MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 3

Assignment Title: Create a Binary Search tree and find its mirror image. Print original & new tree level wise. Find height & print leaf nodes.

Aim: Implement Binary Search tree and its operations.

Prerequisite:

1. Basic concepts of Binary Search tree and its representation
2. Operations of Binary Search Tree.

Objectives:

Implement a Program in C++ for the following operations on Binary Search Tree:

1. Create a Binary Search tree of N nodes using linked list.
2. Recognize and define the basic attributes of a binary tree
3. Perform all operations and print leaf nodes, height and mirror image.

Outcomes:

Upon Completion of the assignment the students will be able to

1. Create and implement BST.
2. Understand and analyse various operation of the BST
3. Utilize BST ADT to create dictionary like application programmes.

Theory:

Binary Search Tree is a node-based binary tree data structure which has the following

properties:

- 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.

A binary search tree, also known as an ordered binary tree is a variant of binary tree in which the nodes are arranged in order. In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node. Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node. The same rule is applicable to every sub-tree in the tree (Ref. Figure 1)

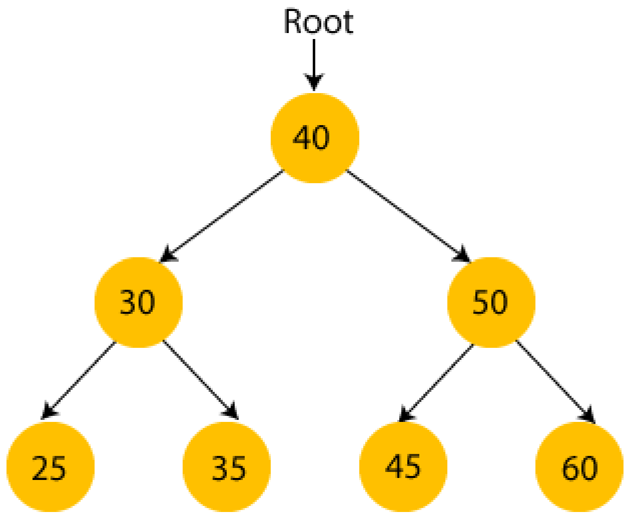


Figure 1: Example of Binary Search Tree

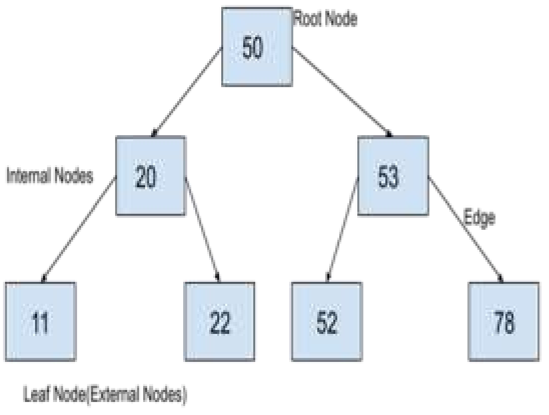
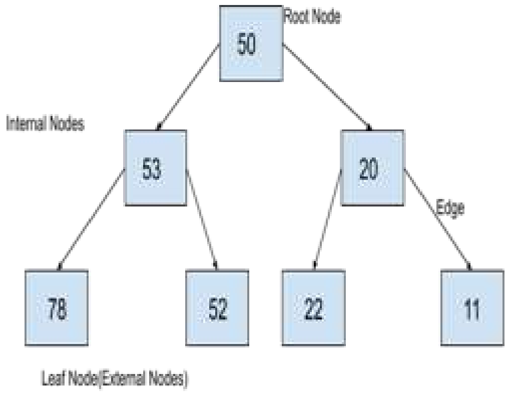
The average running time of a search operation is O(log2n) as at every step, we eliminate half of the sub-tree from the search process. Due to its efficiency in searching elements, binary search trees are widely used in dictionary problems where the code always inserts and searches the elements that are indexed by some key value.
The properties of the Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no order, then we may have to compare every key to search for a given key.

Mirror image of a binary tree:

Mirror image of a binary tree is another binary tree which can be created by swapping left child and right child at each node of a tree. So, to find the mirror image of a binary tree, we just have to swap the left child and right child of each node in the binary tree.

The algorithm for finding a mirror image of a binary tree can be formulated as follows.

