DATA STRUCTURES AND ALGORITHMS LABORATORY

Group B

Assignment No. 1

Name :- Ojus Pravin Jaiswal
Roll No. :- SACO19108
Division :- A

DATA STRUCTURE LABORATORY

Group B

Assignment 1

Title: 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 from root, 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

Objectives:

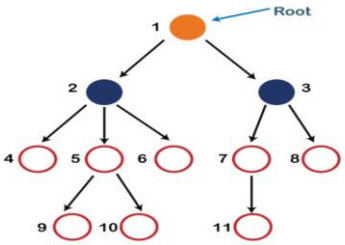
1. To understand concept of tree data structure
2. To understand concept & features of object oriented programming.

Outcome:

To implement Binary Tree Data Structure to store a numbers in it. Also perform basic operations on binary tree.

Theory:

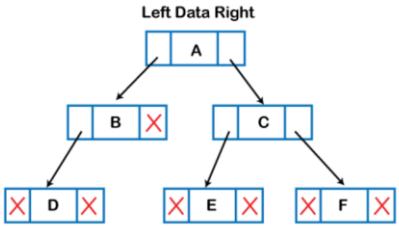
A tree is also one of the data structures that represent hierarchical data. Suppose we want to show the employees and their positions in the hierarchical form then it can be represented as shown below:



- A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.
- A tree data structure is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a Tree are arranged in multiple levels.
- In the Tree data structure, the topmost node is known as a root node. Each node contains some data, and data can be of any type. In the above tree structure, the node contains the name of the employee, so the type of data would be a string.
- Each node contains some data and the link or reference of other nodes that can be called children.

DATA STRUCTURE LABORATORY

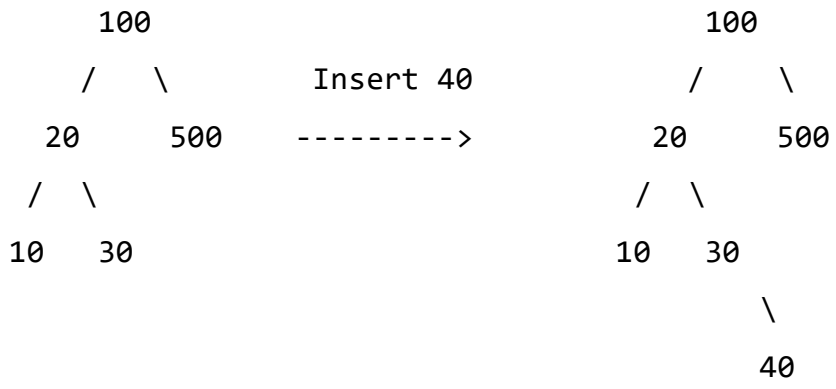
The tree data structure can be created by creating the nodes dynamically with the help of the pointers. The tree in the memory can be represented as shown below:

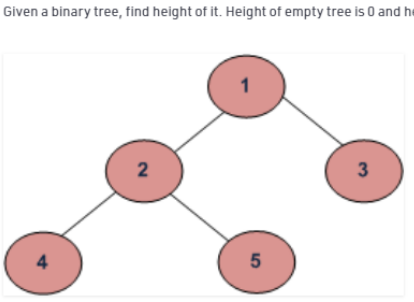


The above figure shows the representation of the tree data structure in the memory. In the above structure, the node contains three fields. The second field stores the data; the first field stores the address of the left child, and the third field stores the address of the right child.

Insertion of a key

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



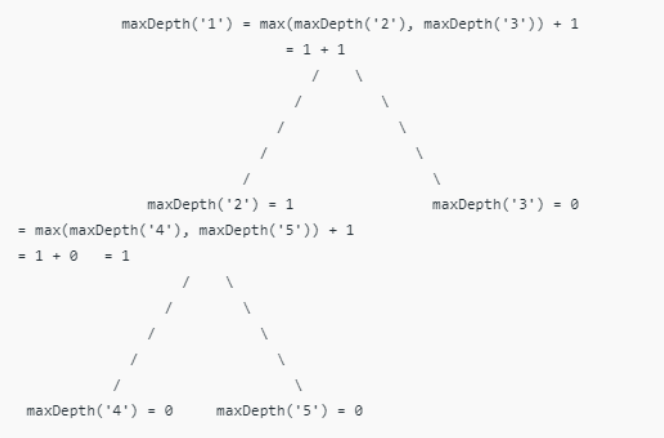


Algorithm: maxDepth()

