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

Batch: B2 Roll No.: 110

Experiment / assignment / tutorial No.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

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: -

Unlike linked lists, one-dimensional arrays and other linear data structures, which are canonically traversed in linear order, trees may be traversed in multiple ways. They may be traversed in depth-first or breadth-first order. There are three common ways to traverse them in depth-first order: in-order, pre-order and post-order. Beyond these basic traversals, various more complex or hybrid schemes are possible, such as depth-limited searches like iterative deepening depth-first search.

Algorithm

Preorder Traversal of BST

1. *Visit the root.*
2. *Traverse the left subtree, i.e., call Preorder(left->subtree)*
3. *Traverse the right subtree, i.e., call Preorder(right->subtree)*

Postorder Traversal of BST

1. *Traverse the left subtree, i.e., call Postorder(left->subtree)*

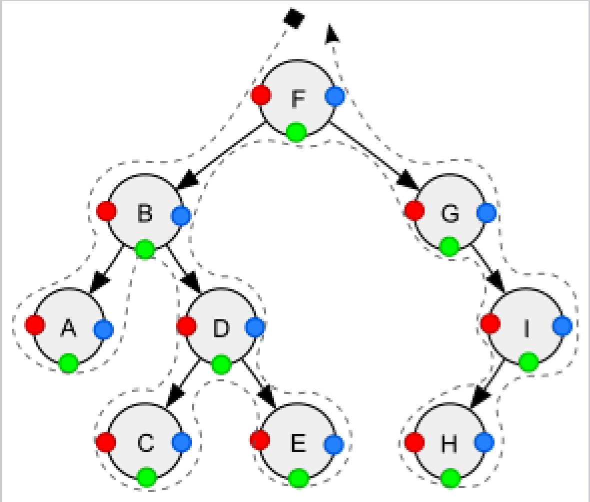
2. *Traverse the right subtree, i.e., call Postorder(right->subtree)*
3. *Visit the root*

Inorder Traversal of BST

1. *Traverse the left subtree, i.e., call Inorder(left->subtree)*
2. *Visit the root.*
3. *Traverse the right subtree, i.e., call Inorder(right->subtree)*

Source https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

Diagram for :



Depth-first traversal (dotted path) of a binary tree:

Pre-order (node visited at position red ●*):*
F, B, A, D, C, E, G, I, H;

In-order (node visited at position green ●*):*
A, B, C, D, E, F, G, H, I;

Post-order (node visited at position blue ●*):*
A, C, E, D, B, H, I, G, F.

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

```
/********************************************************************
*********


array implementation of binary search tree.

********************************************************************
*********/

#include <iostream>

const int len = 31;

using namespace std;

class structure{
```

```
public:  int arr[len];

public:  structure(){


 }

 public: void print(){

     for(int i=0;i<len;i++){

         //cout<<arr[i]<<",";

     }

     infix(1); cout<<"\n";

     prefix(1); cout<<"\n";

     postfix(1); cout<<"\n";


 }

 public : void infix(int a){

     if(arr[a]!=0){

         if(a<len){

             infix(a*2);

         cout<<arr[a]<<",";


         infix(a*2+1);

        }

     }


 }

  public : void prefix(int a){
     if(arr[a]!=0){
         if(a<len){
                cout<<arr[a]<<",";
             infix(a*2);

         infix(a*2+1);
        }
     }

 }
  public : void postfix(int a){
     if(arr[a]!=0){
         if(a<len){
             infix(a*2);

         infix(a*2+1);
         cout<<arr[a]<<",";
        }
     }

 }

public: void add(int a){
        int currentnode=1;
  while(true){
        int currentval=arr[currentnode];
        if(currentval==0){
            arr[currentnode]=a;
            break;
        }
        if(currentval>a){
            currentnode=2*currentnode+1;
        }
        else if(currentval<a){
            currentnode=2*currentnode;
        }
        if(currentnode>len){
            cout<<"Overflow";
           break;
        }

    }
  }
public: int search(int a){
    int currentnode=1;
    while(true){
        int currentval=arr[currentnode];
        if(currentval==a){
            return currentnode;
        }
```

```
          if(currentval>a){
              currentnode=2*currentnode+1;
          }
          else if(currentval<a){
              currentnode=2*currentnode;
          }
          if(currentnode>len){
              return -1;
          }

      }
  }
};
int main()
{   structure *my=new structure();
my->add(20);
my->add(37);
my->add(32);
my->add(13);
my->add(9);
my->add(34);
//cout<<my->search(2);
my->print();
    cout<<"Hello World";
    return 0;

}
```

Output Screenshots for Each Operation:

```
37,34,32,20,13,9,

20,37,34,32,13,9,

37,34,32,13,9,20,

Hello World
```

Conclusion:-
Thus we have successfully understood the working behind BST and different tree traversal techniques have been studied. We studied inorder, preorder and postorder traversal. All these techniques have different applications. We implemented these techniques in C.

PostLab Questions:

1) **Illustrate 2 Applications of Trees.**
   a) **Infix to postfix conversions**
   b) **Dictionary of words for autocompletion.**
2) **Compare and Contrast between B Tree and B+ Tree?**

Both are selfbalancing trees. B+ tree saves keys in nodes and data in leaves. B tree stores both keys and nodes in nodes. B tree is lower in terms of space complexity while B+ tree has lower time complexity.