K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch:_A3 Roll No.:16010121051
Experiment No.
Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementation of BST & Binary tree traversal techniques.

Objective: To Understand and Implement Binary Search Tree, Preorder, Postorder and Inorder Traversal Techniques.

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.geeksforgeeks.org/binary-tree-data-structure/
5. https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html

Abstract:

A tree is a non- linear data structure used to represent hierarchical relationship existing among several data items. It is a finite set of one or more data items such that, there is a special data item called the root of the tree. Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.

A binary tree is a finite set of nodes. It is either empty or It consists a node called root with two disjoint binary trees-Left subtree, Right subtree. The Maximum degree of any node is 2

A Binary Search Tree is a node-based binary tree data structure in which the left subtree of a node contains only nodes with keys lesser than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. The left and right subtree each must also be a binary search tree.

Related Theory: -

Preorder Traversal of BST

Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.

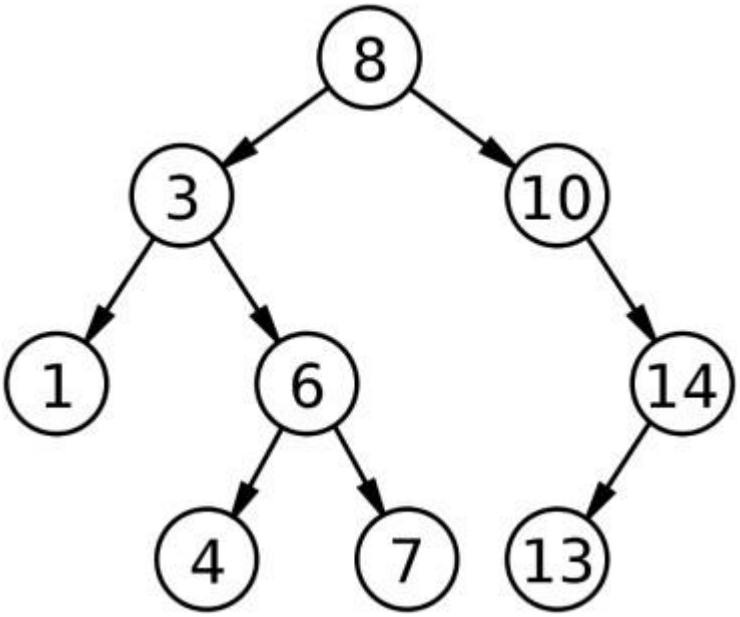
Postorder Traversal of BST

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.

Inorder Traversal of BST

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.

Diagram for :



Preorder Traversal of BST

8 3 1 6 4 7 10 14 13

Postorder Traversal of BST

1 3 4 6 7 8 10 13 14

Inorder Traversal of BST

1 4 7 6 3 13 14 10 8

Algorithm for Implementation of BST & Binary tree traversal techniques:

Search in BST: -

1. Start from the root.

2. Compare the searching element with root, if less than root, then recurse for left, else recurse for right.

3. If the element to search is found anywhere, return true, else return false.

Insertion in BST: -

1. Start from the root.

2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.

3. After reaching the end, just insert that node at left (if less than current) else right.

Deletion in BST: -

1. Search the node that to be deleted in BST.

2. If node has left or right subtree then find the inorder predecessor or inorder successor respectively (call it new node) and replace the node data value with its data. Then make recursive call to the delete function for that new node.

3 Else delete the node and assign the parent's child pointer(which is node) to NULL.

Implementation Details:

1) Enlist all the Steps followed and various options explored.

A structure called Node is created which has two struct pointers (left and right) and

1 int variable (data).

A class called BST is created with functions for insertion, deletion and searching.

For insertion in BST: -

• If root is null create new node and assign the data to it.

• Else traverse in tree according to the data of node.

For searching: -

• If node is null then the node with required data isn't there in the BST.

• Else traverse in tree according to the data of node.

For deletion: -

• Search the node to be deleted.

• If the node has left or right child then replace its data value with inorder

successor(if right child is present) or predecessor of current node. Then search of

that node in the corresponding subtree.

• Else delete the node and assign its parents pointer (which points to current node)

as NULL.

An object of BST class is created and a menu driven program is made for the user to

choose an operation.