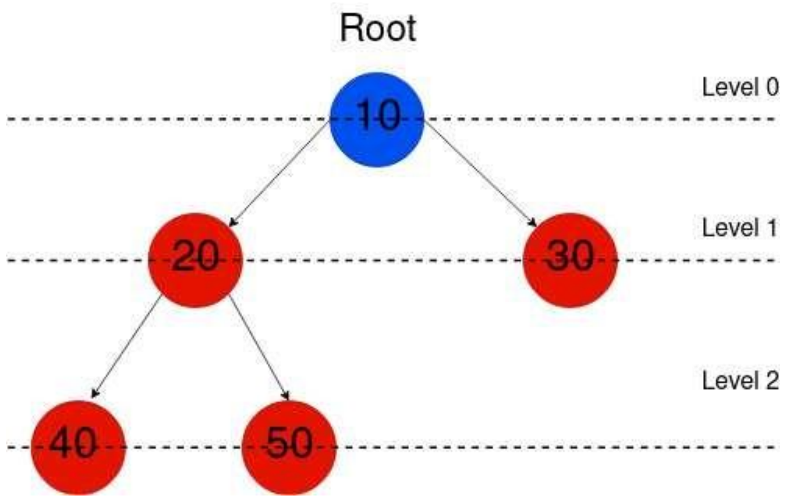
1. Start from the root node.
2. Recursively find the mirror image of the left subtree.
3. Recursively find the mirror image of the right subtree.
4. Swap the left and right subtree.

 Root Node 50 Internal Nodes 20 53 Edge 11 22 52 78 Leaf Node(External Nodes)	 Root Node 50 Internal Nodes 53 20 Edge 78 52 22 11 Leaf Node(External Nodes)
Binary Search Tree	**Mirror Image of Binary Search Tree**

Level Order Traversal

A Level Order Traversal is a traversal which always traverses based on the level of the tree.

So, this traversal first traverses the nodes corresponding to Level 0, and then Level 1, and so on, from the root node.



In the example Binary Tree above, the level order traversal will be:

(Root) 10 -> 20 -> 30 -> 40 -> 50

To do this, we need to do 2 things.

1. We must first find the height of the tree
2. We need to find a way to print the nodes corresponding to every level.

Find the height of the Tree

We will find the height of the tree first. To do this, the logic is simple.

Since the height of the tree is defined as the largest path from the root to a leaf. So we can recursively compute the height of the left and right sub-trees, and find the maximum height of the sub-tree. The height of the tree will then simply be the height of the sub-tree + 1.

Conclusion:

1. In Binary search tree, the left subtree has elements less than the nodes element and the right subtree has elements greater than the nodes element.
2. Binary Search Tree does not allow duplicate values.
3. To print level order without queue, we need to find height of the tree first.

Frequently ask questions:

1. How we can find mirror image of Binary Search Tree?
2. What is a process to find height of Binary Search Tree?
3. Write down steps to perform level order traversal of binary search tree.

MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY PUNE, INDIA

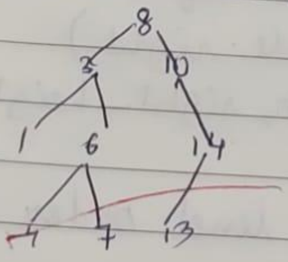
Name - Sourav Shailesh Toshniwal
Class - SY - I (B)
Roll No. - 2213047
Subject - Advanced Data Structure Laboratory.

<u>Assignment - 3</u>

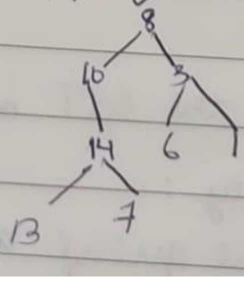
1. How we can find mirror image of binary search Tree?

