Name-Harshita Totala Roll No-SEITA14
Div-A

Assignment-5
Binary search tree

AIM: Implement binary search tree and perform following operations: a) Insert (Handle insertion of duplicate entry)
b) Delete
c) Search
d) Display tree (Traversal)
e) Display - Depth of tree
f) Display - Mirror image
g) Create a copy
h) Display all parent nodes with their child nodes
i) Display leaf nodes
j) Display tree level wise

OBJECTIVE:

1. To understand the concept of binary search tree as a data structure.
2. Applications of BST.

THEORY:

1. Definition of binary search tree

A binary tree in which each internal node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stored in the right subtree of x are greater than or equal to x. This is called binary-search-tree property.

2. Illustrate the above operations graphically.

Searching

Searching a binary tree for a specific value can be a recursive or iterative process. This explanation covers a recursive method.We begin by examining the root node. If the tree is null, the value we are searching for does not exist in the tree. Otherwise, if the value equals the root, the search is successful. If the value is less than the root, search the left subtree. Similarly, if it is greater than the root, search the right subtree. This process is repeated until the value is found or the indicated subtree is null. If the searched value is not found before a null subtree is reached, then the item must not be present in the tree.

Insertion

Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or

left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root.

Deletion

There are three possible cases to consider:

- **Deleting a leaf (node with no children):** Deleting a leaf is easy, as we can simply remove it from the tree.
- **Deleting a node with one child:** Remove the node and replace it with its child.
- **Deleting a node with two children:** Call the node to be deleted N. Do not delete N. Instead, choose either its in-order successor node or its in-order predecessor node, R. Replace the value of N with the value of R, then delete R.

As with all binary trees, a node's in-order successor is the left-most child of its right subtree, and a node's in-order predecessor is the right-most child of its left subtree. In either case, this node will have zero or one children. Delete it according to one of the two simpler cases above.

ALGORITHM:

Define structure for Binary Tree (Mnemonics, Left Pointer, Right Pointer)

Insert Node:

Insert (Root, Node)
Root is a variable of type structure, represent Root of the Tree. Node is a variable of type structure ,represent new Node to insert in a tree.
Step 1: Repeat Steps 2,3 & 4 Until Node do not insert at appropriate position. Step 2: If
Node Data is less that Root Data & Root Left Tree is NULL Then
insert Node to Left.
Else Move Root to Left
Step 3: Else If Node Data is Greater that equal that Root Data & Root Right Tree is NULL Then
insert Node to Right.
Else Move Root to Right.
Step 4: Stop.

Search Node:

Search (Root, Mnemonics)
Root is a variable of type structure, represent Root of the Tree. Mnemonics is array of character. This function search Mnemonics in a Tree.

Step 1: Repeat Steps 2,3 & 4 Until Mnemonics Not find && Root != NULL Step 2: If Mnemonics Equal to Root Data Then
print message Mnemonics present.
Step 3: Else If Mnemonics Greater that Equal that Root Data Then Move Root to Right.
Step 4: Else Move Root to Left.
Step 5: Stop.

Delete Node:

Dsearch(Root, Mnemonics)

Root is a variable of type structure ,represent Root of the Tree. Mnemonics is array of character. Stack is an pointer array of type structure. PTree(Parent of Searched Node),Tree(Node to be deleted), RTree(Pointg to Right Tree),Temp are pointer variable of type structure; Step 1: Search Mnemonics in a Binary Tree Step 2: If Root = = NULL Then Tree is NULL Step 3:
Else //Delete Leaf Node

If Tree->Left = = NULL && Tree->Right = = NULL
Then a) If Root = = Tree Then Root = NULL;
a) If Tree is a Right Child PTree->Right=NULL;
Else PTree->Left=NULL;

Step 4: Else // delete Node with Left and Right children If
Tree->Left != NULL && Tree->Right != NULL Then a)
RTree=Temp=Tree->Right;
b) Do steps i && ii while Temp->Left !=NULL
i) RTree=Temp;
ii) Temp=Temp->Left;
c)RTree->Left=Temp->Right;
d) If Root == Tree//Delete Root Node
Root=Temp;
e) If Tree is a Right Child PTree->Right=Temp;
Else PTree->Left=Temp;
f) Temp->Left=Tree->Left;
g) If RTree!=Temp
Then Temp->Right = Tree->Right;

Step 5: Else //with Right child If
Tree->Right!= NULL
Then a) If Root==Tree Root = Root->Right;
b) If Tree is a Right Child PTree->Right=Tree->Right;
Else PTree->Left=Tree->Left;

Step 6: Else //with Left child If
Tree->Left != NULL
Then a) If Root==Tree Root = Root->Left;
b) If Tree is a Right Child PTree->Right=Tree->Left;
Else PTree->Left=Tree->Left; Step 7: Stop.