Assumptions made for Input:

The value of all nodes are integers and |value|<=INTMAX

Program source code for Implementation of BST & Binary tree traversal techniques :

```
#include<iostream>


using namespace std;


struct Node{
    int data;
    Node *left;
    Node *right;
};


Node* Create(int data){
    Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}


void inorder(Node *root){
    if(root == NULL){
        return;
```

```
    }
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

Node* findMinimum(Node* cur){
    while(cur->left != NULL){
        cur = cur->left;
    }
    return cur;
}

Node* insert(Node* root,int data){
    if(root == NULL){
        return Create(data);
    }
    if(data<root->data){
        root->left = insert(root->left,data);
    }
    else{
        root->right = insert(root->right, data);
    }
    return root;
```

```
}


void search(Node* &cur,int data,Node* &parent){
    while(cur != NULL && cur->data != data){
        parent = cur;
        if(data < cur->data){
            cur = cur->left;
        }
        else{
            cur = cur->right;
        }
    }
}


void remove(Node*& root,int data){
    Node* parent = NULL;
    Node* cur = root;
    search(cur,data,parent);
    if(cur == NULL){
        return;
    }
    if(cur->left==NULL && cur->right==NULL){
        if(cur != root){
            if(parent->left == cur){
```

```
                parent->left = NULL;
            }
            else{
                parent->right = NULL;
            }
        }
        else{
            root = NULL;
        }
        free(cur);
    }
    else if(cur->left && cur->right){
        Node* succ = findMinimum(cur->right);
        int val = succ->data;
        cur->data = val;
    }
    else{
        Node* child = (cur->left)?cur->left:cur->right;
        if(cur != root){
            if(cur == parent->left){
                parent->left = child;
            }
            else{
                parent->right = child;
```

```
            }
        }
        else{
            root = child;
        }
        free(cur);
    }
}

int main(){
    Node* root = NULL;
    cout << "1. Add \n";
    cout << "2. Print the tree \n";
    cout << "3. Remove \n";
    while(1){
        int i,data;
        cout << ">> ";
        cin >> i;
        switch (i)
        {
        case 1:
            cout << "Enter the data \n";
            cin >> data;
            root = insert(root,data);
```

```
            break;

        case 2:

            inorder(root);

            cout << "\n";

            break;

        case 3:

            cout << "Enter the data \n";

            cin >> data;

            remove(root,data);

            break;

        default:

            break;

        }

    }

}
```

Output Screenshots for Each Operation:

```
PS C:\Users\Vishr\Documents\SEM 3\DS> cd "c:\Users\Vishr\Documents\SEM 3\DS\" ; if ($?) { g++ Binary
ee }
1. Add
2. Print the tree
3. Remove
>> 1
Enter the data
10
>> 1
Enter the data
21
>> 1
Enter the data
65
>> 1
Enter the data
1
>> 1
Enter the data
87
>> 2
1 10 21 65 87
>> 3
Enter the data
21
>> 3
Enter the data
65
>> 2
1 10 87
>>
```

Conclusion:-

In this experiment we learnt about various operation and traversals in BST.

PostLab Questions:

1) Illustrate 2 Applications of Trees.

Suffix Tree : For quick pattern searching in a fixed text.
Spanning Trees and shortest path trees are used in routers and bridges respectively in computer networks

2) Compare and Contrast between B Tree and B+ Tree?

S.No	B tree	B+ tree
1.	B tree is a popular terminology that belongs to the computer science family. It is a balancing tree that helps in maintaining and sorting data, and also grants searches, insertions, deletions, and sequential access.	B+ tree is nothing but an advancement of B tree that allows efficient and smooth insertion, deletion, and sequential access.
2.	In the case of B tree, the leaf nodes include data pointers.	In the case of B+ tree, only the leaf nodes include data pointers.
3.	Here, the insertion may take longer.	Here, the insertion is easier and faster than the B tree.
4.	In B tree, there is no duplicate of keys sustained in the tree.	In B+ tree, duplicates of keys are maintained.
5.	The search process may take a longer time because all the keys are not obtainable at the leaf.	Here the search is faster than the B tree as the keys are present at leaf nodes.