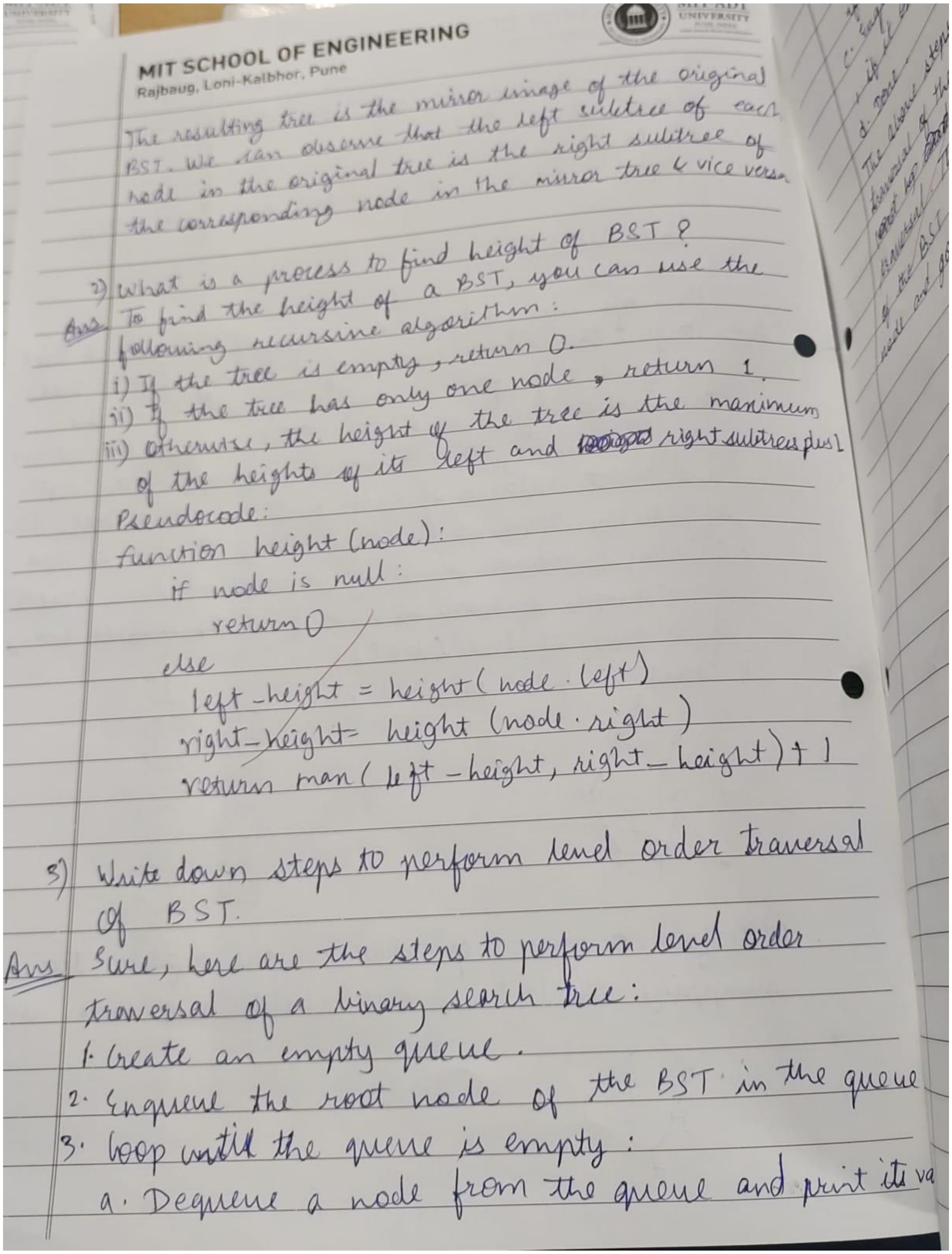
<u>Ans</u> To find the mirror image of a binary search tree, we need to perform a mirror operation on the original binary search tree. The mirror operation on a binary tree involving swapping the left and right children of each node in the tree. This effectively reverses the tree along its vertical axis, creating a mirror image of a original tree. Here's an example of how we can find the mirror image of a original tree (BST):

i) Start with the original BST :



ii) Perform a mirror operation on the original tree by swapping the left and right children of each node




MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

The resulting tree is the mirror image of the original BST. We can observe that the left subtree of each node in the original tree is the right subtree of the corresponding node in the mirror tree & vice versa

2) What is a process to find height of BST ?

Ans To find the height of a BST, you can use the following recursive algorithm:

i) If the tree is empty, return 0.
ii) If the tree has only one node, return 1.
iii) Otherwise, the height of the tree is the maximum of the heights of its left and ~~[illegible]~~ right subtrees plus 1

Pseudocode:

```
function height (node):
    if node is null:
        return 0
    else
        left-height = height (node . left)
        right_height = height (node . right)
        return max ( left-height, right_height ) + 1
```

3) Write down steps to perform level order traversal of BST.

Ans Sure, here are the steps to perform level order traversal of a binary search tree:

1. Create an empty queue.
2. Enqueue the root node of the BST in the queue
3. Loop until the queue is empty:
   a. Dequeue a node from the queue and print its va

b. Enqueue the left child of the dequeued node, if it exists.

c. Enqueue the right child of the dequeued node, if it exists.

d. Done.

The above steps implement a breadth-first search traversal of the BST, which is also known as ~~level by level, starting from~~ level order traversal. This algorithm will print out the nodes of the BST level by level, starting from the root node and going down to the leaf node.

