ADVANCED DATA STRUCTURE

GROUP A

ASSIGNMENT 1

YEAR: 2017-18

COLLEGE: VIIT

Title:

Binary Search Tree

Problem Statement:

Beginning with an empty binary search tree, construct binary search tree by inserting the

values in the order given. After constructing a binary tree -

i. Insert new node

ii. Find number of nodes in longest path

iii. Minimum data value found in the tree

iv. Change a tree so that the roles of the left and right pointers are swapped at

every node.

v. Search a value

Objective:

To understand and implement binary search tree and perform basic operations like insert

node, search node, mirror image of tree, finding minimum value and no of nodes in longest

path.

Software And Hardware Requirement:

1. 64-bit Open source Linux or its derivative.

2. Open Source C++ Programming tool like G++/GCC.

Theory:

A binary search tree (BST) is a totally ordered binary tree. The BST’s total ordering does

the heap’s partial ordering one better; not only is there a relationship between a BST node

and its children, but there is also a definite relationship between the children. In a BST, the

value of a node’s left child is less than the value of the node itself, and the value of a node’s

right child is greater than or equal to the value of the node. Consequently, the value of a

node’s left child is always less than the value of its right child.

Insertion into a BST

Let us insert into a BST the following values, in the order given: 5, 3, 4, 8, 1, 6, 4. Since

the tree is initially empty, the first value, 5, becomes the new tree’s root. The next value, 3,

is less than 5, and so the 3 becomes 5’s left child. The third value, 4, is less than 5, which

means that the 4 must be placed somewhere in the root’s left subtree. Thus we move to

5’s left child, the node containing 3. Since 4 is greater than 3 and the node containing 3

has no right child, a new node containing 4 becomes 3’s right child. The next value, 8, is

greater than the value at the root, and so the 8 must be placed somewhere in the root’s right

subtree. Since the root does not yet have a right child, a new node containing 8 becomes the

root’s right child. The next value to be inserted is 1. We again start at the root and move

left or right depending on the relationship between the value to be inserted and the value at

the current node. Since 1 is less than 5, we move to 5’s left child. The 1 is also less than 3,

the value of the current node. That node has no left child, and so a new node containing 1

is placed as 3’s left child. The sixth value, 6, is greater than the root value, and so we move

to the root’s right child, 8. Since 6 is less than 8 and 8 has no left child, the 6 becomes 8’s

left child. Insertion of the last value, a second 4, requires that we go left and then right,

arriving at the first 4. Since 4 is greater than or equal to 4, the second 4 becomes the right

child of the first. Thus we arrive at the final tree

Finding an Element of a BST

Searching a BST involves the same navigation that is employed for insertion. A search begins

at the root, and left or right ”turns” are made until the target element is located or until

the search encounters an empty child, in which case the target element is not a member of

the collection.

Time Analysis of the Fundamental BST Operations

A tree is said to be balanced if it has the least possible height. The height of a balanced

binary tree with n nodes is approximately lg(n). Since a BST insertion, search, or removal

may require navigation to the deepest node of the tree, the BST insert, find, and remove

operations are all in O(lg(n)) for balanced, or nearly balanced, BSTs. In the worst case,

however, operations insert, find, and remove are in O(n), for in the worst case a BST is

simply a list.

BST Applications

The most natural application of the BST is as a dictionary

Algorithm:

Search Operation in BST

Step 1: Read the search element from the user

Step 2: Compare, the search element with the value of root node in the tree.

2

Step 3: If both are matching, then display "Given node found!!!" and terminate the function

Step 4: If both are not matching, then check whether search element is smaller or larger than that node value.

Step 5: If search element is smaller, then continue the search process in left subtree.

Step 6: If search element is larger, then continue the search process in right subtree.

Step 7: Repeat the same until we found exact element or we completed with a leaf node

Step 8: If we reach to the node with search value, then display "Element is found" and terminate the function.

Step 9: If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

Insertion Operation in BST

Step 1: Create a newNode with given value and set its left and right to NULL.

Step 2: Check whether tree is Empty.

Step 3: If the tree is Empty, then set set root to newNode.

Step 4: If the tree is Not Empty, then check whether value of newNode is smaller or larger than the node (here it is root node).

