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<cstdio>

using namespace std;

typedef struct BST

{

int data;

struct BST \*left, \*right;

}\*node;

int height(node root)

{

int l = root->left ? height(root->left) : 0;

int r = root->right ? height(root->right) : 0;

if(l>r)

return (1 + l);

else

return (1 + r);

}

node accept(int data)

{

node root;

root=new(struct BST);

root->data=data;

root->left=NULL;

root->right=NULL;

return (root);

}

node create(node root,int data)

{

if (root == NULL)

return accept(data);

if (data < root->data)

root->left = create(root->left, data);

else if (data > root->data)

root->right = create(root->right, data);

return root;

}

node search(node root,int num)

{

if (root == NULL || root->data == num)

return root;

if (root->data < num)

return search(root->right, num);

return search(root->left, num);

}

void printGivenLevel(node root, int level)

{

if (root == NULL)

return;

if (level == 1)

cout<<root->data<<" ";

else if (level > 1)

{

printGivenLevel(root->left, level-1);

printGivenLevel(root->right, level-1);

}

}

void printLevelOrder(node root)

{

int i,h=height(root);

for (i=1; i<=h; i++)

{

printGivenLevel(root, i);

cout<<"\n";

}

}

node minValueNode(node root)

{

node current = root;

while (current && current->left != NULL)

current = current->left;

return current;

}

node deleteNode(node root, int num1)

{

if (root == NULL)

return root;

if (num1 < root->data)

root->left = deleteNode(root->left, num1);

else if (num1 > root->data)

root->right = deleteNode(root->right, num1);

else

{

if (root->left == NULL)

{

node temp = root->right;

delete root;

return temp;

}

else if (root->right == NULL)

{

node temp = root->left;

delete root;

return temp;

}

node temp = minValueNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

int main()

{

int num,num1;

node root = NULL;

root = create(root, 50);

create(root, 30);

create(root, 20);

create(root, 40);

create(root, 70);

create(root, 60);

create(root, 80);

cout<<"Enter the element you want to search in the binary search tree\n";

cin>>num;

node a=NULL;

a=search(root,num);

if(a!=NULL)

cout<<"The element is found\n";

else

cout<<"The element was not found\n";

printLevelOrder(root);

cout<<"Enter the data you want to delete\n";

cin>>num1;

root=deleteNode(root,num1);

printLevelOrder(root);

return 0;

}

OUTPUT:-

Enter the element you want to search in the binary search tree

30

The element is found

50

30 70

20 40 60 80

Enter the data you want to delete

50

60

30 70

20 40 80

CONCLUSION:-

Binary search trees are a very powerful data structure to have in your programming language.

If done right, handling large amount of data becomes easier and quicker.