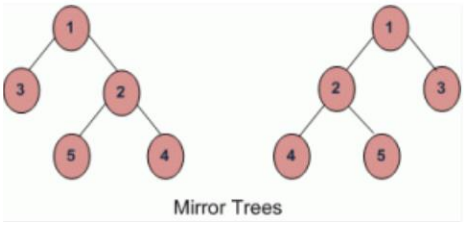
1. If tree is empty then return 0
2. Else
   (a) Get the max depth of left subtree recursively i.e.,
   call maxDepth(tree->left-subtree)
   (a) Get the max depth of right subtree recursively i.e.,
   call maxDepth(tree->right-subtree)

(c) Get the max of max depths of left and right
subtrees and add 1 to it for the current node.
max_depth = max(max dept of left subtree,
max depth of right subtree)
+ 1

(d) Return max_depth



Mirror of a Tree: Mirror of a Binary Tree T is another Binary Tree M(T) with left and right children of all non-leaf nodes interchanged.



Algorithm – Mirror(tree):

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
temp = left-subtree
left-subtree = right-subtree
right-subtree = temp

To find the node with minimum value in a Binary Search Tree Just traverse the node from root to left recursively until left is NULL. The node whose left is NULL is the node with minimum value.

DATA STRUCTURE LABORATORY

Inorder traversal

First, visit all the nodes in the left subtree

Then the root node

Visit all the nodes in the right subtree

Preorder traversal

Visit root node

Visit all the nodes in the left subtree

Visit all the nodes in the right subtree

Postorder traversal

Visit all the nodes in the left subtree

Visit all the nodes in the right subtree

Visit the root node

Search operation in binary search tree will be very similar. Let's say we want to search for the number, what we'll do is we'll start at the root, and then we will compare the value to be searched with the value of the root if it's equal we are done with the search if it's lesser we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are lesser and all the elements in the right subtree are greater. Searching an element in the binary search tree is basically this traversal in which at each step we will go either towards left or right and hence in at each step we discard one of the sub-trees.

Software Required: g++ / gcc compiler- / 64 bit Fedora, eclipse IDE

Input: Book name & its number of sections and subsections along with name.

