ADVANCED DATA STRUCTURE

GROUP A

ASSIGNMENT 2

BATCH B1

YEAR: 2017-18

COLLEGE: VIIT

DATE OF COMPLETION : 30/03/2018

**Aim :**

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.

1 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

2 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.

3 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.

4 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. 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 :**

/\*

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords,deleting keywords, updating values of any entry.

Provide facility to display whole data sorted in ascending/ Descending order.

Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.

\*/

#include<iostream>

#include<string.h>

using namespace std;

int flag=1,i=0;

char key[10];

class node

{

char keyw[10];

char m[100];

node \*right;

node \*left;

friend class dicn;

};

class dicn

{

node \*root,\*temp,\*curr,\*New,\*t;

public:

int create();

void display(node \*);

node \* search(node \*);

void update(node \*);

node \* del(node \*,char \*);

node \* ret\_r(){

return root;

}

node \* findmax(node \* temp){

while(temp->right!=NULL)

{

temp=temp->right;

}

return temp;

}

};

int dicn :: create(){

cout<<endl;

New= new node;

cout<<"Enter keyword : ";

ws(cin);

cin.getline(New->keyw,10);

cout<<"Enter meaning : ";

cin.getline(New->m,100);

New->right=NULL;

New->left=NULL;

if(flag==1)

{

root=New;

temp=root;

flag=0;

}

else

{

temp=root;

while(1)

{

if(strcmp(temp->keyw,New->keyw)>0)

{

if(temp->left==NULL)

{

temp->left=New;

temp=New;

break;

}

else

{

temp=temp->left;

}

}

if(strcmp(temp->keyw,New->keyw)<0)

{

if(temp->right==NULL)

{

temp->right=New;

temp=New;

break;

}

else

{

temp=temp->right;

}

}

}

}

}

void dicn :: display(node \*root){

if(root!=NULL)

{

display(root->left);

cout<<root->keyw<<"\t\t"<<root->m<<endl;

display(root->right);

}

}

node \* dicn :: search(node \*root){

if(root!=NULL)

{

if(strcmp(key,root->keyw)== 0)

return root;

else if(strcmp(key,root->keyw)>0)

search(root->right);

else if(strcmp(key,root->keyw)<0)

search(root->left);

else return NULL;

}

else return NULL;

}

void dicn :: update(node \*temp){

//cout<<temp->keyw;

cout<<"\nUpdate meaning : ";

//ws(cin);

cin.getline(temp->m,100);

}

node \* dicn :: del(node \*temp, char key[]){

node \*p;

char temp1[10];

if(temp==NULL)

return root;

if(strcmp(key,temp->keyw)<0)

temp->left=del(temp->left,key);

else if(strcmp(key,temp->keyw)>0)

temp->right=del(temp->right,key);

else

{

if(temp->left==NULL)

{

p=temp->right;

delete temp;

return p;

}

else if(temp->right==NULL)

{

p=temp->left;

delete temp;

return p;

}

else

{

p=findmax(temp->left);

for(i=0;i<10;i++)

temp->keyw[i]=p->keyw[i];

for(i=0;i<100;i++)

temp->m[i]=p->m[i];

temp->left=del(temp->left,p->keyw);

}

}

return temp;

}

int main(){

dicn ob;

int ans;

do{

cout<<"\nEnter your choice\n";

cout<<"1: Creation of dictionary\n";

cout<<"2: Display in ascending order\n";

cout<<"3: Update a keyword\n";

cout<<"4: Add new keyword\n";

cout<<"5: Deletion\n";

cout<<"0: Exit\n";

cin>>ans;

switch(ans){

case 1: int c;

cout<<"\nEnter number of words to be stored : ";

cin>>c;

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

ob.create();

break;

case 2: cout<<"\n \*\*\*\* DICTIONARY \*\*\*\*\n";

ob.display(ob.ret\_r());

break;

case 3: cout<<"\nEnter keyword which is to be updated\n";

ws(cin);

cin.getline(key,10);

ob.update(ob.search(ob.ret\_r()));

break;

case 4: ob.create();

break;

case 5: cout<<"\nEnter keyword which is to be deleted\n";

ws(cin);

cin.getline(key,10);

ob.del(ob.ret\_r(),key);

break;

}

}while(ans!=0);

}

**Output:**

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

1

Enter number of words to be stored : 3

Enter keyword : apple

Enter meaning : fruit

Enter keyword : lotus

Enter meaning : flower

Enter keyword : horse

Enter meaning : animal

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

2

\*\*\*\* DICTIONARY \*\*\*\*

apple fruit

horse animal

lotus flower

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

3

Enter keyword which is to be updated

apple

Update meaning : seedless fruit

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

4

Enter keyword : zebra

Enter meaning : animal

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

2

\*\*\*\* DICTIONARY \*\*\*\*

apple seedless fruit

horse animal

lotus flower

zebra animal

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

5

Enter keyword which is to be deleted

horse

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

2

\*\*\*\* DICTIONARY \*\*\*\*

apple seedless fruit

lotus flower

zebra animal

Enter your choice

1: Creation of dictionary

2: Display in ascending order

3: Update a keyword

4: Add new keyword

5: Deletion

0: Exit

0

**Conclusion:**

Through this assignment we understood implementation of BST and perform basic operations like insert node, search node,mirror image of tree,ﬁnding minimum value and no of nodes in longest path.