Name: Abdulmuiz Khalid Shaikh
Roll no.:2101062
Subject: Data Structure and Algorithm Laboratory
Assignment No.8

```
#include<iostream>
#include<algorithm>
#include<cstdlib>
usingnamespacestd;
classnode
{
public:
stringword,meaning;
intht;
node*left,*right;
};
classAVL
{
public:
node*root;
AVL()
{
root=NULL;
}
node*insert(node*,string,string);
node*deleteNode(node*,string);
voidpreorder(node*);
voidinorder(node*);
node*RotateRight(node*);
node*RotateLeft(node*);
node*RR(node*);
node*LL(node*);
node*LR(node*);
node*RL(node*);
intheight(node*);
intBF(node*);
voidsearch(node*,string);
voidmodify(node*,string);
};
intAVL::height(node*temp)
{
intlh,rh;
if(temp==NULL)
return0;
lh=(temp->left==NULL)?0:1+temp->left->ht;
rh=(temp->right==NULL)?0:1+temp->right->ht;
returnmax(lh,rh);
}
intAVL::BF(node*temp)
{
if(temp==NULL)
return0;
returnheight(temp->left)-height(temp->right);
}
node*AVL::RotateRight(node*parent)
{
node*temp;
temp=newnode;
temp=parent->left;
```

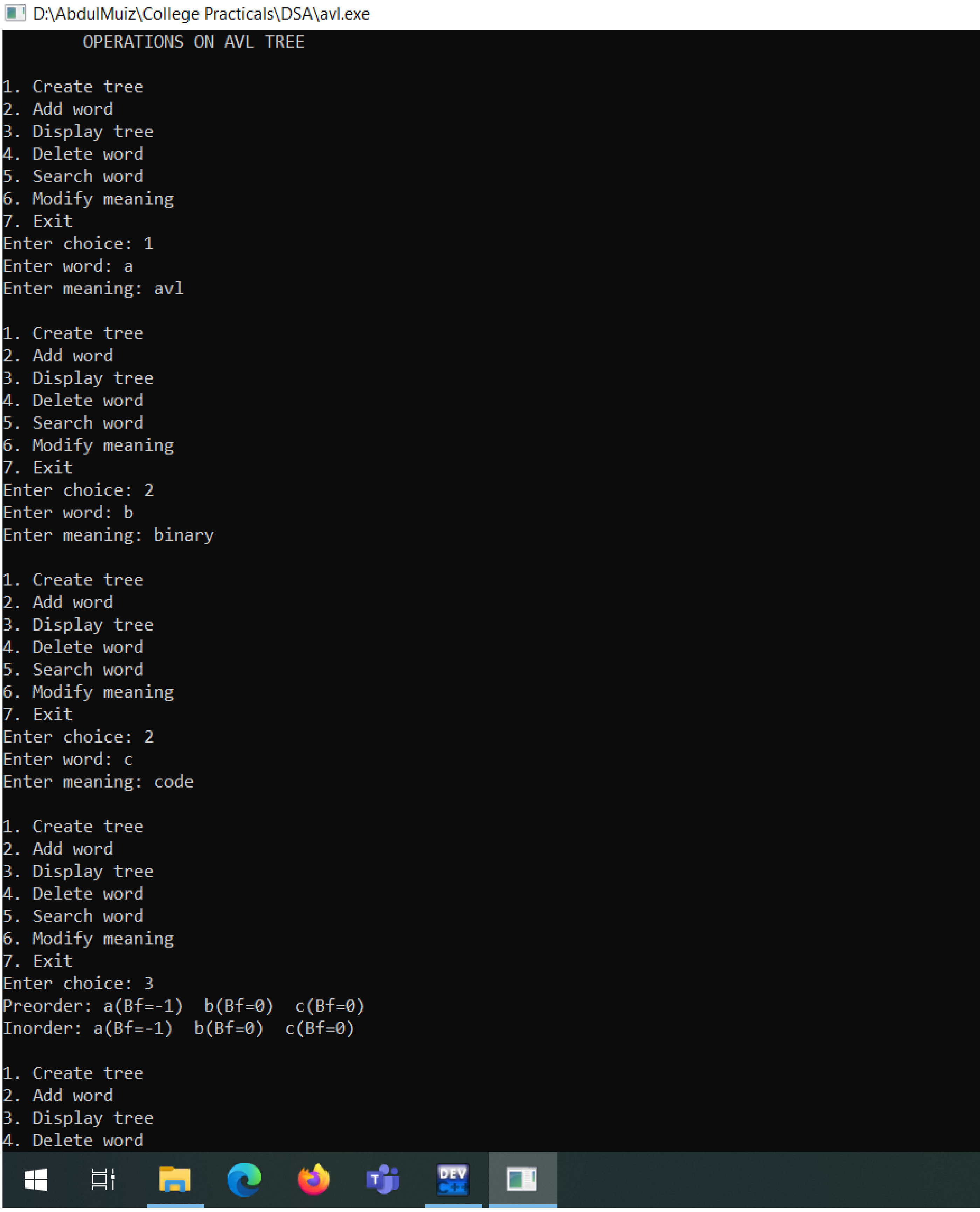
```
parent->left=temp->right;
temp->right=parent;
parent->ht=height(parent);
temp->ht=height(temp);
returntemp;
}
node*AVL::RotateLeft(node*parent)
{
node*temp;
temp=newnode;
temp=parent->right;
parent->right=temp->left;
temp->left=parent;
parent->ht=height(parent);
temp->ht=height(temp);
returntemp;
}
node*AVL::RR(node*T)
{
T=RotateLeft(T);
returnT;
}
node*AVL::LL(node*T)
{
T=RotateRight(T);
returnT;
}
node*AVL::LR(node*T)
{
T->left=RotateLeft(T->left);
T=RotateRight(T);
returnT;
}
node*AVL::RL(node*T)
{
T->right=RotateRight(T->right);
T=RotateLeft(T);
returnT;
}
node*AVL::insert(node*temp,stringstr_w,stringstr_m)
