Submitted By

Priyanka Rana

CSE 2C

1900300100164

Q-Program to implement binary search tree and implement addition and deletion of a node in the tree

#include <iostream>

#include <cstdlib>

using namespace std;

struct nod //node declaration

{

int info;

struct nod \*l;

struct nod \*r;

} \* r;

class BST

{

public: //functions declaration

void search(nod \*, int);

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

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

void del(int);

void casea(nod \*, nod \*);

void caseb(nod \*, nod \*);

void casec(nod \*, nod \*);

void preorder(nod \*);

void inorder(nod \*);

void postorder(nod \*);

void show(nod \*, int);

BST()

{

r = NULL;

}

};

void BST::find(int i, nod \*\*par, nod \*\*loc) //find the position of the item

{

nod \*ptr, \*ptrsave;

if (r == NULL)

{

\*loc = NULL;

\*par = NULL;

return;

}

if (i == r->info)

{

\*loc = r;

\*par = NULL;

return;

}

if (i < r->info)

ptr = r->l;

else

ptr = r->r;

ptrsave = r;

while (ptr != NULL)

{

if (i == ptr->info)

{

\*loc = ptr;

\*par = ptrsave;

return;

}

ptrsave = ptr;

if (i < ptr->info)

ptr = ptr->l;

else

ptr = ptr->r;

}

\*loc = NULL;

\*par = ptrsave;

}

void BST::search(nod \*root, int data) //searching

{

int depth = 0;

nod \*temp = new nod;

temp = root;

while (temp != NULL)

{

depth++;

if (temp->info == data)

{

cout << "\nData found at depth: " << depth << endl;

return;

}

else if (temp->info > data)

temp = temp->l;

else

temp = temp->r;

}

cout << "\n Data not found" << endl;

return;

}

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

{

if (r == NULL)

{

r = new nod;

r->info = newnode->info;

r->l = NULL;

r->r = 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->l != NULL)

{

insert(tree->l, newnode);

}

else

{

tree->l = newnode;

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

(tree->l)->r = NULL;

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

return;

}

}

else

{

if (tree->r != NULL)

{

insert(tree->r, newnode);

}

else

{

tree->r = newnode;

(tree->r)->l = NULL;

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

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

return;

}

}

}

void BST::del(int i)

{

nod \*par, \*loc;

if (r == NULL)

{

cout << "Tree empty" << endl;

return;

}

find(i, &par, &loc);

if (loc == NULL)

{

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

return;

}

if (loc->l == NULL && loc->r == NULL)

{

casea(par, loc);

cout << "item deleted" << endl;

}

if (loc->l != NULL && loc->r == NULL)

{

caseb(par, loc);

cout << "item deleted" << endl;

}

if (loc->l == NULL && loc->r != NULL)

{

caseb(par, loc);

cout << "item deleted" << endl;

}

if (loc->l != NULL && loc->r != NULL)

{

casec(par, loc);

cout << "item deleted" << endl;

}

free(loc);

}

void BST::casea(nod \*par, nod \*loc)

{

if (par == NULL)

{

r = NULL;

}

else

{

if (loc == par->l)

par->l = NULL;

else

par->r = NULL;

}

}

void BST::caseb(nod \*par, nod \*loc)

{

nod \*child;

if (loc->l != NULL)

child = loc->l;

else

child = loc->r;

if (par == NULL)

{

r = child;

}

else

{

if (loc == par->l)

par->l = child;

else

par->r = child;

}

}

void BST::casec(nod \*par, nod \*loc)

{

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

ptrsave = loc;

ptr = loc->r;

while (ptr->l != NULL)

{

ptrsave = ptr;

ptr = ptr->l;

}

suc = ptr;

parsuc = ptrsave;

if (suc->l == NULL && suc->r == NULL)

casea(parsuc, suc);

else

caseb(parsuc, suc);

if (par == NULL)

{

r = suc;

}

else

{

if (loc == par->l)

par->l = suc;

else

par->r = suc;

}

suc->l = loc->l;

suc->r = loc->r;

}

void BST::preorder(nod \*ptr)

{

if (r == NULL)

{

cout << "Tree is empty" << endl;

return;

}

if (ptr != NULL)

{

cout << ptr->info << " ";

preorder(ptr->l);

preorder(ptr->r);

}

}

void BST::inorder(nod \*ptr) //inorder traversal

{

if (r == NULL)

{

cout << "Tree is empty" << endl;

return;

}

if (ptr != NULL)

{

inorder(ptr->l);

cout << ptr->info << " ";

inorder(ptr->r);

}

}

void BST::postorder(nod \*ptr) //postorder traversal

{

if (r == NULL)

{

cout << "Tree is empty" << endl;

return;

}

if (ptr != NULL)

{

postorder(ptr->l);

postorder(ptr->r);

cout << ptr->info << " ";

}

}

void BST::show(nod \*ptr, int level) //print the tree

{

int i;

if (ptr != NULL)

{

show(ptr->r, level + 1);

cout << endl;

if (ptr == r)

cout << "Root→: ";

else

{

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

cout << " ";

}

cout << ptr->info;

show(ptr->l, level + 1);

}

}

int main(){

int c, n, item;

BST bst;

nod \*t;

while (1)

{

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

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

cout << "3.Search Element" << endl;

cout << "4.Inorder Traversal" << endl;

cout << "5.Preorder Traversal" << endl;

cout << "6.Postorder Traversal" << endl;

cout << "7.Display the tree" << endl;

cout << "8.Quit" << endl;

cout << "Enter your choice : ";

cin >> c;

switch (c)

{

case 1:

t = new nod;

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

cin >> t->info;

bst.insert(r, t);

break;

case 2:

if (r == NULL)

{

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

continue;

}

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

cin >> n;

bst.del(n);

break;

case 3:

cout << "Search:" << endl;

cin >> item;

bst.search(r, item);

break;

case 4:

cout << "Inorder Traversal of BST:" << endl;

bst.inorder(r);

cout << endl;

break;

case 5:

cout << "Preorder Traversal of BST:" << endl;

bst.preorder(r);

cout << endl;

break;

case 6:

cout << "Postorder Traversal of BST:" << endl;

bst.postorder(r);

cout << endl;

break;

case 7:

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

bst.show(r, 1);

cout << endl;

break;

case 8:

exit(1);

default:

cout << "Wrong choice" << endl; }

}

}