Display:

In order to display content of each node in exactly once, a BT can be traversed in either PreOrder, Post- Order, In-Order.
Here Pre-order Traversal is considered

1. Visit the root node R First
2. Then visit the left sub tree of R in Pre-Order fashion
3. Finally, visit the right sub tree of R in Pre-Order fashion

Depth of a Tree:

This function finds the depth of the linked binary tree.

1. Start from the ROOT node and store the address of the root node in **ptr.**
2. Check whether **ptr** contains NULL or not. If **ptr** is NULL then tree is empty.
3. Otherwise

a. Get the height of left sub tree, say leftHeight.
b. Get the height of right sub tree, say rightHeight.
c. Take the Max(leftHeight, rightHeight) and add 1 for the root and return. d. Call recursively.

Mirror Image:

Root is a variable of type structure ,represent Root of the Tree.
Queue is an pointer array of type structure. Front & Rear variable of type integer.
Mirror(Root)
Step 1: Queue[0]=Root;//Insert Root Node in a Queue Step 2:
Repeat Steps 3,4,5,6,7 & 8 Until Queue is Empty Step 3: Root
= Queue[Front++];
Step 4: Temp1 = Root->Left; Step 5: Root->Left
= Root->Right; Step 6:
Root->Right = Temp1; Step 7: If
Root->Left != NULL
Then Queue[Rear++] = Tree->Left;//insert Left SubTree
Step 8: If Root->Right != NULL

Then Queue[Rear++] = Root->Right;//insert Right SubTree Step 9:
Stop.

Create a Copy of a Tree:

This function will make a copy of the linked binary tree. The function should allocate memory for necessary nodes and copy respective contents in it.

1. Start from the ROOT node. The address of the root node will be in the **ptr.** Let newroot be the root of the new tree after the copy.
2. Check whether **ptr** contains NULL or not. If **ptr** is NULL then newroot = NULL
3. Otherwise

    (i) Allocate memory space for the new node
    (ii) Copy the data from current node from the old tree to node in the new tree.
    (iii) Traverse each node in the left subtrees from the old tree and repeat step (i) and (ii).
    (iv) Traverse each node in the right subtrees from the old tree and repeat step (i) and (ii).

4. Stop

Display Leaf-Nodes:

This function displays all the leaf nodes in the linked binary tree.

1. Start from the ROOT node and store the address of the root node in **ptr.**
2. Check whether **ptr** contains NULL or not. If **ptr** is NULL then tree is empty.
3. Otherwise

    a. Check whether the left child and right child of **ptr** are empty or not. If the left child and right child of **ptr** are NULL then display the content of the node.
    b. Traverse the left subtrees and the right subtrees and repeat step a for all the nodes.

4. Stop

Breath First Search(Levelwise Display):

Root is a variable of type structure ,represent Root of the Tree.
Queue is an pointer array of type structure. Front & Rear variable of type integer. BFS(Root)
Step 1:If Root == NULL Then Empty Tree;
Step 2: Else Queue[0] = Root;// insert Root of the Tree in a Queue

Step 3: Repeat Steps 4,5,6 & 7 Until Queue is Empty
Step 4: Tree=Queue[Front++]; //Remove Node From Queue Step
5: print Root Data
Step 6: If Root->Left != NULL
Then Queue[++Rear] = Tree->Left;//insert Left Subtree in a Queue Step
7: If Root->Right != NULL
Then Queue[++Rear] = Root->Right; //insert Left Subtree in a Queue Else
if Root->Right == NULL And Root->Left == NULL
Leaf++;//Number of Leaf Nodes Step
8: Stop.

