SC165

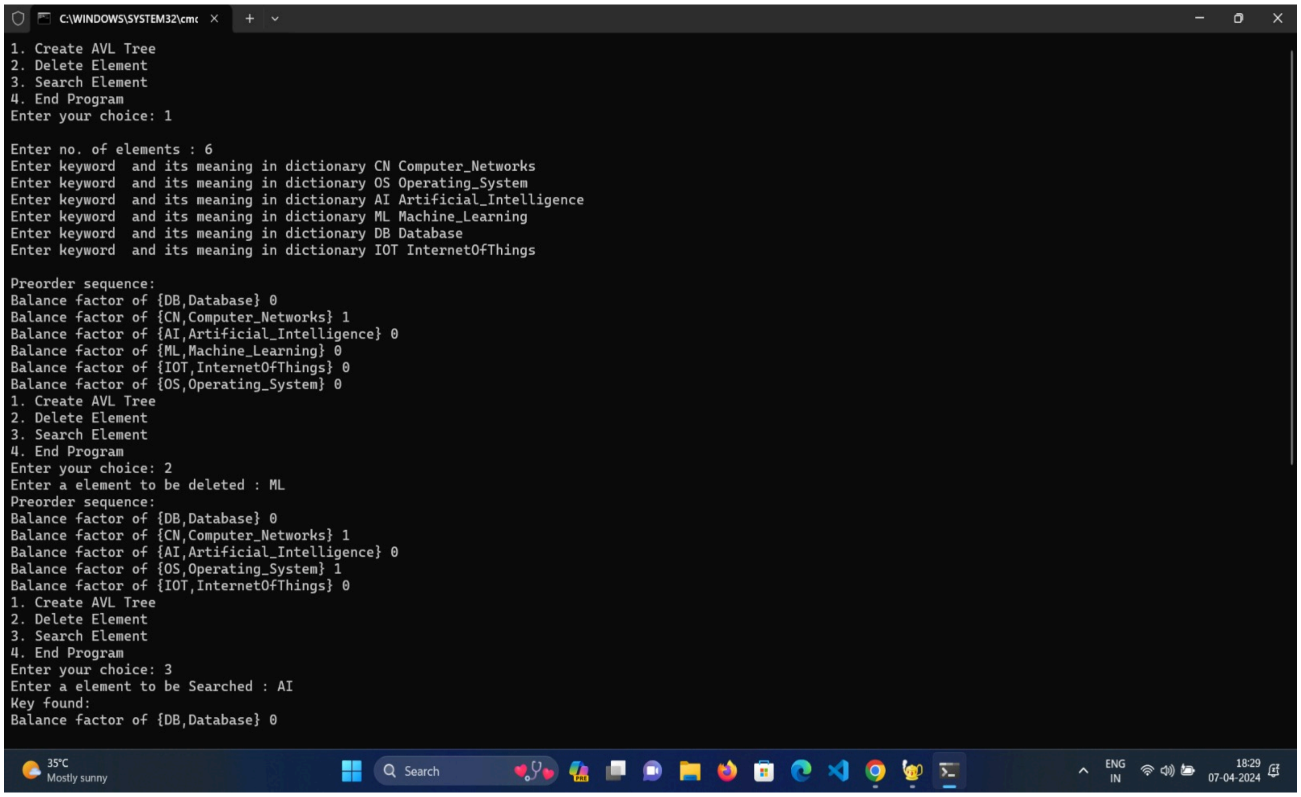
Problem Statement: A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide a facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use a Height balanced tree and find the complexity for finding a keyword.