{
if(temp==NULL)
{
temp=newnode;
temp->word=str_w;
temp->meaning=str_m;
temp->left=temp->right=NULL;
}
else
{
if(str_w.compare(temp->word)>0)
{
temp->right=insert(temp->right,str_w,str_m);
if(BF(temp)==-2)
temp=(str_w.compare(temp->right->word)>0)?RR(temp):RL(temp);
}
else
{
temp->left=insert(temp->left,str_w,str_m);
if(BF(temp)==2)
temp=(str_w.compare(temp->left->word)<0)?LL(temp):LR(temp);
}
}
temp->ht=height(temp);
returntemp;
}
node*AVL::deleteNode(node*temp,stringstr_w)
{
if(temp==NULL)
returnNULL;
else
{
if(str_w.compare(temp->word)>0)
{
temp->right=deleteNode(temp->right,str_w);
if(BF(temp)==2)
temp=(BF(temp->left)>=0)?LL(temp):LR(temp);
}
else
{
if(str_w.compare(temp->word)<0)
{
temp->left=deleteNode(temp->left,str_w);
if(BF(temp)==-2)
temp=(BF(temp->right)<=0)?RR(temp):RL(temp);
}
else
{
if(temp->right!=NULL)
{
node*temp1;
temp1=temp->right;
while(temp1->left!=NULL)
temp1=temp1->left;
temp->word=temp1->word;
temp->right=deleteNode(temp->right,temp1->word);
if(BF(temp)==2)
temp=(BF(temp->left)>=0)?LL(temp):LR(temp);
}
else
returntemp->left;
}
}
}
temp->ht=height(temp);
returntemp;
}
voidAVL::preorder(node*root)
{
if(root!=NULL)
{
cout<<root->word<<"(Bf="<<BF(root)<<") ";
preorder(root->left);
preorder(root->right);
}
}
voidAVL::inorder(node*root)
{
if(root!=NULL)
{
```

```
inorder(root->left);
cout<<root->word<<"(Bf="<<BF(root)<<") ";
inorder(root->right);
}
}
voidAVL::search(node*root,stringstr_w)
{
if(str_w.compare(root->word)<0)
{
if(root->left==NULL)
cout<<"Wordnotfound";
else
search(root->left,str_w);
}
elseif(str_w.compare(root->word)>0)
{
if(root->right==NULL)
cout<<"Wordnotfound";
else