Step 5: If newNode is smaller than or equal to the node, then move to its left child. If newNode is larger than the node, then move to its right child.

Step 6: Repeat the above step until we reach to a leaf node (e.i., reach to NULL).

Step 7: After reaching a leaf node, then isert the newNode as left child if newNode is smaller or equal to that leaf else insert it as right child.

Code :

/\*

\* BSTree.cpp

\*

\* Created on: 12 Jan 2017

\* Author: Pratik Gare

\*/

#include<iostream>

using namespace std;

typedef struct node

{

int data;

struct node \*left;

struct node \*right;

}node;

class Tree

{

private:

int val;

char ch;

node \*head,\*New,\*temp;

public:

node \*Create\_Node();

node \*Create\_BST(node \*root);

3

void Inorder(node \*root);

void Insert\_Node(node \*root);

void Mirror(node \*root);

void Search(node \*root);

void MinEle(node \*root);

void MaxEle(node \*root);

int LongPath(node \*root);

void Mirror\_NC(node \*root);

};

node \*Tree::Create\_Node()

{

cout<<"\nEnter Node-Data : ";

cin>>val;

New=new node;

New->data=val;

New->left=NULL;

New->right=NULL;

return New;

}

node \*Tree::Create\_BST(node \*root)

{

char ch1;

int cnt=0;

do

{

New=Create\_Node();

cnt++;

if(root==NULL)

{

root=New;

cout<<"\nRoot Node Created....!!!";

}

else

{

temp=root;

while(1)

{

if(temp->data < New->data)

{

if(temp->right==NULL)

{

temp->right=New;

cout<<New->data<<" Added to right of "<<temp->data;

break;

4

}

else

temp=temp->right;

}

else if(temp->data > New->data)

{

if(temp->left==NULL)

{

temp->left=New;

cout<<New->data<<" Added to left of "<<temp->data;

break;

}

else

temp=temp->left;

}

else

{

cout<<"\nSame Data Not Allowed...!!!\n";

break;

}

}

}

cout<<"\nDo you want to Add More(y/n) : ";

cin>>ch1;

}while(ch1==’y’|| ch1==’Y’);

cout<<"\nTotal Nodes In Tree : "<<cnt;;

return root;

}

void Tree::Insert\_Node(node \*root)

{

New=Create\_Node();

if(root==NULL)

{

root=New;

cout<<"\nRoot Node Created....!!!";

}

else

{

temp=root;

while(1)

{

if(temp->data < New->data)

{

if(temp->right==NULL)

{

5

temp->right=New;

cout<<New->data<<" Added to right of "<<temp->data;

break;

}

else

temp=temp->right;

}

else if(temp->data > New->data)

{

if(temp->left==NULL)

{

temp->left=New;

cout<<New->data<<" Added to left of "<<temp->data;

break;

}

else

temp=temp->left;

}

else

{

cout<<"\nSame Data Not Allowed...!!!\n";

break;

}

}

}

}

void Tree::Inorder(node \*root)

{

if(root==NULL)

return ;

if(root!=NULL)

{

Inorder(root->left);

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

Inorder(root->right);

}

}

/\*void Tree::Mirror(node \*root)

{

if(root==NULL)

return ;

else

{

Mirror(root->left);

Mirror(root->right);

// Swapping The left n right //

6

temp=root->left;

root->left=root->right;

root->right=temp;

}

}\*/

void Tree::Mirror\_NC(node \*root)

{

if(root==NULL)

return ;

else

{

// Swapping The left n right //

temp=root->left;

root->left=root->right;

root->right=temp;

}

node \*temp1;

temp1=root->left;

while(temp1!=NULL)

{

temp=temp1->left;

temp1->left=temp1->right;

temp1->right=temp;

temp1=temp1->left;

}

node \*temp2;

temp2=root->right;

while(temp2!=NULL)

{

temp=temp2->left;

temp2->left=temp2->right;

temp2->right=temp;

temp2=temp2->right;

}

}

void Tree::Search(node \*root)

{

int key;

int flag=0;

cout<<"\nEnter Data To Find : ";

cin>>key;

if(root->data==key)

{

7

cout<<"\nData Found...!!!";

flag=1;

return ;