Code:

```
/*   Ass 3: Create BST ,its mirror image ,display it
levelwise,display leaf nodes,find height of BST */


#include <iostream>
using namespace std;
struct  tnode{
    int info;   // node value
    struct tnode *left; //left child pointer
    struct tnode *right; //right child pointer
};
class bintree
{
    tnode *root;
    public:
         bintree()  //constructor
        {
        root =NULL;
        }
          int isEmpty() //function to check if tree is empty
          {
              if(root==NULL)
                  return 1;
              return 0;
          }
         void printGivenLevel(tnode* root, int level);
         void printLevelOrder(); //print levelwise
         void maxDepth1(); //prints height of tree wrapper
         void plnode(); //print leaf nodes wrapper
         void printLeafNodes(tnode* root);
         int  maxDepth(tnode* node);
         void insert(int val); //function for insertion of node in
tree
         void mirror1();
         void mirror(struct tnode *);//create mirror image wrapper
```

```
};


void bintree::plnode()
{


    cout<<endl<<"leaf nodes are:";
    printLeafNodes(root);
}


void bintree:: printLeafNodes(tnode *root)
{
    // if node is null, return
    if (!root)
        return;


    // if node is leaf node, print its data
    if (!root->left && !root->right)
    {
        cout << root->info << " ";
        return;
    }


    // if left child exists, check for leaf
    // recursively
    if (root->left)
       printLeafNodes(root->left);


    // if right child exists, check for leaf
    // recursively
    if (root->right)
       printLeafNodes(root->right);
}


void bintree::insert(int v)
{
    tnode *ctnode = new tnode;
    tnode *parent;
    ctnode->info =v;
    ctnode->left=NULL;
    ctnode->right=NULL;
    parent=NULL;
    if(isEmpty())
    {
        root=ctnode;
    }
    else
    {
        tnode *p=root;
        while(p!=NULL)
        {
            parent =p;
            if(v>p->info)
                p=p->right;
            else
                p=p->left;


        }
        if(v<parent->info)
            parent->left=ctnode;
        else
            parent->right=ctnode;
    }


    }


void bintree::printGivenLevel(tnode* root, int level)
```

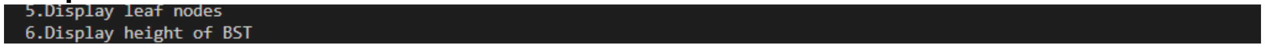
```
{
    if (root == NULL)
        return;
    if (level == 1)
        cout<<root->info<<" ";
    else if (level > 1) {
        printGivenLevel(root->left, level - 1);
        printGivenLevel(root->right, level - 1);
    }
}


void bintree::printLevelOrder()
{
    int h = maxDepth(root);
    int i;
    cout<<endl<<"level wise"<<endl;
    for (i = 1; i <= h; i++) {
        printGivenLevel(root, i);
        cout<<endl;
    }
}








void bintree::  maxDepth1()
{
int d = maxDepth(root);
cout<<"height:"<<d;


}



    int bintree:: maxDepth(tnode* node)
```

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


        /* use the larger one */
        if (lDepth > rDepth)
            return (lDepth + 1);
        else
            return (rDepth + 1);
    }
}




    void bintree:: mirror1()
    {



        cout<<endl<<"mirror image is :";
        mirror(root);
        printLevelOrder();
    }
    void bintree:: mirror(struct tnode *node)
    {
        if(node==NULL)
        {
            return;
        }
        else
        {
            struct tnode *temp;
            mirror(node->left);
            mirror(node->right);
```

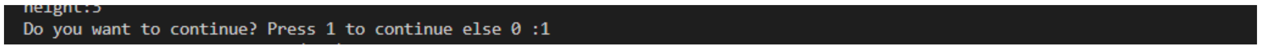
```
                temp=node->left;
                node->left=node->right;
                node->right=temp;
            }
        }



    int main() {
    bintree b;
    int ch,ch1,n,d;
    do{
            cout<<"1.Create Binary Search Tree (BST)"<<endl;
            cout<<"2.Insert a node in BST "<<endl;
            cout<<"3.Display BST levelwise"<<endl;
            cout<<"4.Create Mirror image of BST"<<endl;
            cout<<"5.Display leaf nodes"<<endl;
            cout<<"6.Display height of BST"<<endl;
            cout<<"Enter your choice : ";
            cin>>ch1;
            switch(ch1)
            {


                case 1: cout<<"How many nodes in BST : ";
                        cin>>n;
                        for(int i=0;i<n;i++)
                        {
                            cout<<"Enter data for node";
                            cin>>d;
                            b.insert(d);
                        }
                        break;
                case 2: cout<<"Enter data for node";
                        cin>>d;
                        b.insert(d);
                        break;
                case 3:
```

```
                                b.printLevelOrder();
                                break;
                        case 4:
                                b.mirror1();
                                break;
                        case 5:
                                b.plnode();
                                break;
                        case 6:
                                b.maxDepth1();
                                break;
                        default:
                                cout<<endl<<"Enter valid choice";
                                break;


                }
        cout<<endl<<"Do you want to continue? Press 1 to continue else 0
:";
        cin>>ch;


        }while(ch==1);
        return 0;
}
```

Output:



```
 5.Display leaf nodes
 6.Display height of BST
```



```
height:5
Do you want to continue? Press 1 to continue else 0 :1
```