Program-

```
#include <iostream>
#include<stdlib.h>
using namespace std;

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

node *insert(node *root, int val){
    if (root == NULL){
        node *temp;
        temp=new node;
        temp->data=val;
        temp->left=temp->right=NULL;
        return temp;
    }
    if (val < root->data){
        root->left = insert(root->left, val);
    }
    else{

        root->right = insert(root->right, val);
    }
    return root;
}

void inorder(node *root)
{
    if (root == NULL){
        return;
    }
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

node* inorderSucc(node* root){
    node* curr=root;
    while(curr && curr->left!=NULL){
        curr=curr->left;
```

```
    }
    return curr;
}

node *delete_ele(node *root, int key){
    if(key<root->data){
        root->left=delete_ele(root->left, key);
    }
    else if(key>root->data){
        root->right=delete_ele(root->right,key);
    }

    else{
        if(root->left==NULL){
            node* temp=root->right;
            free(root);
            return temp;
        }
        else if(root->right==NULL){
            node* temp=root->left;
            free(root);
            return temp;
        }
        node* temp=inorderSucc(root->right);
        root->data=temp->data;
        root->right=delete_ele(root->right, temp->data);
    }
    return root;
}

node* search(node* root, int val){
    if(root==NULL){
        return NULL;
    }
    if(val>root->data){
        return search(root->right, val);
    }
    else if(val<root->data){
        return search(root->left, val);
    }
    else{
        return root;
    }
}
```

```
void mirror(node* root){
    if(root==NULL){
        return;
    }
    else{
        struct node *temp;
        mirror(root->left);
        mirror(root->right);
        swap(root->left,root->right);
    }
}

node *copy(node *root){
    node *temp=NULL;
    if(root!=NULL){
        temp=new node();
        temp->data=root->data;
        temp->left=copy(root->left);
        temp->right=copy(root->right);
    }
    return temp;
}

void leafNodes(node* root){
    if(root==NULL){
        return;
    }
    if(!root->left && !root->right){
        cout<<root->data<<" ";
        return;
    }
    if(root->right)
       leafNodes(root->right);
    if(root->left)
       leafNodes(root->left);
}

int Height(node* root){
    if(root==NULL){
        return 0;
    }
    int lheight=Height(root->left);
```

```
    int rheight=Height(root->right);
    return max(lheight,rheight)+1;
}

node *Min(node *root){
    if(root==NULL){
        return NULL;
    }
    if(root->left)
        return Min(root->left);
    else
        return root;
}

node *Max(node *root){
    if(root==NULL){
        return NULL;
    }
    if(root->right)
       return Max(root->right);
    else
        return root;
}

int main()
{
    cout<<"\n** Welocme to Binary Search Tree **"<<endl;
    node *root=NULL, *temp;
    int ch;
    do{
        cout<<"\n---------------\n  Main Menu\n---------------"<<endl;
        cout<<"1) Insert" << endl;
        cout<<"2) Delete" << endl;
        cout<<"3) Search" << endl;
        cout<<"4) Create copy "<<endl;
        cout<<"5) Display leaf nodes "<<endl;
        cout<<"6) Height of tree"<<endl;
        cout<<"7) Find minimum"<<endl;
        cout<<"8) Find maximum"<<endl;
        cout<<"9) Mirror image"<<endl;
        cout<<"10) Exit"<<endl;
        cout<<"-----------------"<<endl;
        cout<<"\nEnter your choice: ";
        cin>>ch;
```

```
        switch (ch){
        case 1:
            cout << "Enter the element to be insert :  ";
            cin >> ch;
            root= insert(root, ch);
            cout << " Elements in Tree are ->  ";
            inorder(root);
            break;
        case 2:
            cout<<"Enter the element to be deleted :  ";
            cin>>ch;
            root=delete_ele(root, ch);
            cout<<"* Element deleted successfully *"<<endl;
            cout<<"Current elements in Tree are :  ";
            inorder(root);
            break;
        case 3:
            cout<<"Enter the element to search :   ";
            cin>>ch;
            temp=search(root, ch);
            if(temp==NULL){
                cout<<"*Element not found*";
            }
            else{
                cout<<"*Element found*";
            }
            break;
        case 4:
            cout<<"Copy of the Tree is :   ";
            root=copy(root);
            inorder(root);
            break;
        case 5:
            cout<<"The leaf nodes are :  ";
            leafNodes(root);
            break;
        case 6:
            cout<<"Height of the Tree is:  "<<Height(root);
            break;
        case 7:
            temp=Min(root);
            cout<<"\nMinimum element is :  "<<temp->data;
            break;
        case 8:
```

```
            temp=Max(root);
            cout<<"\nMaximum element is:   "<<temp->data;
            break;
        case 9:
            mirror(root);
            cout<<"Mirror image is:  ";
            inorder(root);
            break;
        case 10:
            cout<<"Thank you !! Do Visit Again";
        default:
            cout<<"\nInvalid choice !! Please enter your choice again";
        }
    }while(ch!=10);
}
```

Output-

```
** Welocme to Binary Search Tree **

---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 1
Enter the element to be insert :  4
 Elements in Tree are ->  4
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 1
Enter the element to be insert :  8
 Elements in Tree are ->  4 8
---------------
  Main Menu
---------------
```

1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 1
Enter the element to be insert : 3
Elements in Tree are -> 3 4 8

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 3
Enter the element to search : 3
Element found

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree

7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 4
Copy of the Tree is : 3 4 8

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 5
The leaf nodes are : 8 3

Main Menu
---
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 6
Height of the Tree is: 2

```
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 7

Minimum element is :  3
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit
------------------

Enter your choice: 8

Maximum element is:   8
---------------
  Main Menu
---------------
1) Insert
2) Delete
3) Search
```

4) Create copy
5) Display leaf nodes
6) Height of tree
7) Find minimum
8) Find maximum
9) Mirror image
10) Exit

Enter your choice: 9
Mirror image is: 8 4 3

Conclusion: Thus we had Implemented binary search tree and performed following operations:

a) Insert (Handle insertion of duplicate entry)
b) Delete
c) Search
d) Display tree (Traversal)
e) Display - Depth of tree
f) Display - Mirror image
g) Create a copy
h) Display all parent nodes with their child nodes
i) Display leaf nodes
j) Display tree level wise