DATA STRUCTURE LABORATORY

Program:

```
//============================================================================
// Name        : DSA.cpp
// Author      : Ojus
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

class BST
{
    int data;
    BST *left, *right;

public:
    // Default constructor.
    BST();

    // Parameterized constructor.
    BST(int);

    // Insert function.
    BST* Insert(BST*, int);
    BST* search(BST* , int );
    // traversal.
    void Inorder(BST*);
    void Preorder(BST*);
    void Postorder(BST*);
    int maxDepth(BST*);
    int minValue(BST*) ;
    void mirror(BST*) ;
};

// Default Constructor definition.
BST ::BST()
    : data(0)
    , left(NULL)
    , right(NULL)
{
}

// Parameterized Constructor definition.
BST ::BST(int value)
{
    data = value;
    left = right = NULL;
}

// Insert function definition.
BST* BST ::Insert(BST* root, int value)
{
    if (!root)
    {
        // Insert the first node, if root is NULL.
```

```
        return new BST(value);
    }

    // Insert data.
    if (value > root->data)
    {
        // Insert right node data, if the 'value'
        // to be inserted is greater than 'root' node data.

        // Process right nodes.
        root->right = Insert(root->right, value);
    }
    else
    {
        // Insert left node data, if the 'value'
        // to be inserted is greater than 'root' node data.

        // Process left nodes.
        root->left = Insert(root->left, value);
    }

    // Return 'root' node, after insertion.
    return root;
}

// Inorder traversal function.
// This gives data in sorted order.
void BST ::Inorder(BST* root)
{
    if (!root) {
        return;
    }
    Inorder(root->left);
    cout << root->data << endl;
    Inorder(root->right);
}

void BST ::Preorder(BST* root)
{
    if (!root) {
        return;
    }
    cout << root->data << endl;
    Preorder(root->left);
    Preorder(root->right);

}
void BST ::Postorder(BST* root)
{
    if (!root) {
        return;
    }
    Postorder(root->left);
    Postorder(root->right);
    cout << root->data << endl;
}
int BST ::maxDepth(BST* root)
{
    if (!root)
```

```
        return 0;
    else
    {
        /* compute the depth of each subtree */
        int lDepth = maxDepth(root->left);
        int rDepth = maxDepth(root->right);

        /* use the larger one */
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);
    }
}
int BST ::minValue(BST* root)
{
      if (!root)
              return 0;
/* loop down to find the leftmost leaf */
      else{
             while (root->left != NULL)
             {
                    root = root->left;
             }
             return(root->data);
      }
}
void BST ::mirror(BST* root)
{
      BST *temp;
            if(root!=NULL)
            {
                  temp=root->left;
                  root->left=root->right;
                  root->right=temp;
                  mirror(root->left);
                  mirror(root->right);
            }
}
BST* BST::search(BST* root, int key)
{
       BST *ptr;
       ptr=root;
     while(ptr) {
       if(key>ptr->data)
              ptr=ptr->right;
       else if(key<ptr->data)
              ptr=ptr->left;
       else
              break;
       }
       if(ptr) {
              cout<<"\n Element "<<key<<"which was searched is found "<<endl;
       }
       else
              cout<<"\n Element" <<key<<"does not exist in the binary
tree"<<endl;
       return(root);
}
```

```
// Driver code
int main()
{
    BST b, *root = NULL;
    int n,e,c;
    cout<<"\n----------BINARY SEARCH TREES----------\n";
    cout<<"\nEnter no. of nodes : ";
    cin>>n;
    cout<<"\nEnter root node : ";
    cin>>e;
    root = b.Insert(root, e);
    for(int i=2;i<(n+1);i++){
       cout<<"\nEnter node no. "<<i<<" : ";
       cin>>e;
       b.Insert(root, e);
    }
    while(1){
    cout<<"\n-----Menu-----\n1) Insert node in BST\n2) Inorder Traversal Of
BST\n3) Preorder Traversal Of BST\n4)Postorder Traversal Of BST\n5) Max Depth Of
BST\n6) Minimum Value in BST\n7) Mirror Of BST\n8) Search node in BST\n9) Exit
Program ";
    cout<<"\nEnter your choice : ";
    cin>>c;
    switch(c){
       case 1:
             cout<<"\nEnter node : ";
             cin>>e;
             b.Insert(root, e);
             break;
       case 2:
             cout<<".....................Inorder....................."<<endl;
             b.Inorder(root);
             break;
       case 3:

       cout<<".....................Preorder....................."<<endl;
             b.Preorder(root);
             break;
       case 4:
             cout<<".....................Postorder................."<<endl;
             b.Postorder(root);
             break;
       case 5:
             cout<<".....................MaxDepth................."<<endl;
             cout<<b.maxDepth(root)<<endl;
             break;
       case 6:
             cout<<".....................MinValue................."<<endl;
             cout<<b.minValue(root)<<endl;
             break;
       case 7:
             cout<<".....................Mirror................."<<endl;
             b.mirror(root);
             b.Inorder(root);
             break;
       case 8:
             cout<<".....................Search................."<<endl;
             b.search(root, 72);
             break;
```

```
        case 9:
                cout<<"\nProgram Exited!!!";
                exit(0);
        default:
                cout<<"\nWrong choice entered!!!";
    }
    }
    return 0;
}
```

DATA STRUCTURE LABORATORY

Output:

```
----------BINARY SEARCH TREES----------

Enter no. of nodes : 10

Enter root node : 50

Enter node no. 2 : 0

Enter node no. 3 : 10

Enter node no. 4 : 20

Enter node no. 5 : 20

Enter node no. 6 : 40

Enter node no. 7 : 60

Enter node no. 8 : 70

Enter node no. 9 : 80

Enter node no. 10 : 90

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 1

Enter node : 100

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 2

.....Inorder.....
0
10
20
20
40
50
60
70
80
90
100
```

```
-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 3

.....................Preorder.....................
50
0
10
20
20
40
60
70
80
90
100

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 4

.....................Postorder.................
20
40
20
10
0
100
90
80
70
60
50

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 5
```

```
.....................MaxDepth.............
6

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 6

.....................MinValue.............
0

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 7

.....................Mirror.............
100
90
80
70
60
50
40
20
20
10
0

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 8

.....................Search.............

Enter node to be searched : 55

Element 55 does not exist in the binary tree
```

```
-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 10

Wrong choice entered!!!

-----Menu-----
1) Insert node in BST
2) Inorder Traversal Of BST
3) Preorder Traversal Of BST
4) Postorder Traversal Of BST
5) Max Depth Of BST
6) Minimum Value in BST
7) Mirror Of BST
8) Search node in BST
9) Exit Program

Enter your choice : 9

Program Exited!!!

[Program finished]
```

Conclusion: This program implements tree data structure.