}

else

{

temp=root;

while(temp!=NULL)

{

if(temp->data==key)

{

cout<<"\nData Found...!!!";

return ;

}

if(temp->data < key)

temp=temp->right;

else if(temp->data > key)

temp=temp->left;

};

}

if(flag==0)

{

cout<<"\nData Not Found...!!!";

return ;

}

}

void Tree::MinEle(node \*root)

{

temp=root;

while(temp->left!=NULL)

temp=temp->left;

cout<<"\nMinimum Element : "<<temp->data<<endl;

temp=root;

while(temp->right!=NULL)

temp=temp->right;

cout<<"\nMaximum Element : "<<temp->data<<endl;

}

/\*void Tree::MinEle(node \*root)

{

temp=root;

if(temp->left!=NULL)

MinEle(temp->left);

cout<<"\nMinimum Element : "<<temp->data<<endl;

}

void Tree::MaxEle(node \*root)

{

temp=root;

if(temp->right!=NULL)

MaxEle(temp->right);

cout<<"\nMaximum Element : "<<temp->data<<endl;

}\*/

int Tree::LongPath(node \*root)

{

int cntL,cntR;

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

return 0;

else

{

cntL=LongPath(root->left);

cntR=LongPath(root->right);

if(cntR > cntL)

return (cntR);

else

return (cntL);

}

}

int main()

{

int ch;

Tree b;

node \*root=NULL;

do

{

cout<<"\n\n\t MENU \n";

cout<<"\n1.Create BST\n2.Insert Node\n3.Search Node\n4.Mirror Image\n5.MinMax Element\n6.Longest Path\n0.Exit\n";

cout<<"\n\tEnter your choice : ";

cin>>ch;

switch(ch)

{

case 0:

return 0;

case 1:

root=b.Create\_BST(root);

cout<<"\nInorder Traversal : ";

b.Inorder(root);

break;

case 2:

b.Insert\_Node(root);

cout<<"\nInorder Traversal : ";

b.Inorder(root);

break;

case 3:

b.Search(root);

9

break;

case 4:

//b.Mirror(root);

//cout<<"\nMirror BST Traversal : ";

b.Mirror\_NC(root);

cout<<"\nMirror BST Traversal(NC) : ";

b.Inorder(root);

break;

case 5:

b.MinEle(root);

//b.MaxEle(root);

break;

case 6:

int height=b.LongPath(root);

cout<<"\nLongest Path : "<<height<<endl;

break;

}

}while(1);

return 0;

}

Output :

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 1

Enter Node-Data : 200

Root Node Created....!!!

Do you want to Add More(y/n) : y

Enter Node-Data : 100

100 Added to left of 200

Do you want to Add More(y/n) : y

10

Enter Node-Data : 220

220 Added to right of 200

Do you want to Add More(y/n) : y

Enter Node-Data : 250

250 Added to right of 220

Do you want to Add More(y/n) : y

Enter Node-Data : 210

210 Added to left of 220

Do you want to Add More(y/n) : y

Enter Node-Data : 100

Same Data Not Allowed...!!!

Do you want to Add More(y/n) : y

Enter Node-Data : 150

150 Added to right of 100

Do you want to Add More(y/n) : y

Enter Node-Data : 90

90 Added to left of 100

Do you want to Add More(y/n) : y

Enter Node-Data : 4

4 Added to left of 90

Do you want to Add More(y/n) : n

Total Nodes In Tree : 9

Inorder Traversal : 4 90 100 150 200 210 220 250

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 2

Enter Node-Data : 10

10 Added to right of 4

Inorder Traversal : 4 10 90 100 150 200 210 220 250

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 3

Enter Data To Find : 100

Data Found...!!!

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 3

Enter Data To Find : 545

Data Not Found...!!!

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 4

Mirror BST Traversal(NC) : 250 220 210 200 150 100 90 10 4

MENU

1.Create BST

2.Insert Node

3.Search Node

4.Mirror Image

5.MinMax Element

6.Longest Path

0.Exit

Enter your choice : 5

Minimum Element : 250

Maximum Element : 4

conclusion

Through this assignment we understood implementation of BST and perform basic oper-

ations like insert node, search node,mirror image of tree,finding minimum value and no of

nodes in longest path