2duo 2.93 GHZ CPU, 4GB RAM, 320GB HDD, 17” Monitor, Keyboard and Mouse.

Theory: - Binary search tree is the binary tree which satisfies the following condition:

a) All elements should be distinct.
b) For any node its left subtree should contain the less values than the node and its right subtree should contain the larger values than the node.
c) All subtree should be BST.

Searching time in the BST is O(log n).

e.g.

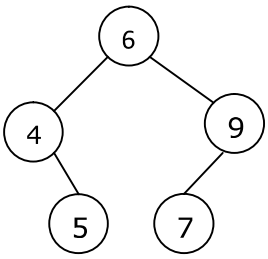

Fig.(a) Valid BST

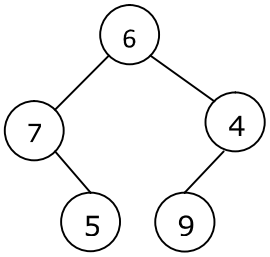

Fig.(b) Invalid BST

Representation Used for BST:
In this assignment linked representation of BST is used. For that we need to define node structure to be linked in BST structure. The BST node structure has the following fields.

*Left	Data	*Right

The node contains 3 fields as follows:

a) ***Left** holds the address of left child/subtree if present otherwise NULL.
b) **Data** field hold data of node may contain single value or record.
c) ***Right** holds the address of right child/subtree if present otherwise NULL.

Algorithm: - The various algorithms are as follows:

A. Create ()
1. Start
2. Read the value.
3. If value is already present in BST
   Do not insert.
   Otherwise
   Insert(Value);
4. Go to step 2 if you want to add anymore values.
5. Stop.

B. Insert(Val)
1. Start
2. Create the new node P initialized with value given.
3. If Root is empty
   Root=P; Go to
4. Initialize T to Root.
5. If Val< T->Val
   If left child is empty
   T->left=P; Go to 6
   Otherwise
   Descend the tree in left direction by one level.go to5
   Otherwise
   If right child is empty

```
                T->right=P; Go to 6
             Otherwise
                Descend the tree in right direction by one level.goto5
     6. Stop

C. Search(Val)
     1. Start
     2. Initialize T with Root
     3. If T is NULL go to 6
     4. If T->data==Val
           return T go to 6
     5. If Val<T->data
           Descend the tree in left direction by one level go to 4
     6. If Val>T->data
           Descend the tree in right direction by one level go to 4
     7. Stop

D. Height(T)
     1. Start
     2. If T is NULL return 0 go to 7
     3. If T is leaf node return 0 go to 7
     4. HL = Height(Left Subtree)
     5. HR = Height(Right Subtree)
     6. If HL>HR
           return HL+1 go to 7
        Otherwise
           return HR+1 go to 7
     7. Stop

E. Find_min(T)
     1. Start
     2. Initialize T with Root
     3. If T is NULL return -1 go to 6
     4. Descend the tree in left direction by one level
     5. If T->Left is NULL return T->data go to 6
        Otherwise go to 4
     6. Stop

F. Find_Max(T)
     1. Start
     2. Initialize T with Root
     3. If T is NULL return -1 go to 6
     4. Descend the tree in right direction by one level
     5. If T->Right is NULL return T->data go to 6
        Otherwise go to 4
     6. Stop

G. Mirror_Image(T)
     1. Start
     2. If T is NULL return go to 7
```

3. Create new node P
4. P->left = Mirror_Image(T->right)
5. P->right = Mirror_Image(T->left)
6. return P
7. Stop

H. Inorder(T)

1. Start
2. If T is NULL return go to 6
3. Inorder(T->left)
4. Visit T
5. Inorder(T->right)
6. Stop

Program Code with Sample Output:-

Analysis of Algorithm:-

1. Create() = O(n)
2. Insert() = O(log n)
3. Search() = O(log n)
4. Height() = O(log n)
5. Find_Min()= Find_Max() = At most O(n)
6. Inorder() = O(n)

Applications:-

1. BST can be used to implement Dictionary.
2. Used in *many* search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
3. It is used to implement multilevel indexing in DATABASE.
4. It is used to implement searching Algorithm.

Conclusion:-

In your Own words.

References:-

1. Horowitz, Sahani, Dinesh Mehata, —Fundamentals of Data Structures in C++, Galgotia Publisher, ISBN: 8175152788, 9788175152786.
2. Peter Brass, —Advanced Data Structures‖, Cambridge University Press, ISBN: 978-1-107-43982-5.

Date of Completion:-

FAQ's:-

1. What are advantages of dynamic memory allocation over static memory allocation?
2. What is necessity of friend class in oop?
3. If tree is complete binary search tree having total number of nodes are 1000 then what is the height of it?
4. What are the maximum no of comparison required to search value in BST of n nodes?
5. What are the applications of BST?