```
#include<bits/stdc++.h>
using namespace std;
int s;
class node
{
public:
string keyword;
string meaning;
node *left,*right;
int ht;
node(){
keyword="";
meaning="";
left=NULL;
right=NULL;
ht=0;
}
node(string k, string m){
keyword=k;
meaning=m;
left=NULL;
right=NULL;
}
};
class AVL{
public:
node *root=NULL;
node *insert(node *,string, string);
node *Delete(node *,string);
void preorder(node *);
int Search(node *,string);
//void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int balanceFactor(node *);
};
node *AVL::insert(node *T,string k, string m)
{
if(T == NULL){
T=new node(k,m);
// T = (node*)malloc(sizeof(node));
T -> keyword = k;
T->meaning=m;
T -> left = NULL;
T -> right = NULL;
}
else
if(k > T->keyword)
{
T -> right = insert(T -> right,k, m);
if(balanceFactor(T) == -2)
if( k > T -> right -> keyword)
T = RR(T);
else
T = RL(T);
}
else
if(k<T->keyword)
{
T -> left = insert(T -> left,k,m );
if(balanceFactor(T)==2)
if(k < T-> left -> keyword)
T=LL(T);
else
T=LR(T);
}
T -> ht = height(T);
return(T);
}
node *AVL:: Delete(node *T,string k)
{
node *p;
if(T == NULL)
{
return NULL;
}
else
if(k > T->keyword)
{
T -> right = Delete(T -> right,k);
if(balanceFactor(T) == 2)
if(balanceFactor(T -> left) >= 0)
T = LL(T);
else
T = LR(T);
}
else
if(k < T -> keyword)
{
T -> left = Delete(T->left,k);
if(balanceFactor(T)==-2)
if(balanceFactor(T->right)<=0)
T=RR(T);
else
T=RL(T);
}
else
{
if(T -> right != NULL)
{
p = T -> right;
while(p -> left != NULL)
p = p -> left;
T -> keyword = p -> keyword;
T-> meaning = p->meaning;
T -> right = Delete(T -> right,p->keyword);
if(balanceFactor(T) == 2)
if(balanceFactor (T -> left) >= 0)
T=LL(T);
else
T=LR(T);
}
else
return(T->left);
}
T ->ht = height(T);
return(T);
}
int AVL::Search(node *T, string k){
```

```
if (T==NULL)
return -1;
string t=T->keyword;
if(t==k){
s=1;
}
else if(k<t)
s=Search(T->left, k );
else
s= Search(T->right, k );
return s;
}
int AVL::height(node *T)
{
int lh,rh;
if(T == NULL)
return(0);
if( T -> left == NULL)
lh = 0;
else
lh = 1 + T -> left -> ht;
if(T -> right == NULL)
rh = 0;
else
rh=1+T->right->ht;
if(lh > rh)
return(lh);
return(rh);
}
node * AVL::rotateright(node *x)
{
node *y;
y = x -> left;
x -> left = y -> right;
y -> right = x;
x -> ht = height(x);
y -> ht = height(y);
return(y);
}
node * AVL::rotateleft(node *x)
{
node *y;
y = x -> right;
x -> right = y -> left;
```

```
y -> left = x;
x -> ht = height(x);
y -> ht = height(y);
return(y);
}
node *AVL::RR(node *T)
{
T = rotateleft(T);
return(T);
}
node *AVL::LL(node *T)
{
T = rotateright(T);
return(T);
}
node *AVL::LR(node *T)
{
/* T -> left = rotateleft(T->left);
T = rotateright(T);*/
T -> left = RR(T->left);
T = LL(T);
return(T);
}
node *AVL::RL(node *T)
{
/* T -> right = rotateright(T->right);
T = rotateleft(T);*/
T -> right = LL(T->right);
T = RR(T);
return(T);
}
int AVL::balanceFactor(node *T)
{
int lh,rh;
if( T == NULL)
return(0);
if(T -> left == NULL)
lh = 0;
else
lh = 1 + T->left->ht;
if(T -> right == NULL)
rh = 0;
else
rh = 1 + T -> right -> ht;
```

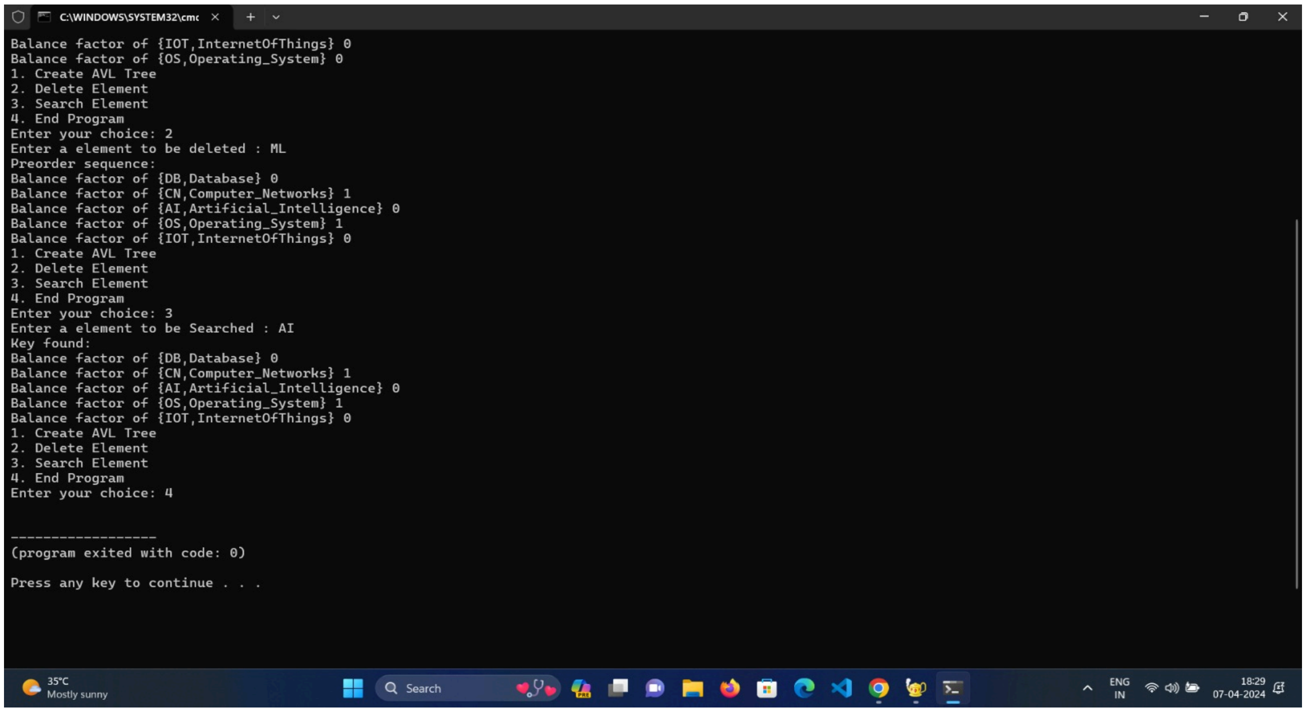
```
return(lh-rh);
}
void AVL::preorder(node *T)
{
if( T != NULL)
{
cout << "Balance factor of {" << T -> keyword << ","<<T -> meaning<<"} "<<
balanceFactor(T) << endl;
preorder(T->left);
preorder(T->right);
}
}
/*void AVL:: inorder(node *T)
{
if( T != NULL)
{
inorder(T->left);