search(root->right,str_w);
}
else
{
cout<<"Word:"<<root->word<<endl;
cout<<"Meaning:"<<root->meaning<<endl;
}
}
voidAVL::modify(node*root,stringstr_w)
{
if(str_w.compare(root->word)<0)
{
if(root->left==NULL)
cout<<"Wordnotfound";
else
modify(root->left,str_w);
}
elseif(str_w.compare(root->word)>0)
{
if(root->right==NULL)
cout<<"Wordnotfound";
else
modify(root->right,str_w);
}
else
{
getline(cin,root->meaning);
cout<<"Enternewmeaning:";
getline(cin,root->meaning);
}
}
intmain()
{
AVLTree;
intch;
stringstr1,str2;
cout<<"\tOPERATIONSONAVLTREE\t"<<endl;
while(true)
{
cout<<"\n1.Createtree"<<endl;
cout<<"2.Addword"<<endl;
```

```
cout<<"3.Displaytree"<<endl;
cout<<"4.Deleteword"<<endl;
cout<<"5.Searchword"<<endl;
cout<<"6.Modifymeaning"<<endl;
cout<<"7.Exit"<<endl;
cout<<"Enterchoice:";
cin>>ch;
switch(ch)
{
case1:
case2:cout<<"Enterword:";
cin>>str1;
getline(cin,str2);
cout<<"Entermeaning:";
getline(cin,str2);
Tree.root=Tree.insert(Tree.root,str1,str2);
break;
case3:cout<<"Preorder:";Tree.preorder(Tree.root);cout<<endl;
cout<<"Inorder:";Tree.inorder(Tree.root);cout<<endl;
break;
case4:cout<<"Enterword:";
cin>>str1;
Tree.root=Tree.deleteNode(Tree.root,str1);
break;
case5:cout<<"Enterword:";
cin>>str1;
Tree.search(Tree.root,str1);
break;
case6:cout<<"Enterword:";
cin>>str1;
Tree.modify(Tree.root,str1);
break;
case7:exit(1);break;
}
}
return0;
}

/*
OPERATIONSONAVLTREE
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:1
Enterword:a
Entermeaning:avl
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:2
Enterword:b
Entermeaning:binary
```

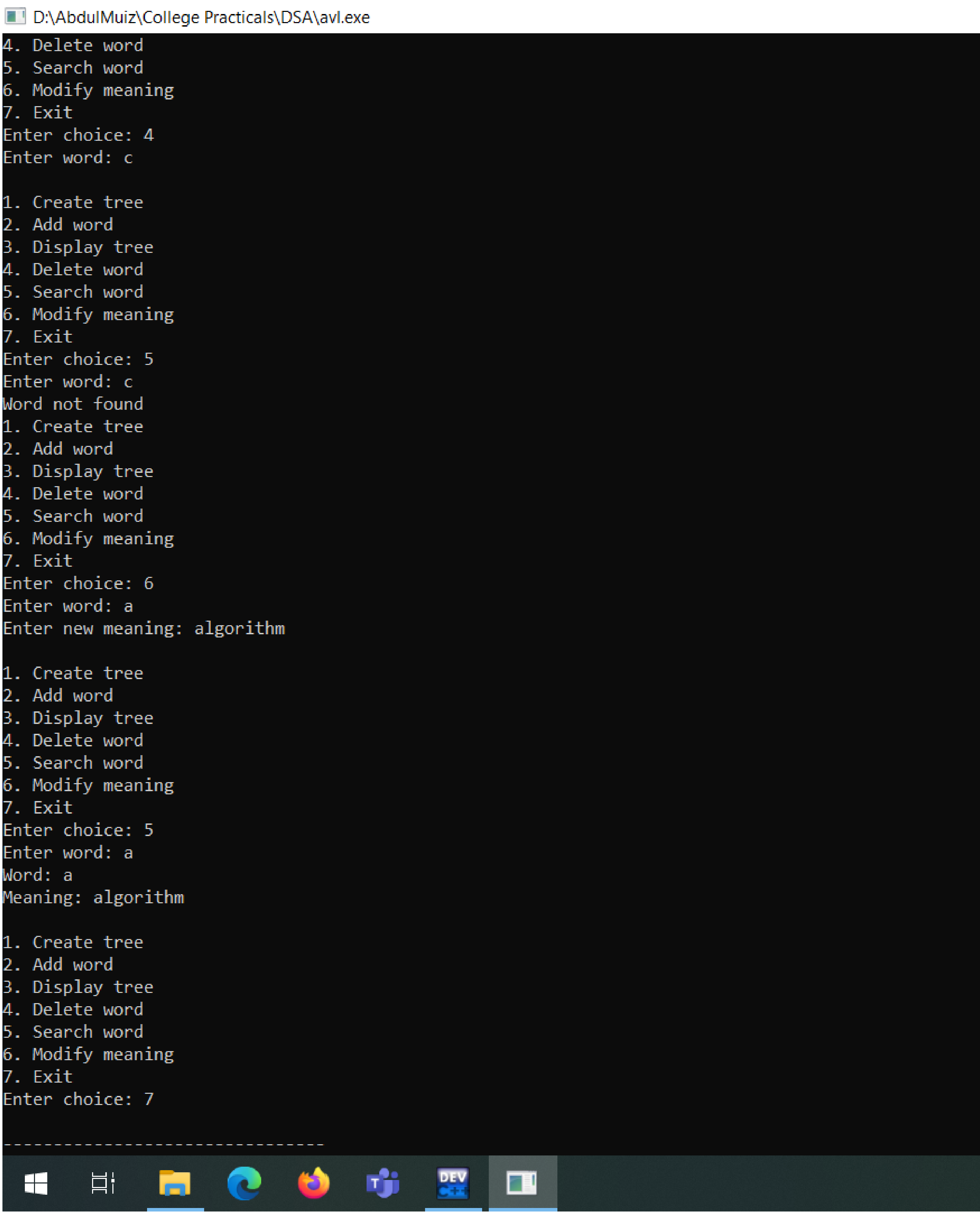
```
