AIM:-

Assignment 12

For a BST perform the following operations:

1. Search

2. Level-wise Display

3. Delete

OBJECTIVE:-

Searching for a specific element in Binary Search Tree and to display each node level-wise and to delete a specific node from the BST.

Theory:-

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root’s key, we recur for right subtree of root node. Otherwise we recur for left subtree.

If the node doesn’t have children then just delete the reference for this node from its parent and recycle the memory.

There are 3 cases for deletion.

Algorithm :-

Deletion:-

1.Node to be removed has no children.

This case is quite simple. Algorithm sets corresponding link of the parent to NULL and disposes the node.

2.Node to be removed has one child.

It this case, node is cut from the tree and algorithm links single child (with it's subtree) directly to the parent of the removed node.

3.Node to be removed has two children.

This is the most complex case. To solve it, let us see one useful BST property first. We are going to use the idea, that the same set of values may be represented as different binary-search trees

Search Operation

Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

SOURCE CODE:

# include <iostream>

# include <cstdlib>

using namespace std;

struct node //NODE

{

int info;

struct node \*left;

struct node \*right;

}\*root;

class BST //Class

{

public:

void find(int, node \*\*, node \*\*);

void insert(node \*, node \*);

void search(int a, node \*p);

void del(int);

void case\_a(node \*,node \*);

void case\_b(node \*,node \*);

void case\_c(node \*,node \*);

void display(node \*, int);

BST()

{

root = NULL;

}

};

int main() //MAIN

{

int choice, num,st;

BST bst;

node \*temp;

while (1)

{

cout<<"-----------------"<<endl;

cout<<"Operations on BST"<<endl;

cout<<"-----------------"<<endl;

cout<<"1.Insert Element "<<endl;

cout<<"2.Delete Element "<<endl;

cout<<"3.Display"<<endl;

cout<<"4.Search"<<endl;

cout<<"5.Quit."<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

temp = new node;

cout<<"Enter the number to be inserted : ";

cin>>temp->info;

bst.insert(root, temp);

break;

case 2:

if (root == NULL)

{

cout<<"Tree is empty, nothing to delete"<<endl;

break;

}

cout<<"Enter the number to be deleted : ";

cin>>num;

bst.del(num);

break;

case 3:

cout<<"Display BST:"<<endl;

bst.display(root,1);

cout<<endl;

break;

case 4:

if(root == NULL)

{

cout<<"Tree is empty.\n";

break;

}

cout<<"Enter search term.\n";

cin>>st;

bst.search(st,root);

break;

case 5:

exit(1);

default:

cout<<"Wrong choice"<<endl;

}

}

}

void BST::search(int a,node \*pt){

if(a == pt->info)

{

cout<<"Element Found\n";

return;

}

if(a <pt->info){

search(a,pt->left);

}

else if(a>pt->info){

search(a,pt->right);

}

else if(pt->left==NULL && pt->right==NULL&&pt->info!=a){

cout<<"Element not found.\n";

return;

}

}

/\*

\* Find Element in the Tree

\*/

void BST::find(int item, node \*\*par, node \*\*loc)

{

node \*ptr, \*ptrsave;

if (root == NULL)

{

\*loc = NULL;

\*par = NULL;

return;

}

if (item == root->info)

{

\*loc = root;

\*par = NULL;

return;

}

if (item < root->info)

ptr = root->left;

else

ptr = root->right;

ptrsave = root;

while (ptr != NULL)

{

if (item == ptr->info)

{

\*loc = ptr;

\*par = ptrsave;

return;

}

ptrsave = ptr;

if (item < ptr->info)

ptr = ptr->left;

else

ptr = ptr->right;

}

\*loc = NULL;

\*par = ptrsave;

}

/\*

\* Inserting Element into the Tree

\*/

void BST::insert(node \*tree, node \*newnode)

{

if (root == NULL)

{

root = new node;

root->info = newnode->info;

root->left = NULL;

root->right = NULL;

cout<<"Root Node is Added"<<endl;

return;

}

if (tree->info == newnode->info)

{

cout<<"Element already in the tree"<<endl;

return;

}

if (tree->info > newnode->info)

{

if (tree->left != NULL)

{

insert(tree->left, newnode);

}

else

{

tree->left = newnode;

(tree->left)->left = NULL;

(tree->left)->right = NULL;

cout<<"Node Added To Left"<<endl;

return;

}

}

else

{

if (tree->right != NULL)

{

insert(tree->right, newnode);

}

else

{

tree->right = newnode;

(tree->right)->left = NULL;

(tree->right)->right = NULL;

cout<<"Node Added To Right"<<endl;

return;

}

}

}

/\*

\* Delete Element from the tree

\*/

void BST::del(int item)

{

node \*parent, \*location;

if (root == NULL)

{

cout<<"Tree empty"<<endl;

return;

}

find(item, &parent, &location);

if (location == NULL)

{

cout<<"Item not present in tree"<<endl;

return;

}

if (location->left == NULL && location->right == NULL)

case\_a(parent, location);

if (location->left != NULL && location->right == NULL)

case\_b(parent, location);

if (location->left == NULL && location->right != NULL)

case\_b(parent, location);

if (location->left != NULL && location->right != NULL)

case\_c(parent, location);

free(location);

}

/\*

\* Case A

\*/

void BST::case\_a(node \*par, node \*loc )

{

if (par == NULL)

{

root = NULL;

}

else

{

if (loc == par->left)

par->left = NULL;

else

par->right = NULL;

}

}

/\*

\* Case B

\*/

void BST::case\_b(node \*par, node \*loc)

{

node \*child;

if (loc->left != NULL)

child = loc->left;

else

child = loc->right;

if (par == NULL)

{

root = child;

}

else

{

if (loc == par->left)

par->left = child;

else

par->right = child;

}

}

/\*

\* Case C

\*/

void BST::case\_c(node \*par, node \*loc)

{

node \*ptr, \*ptrsave, \*suc, \*parsuc;

ptrsave = loc;

ptr = loc->right;

while (ptr->left != NULL)

{

ptrsave = ptr;

ptr = ptr->left;

}

suc = ptr;

parsuc = ptrsave;

if (suc->left == NULL && suc->right == NULL)

case\_a(parsuc, suc);

else

case\_b(parsuc, suc);

if (par == NULL)

{

root = suc;

}

else

{

if (loc == par->left)

par->left = suc;

else

par->right = suc;

}

suc->left = loc->left;

suc->right = loc->right;

}

/\*

\* Display Tree Structure

\*/

void BST::display(node \*ptr, int level)

{

int i;

if (ptr != NULL)

{

display(ptr->right, level+1);

cout<<endl;

if (ptr == root)

cout<<"Root->: ";

else

{

for (i = 0;i < level;i++)

cout<<" ";

}

cout<<ptr->info;

display(ptr->left, level+1);

}

}