cout << “Balance Factor “< element<<” “ << balanceFactor(T) << endl;
inorder(T->right);
}
}*/
int main()
{
AVL avl;
//node *root=NULL;
int n,i,option;
string k,m;
do {
cout << "1. Create AVL Tree\n";
cout << "2. Delete Element\n";
cout << "3. Search Element\n";
cout << "4. End Program\n";
cout << "Enter your choice: ";
cin >> option;
switch(option)
{
case 1: cout << "\nEnter no. of elements : ";
cin >> n;
avl.root = NULL;
for( i = 0; i < n;i++) {
cout << "Enter keyword and its meaning in dictionary ";
cin >> k>>m;
avl.root = avl.insert(avl.root,k,m);
}
cout << "\nPreorder sequence:\n";
avl.preorder(avl.root);
// cout << “\n\nInorder sequence:\n“;
//inorder(avl.root);
break;
case 2: cout << "Enter a element to be deleted : ";
cin >> k;
avl.root = avl.Delete(avl.root,k);
cout << "Preorder sequence:\n";
avl.preorder(avl.root);
break;
case 3: cout << "Enter a element to be Searched : ";
cin >> k;
int s = avl.Search(avl.root,k);
if(s!=-1)
cout << "Key found:\n";
else
cout << "Key not found:\n";
avl.preorder(avl.root);
break;
}
}while(option!=4);
return 0;
}
```



```
1. Create AVL Tree
2. Delete Element
3. Search Element
4. End Program
Enter your choice: 1

Enter no. of elements : 6
Enter keyword  and its meaning in dictionary CN Computer_Networks
Enter keyword  and its meaning in dictionary OS Operating_System
Enter keyword  and its meaning in dictionary AI Artificial_Intelligence
Enter keyword  and its meaning in dictionary ML Machine_Learning
Enter keyword  and its meaning in dictionary DB Database
Enter keyword  and its meaning in dictionary IOT InternetOfThings

Preorder sequence:
Balance factor of {DB,Database} 0
Balance factor of {CN,Computer_Networks} 1
Balance factor of {AI,Artificial_Intelligence} 0
Balance factor of {ML,Machine_Learning} 0
Balance factor of {IOT,InternetOfThings} 0
Balance factor of {OS,Operating_System} 0
1. Create AVL Tree
2. Delete Element
3. Search Element
4. End Program
Enter your choice: 2
Enter a element to be deleted : ML
Preorder sequence:
Balance factor of {DB,Database} 0
Balance factor of {CN,Computer_Networks} 1
Balance factor of {AI,Artificial_Intelligence} 0
Balance factor of {OS,Operating_System} 1
Balance factor of {IOT,InternetOfThings} 0
1. Create AVL Tree
2. Delete Element
3. Search Element
4. End Program
Enter your choice: 3
Enter a element to be Searched : AI
Key found:
Balance factor of {DB,Database} 0
```



```
Balance factor of {IOT,InternetOfThings} 0
Balance factor of {OS,Operating_System} 0
1. Create AVL Tree
2. Delete Element
3. Search Element
4. End Program
Enter your choice: 2
Enter a element to be deleted : ML
Preorder sequence:
Balance factor of {DB,Database} 0
Balance factor of {CN,Computer_Networks} 1
Balance factor of {AI,Artificial_Intelligence} 0
Balance factor of {OS,Operating_System} 1
Balance factor of {IOT,InternetOfThings} 0
1. Create AVL Tree
2. Delete Element
3. Search Element
4. End Program
Enter your choice: 3
Enter a element to be Searched : AI
Key found:
Balance factor of {DB,Database} 0
Balance factor of {CN,Computer_Networks} 1
Balance factor of {AI,Artificial_Intelligence} 0
Balance factor of {OS,Operating_System} 1
Balance factor of {IOT,InternetOfThings} 0
1. Create AVL Tree
2. Delete Element
3. Search Element
4. End Program
Enter your choice: 4


------------------
(program exited with code: 0)

Press any key to continue . . .
```