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:2
Enterword:c
Entermeaning:code
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:3
Preorder:a(Bf=-1) b(Bf=0) c(Bf=0)
Inorder:a(Bf=-1) b(Bf=0) c(Bf=0)
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:4
Enterword:c
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:5
Enterword:c
Wordnotfound
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:6
Enterword:a
Enternewmeaning:algorithm
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:5
Enterword:a
Word:a
Meaning:algorithm
```

```
1.Createtree
2.Addword
3.Displaytree
4.Deleteword
5.Searchword
6.Modifymeaning
7.Exit
Enterchoice:7
*/
```



D:\AbdulMuiz\College Practicals\DSA\avl.exe

```
        OPERATIONS ON AVL TREE

1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 1
Enter word: a
Enter meaning: avl

1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 2
Enter word: b
Enter meaning: binary

1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 2
Enter word: c
Enter meaning: code

1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 3
Preorder: a(Bf=-1)  b(Bf=0)  c(Bf=0)
Inorder: a(Bf=-1)  b(Bf=0)  c(Bf=0)

1. Create tree
2. Add word
3. Display tree
4. Delete word
```



D:\AbdulMuiz\College Practicals\DSA\avl.exe

```
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 4
Enter word: c

1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 5
Enter word: c
Word not found
1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 6
Enter word: a
Enter new meaning: algorithm

1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 5
Enter word: a
Word: a
Meaning: algorithm

1. Create tree
2. Add word
3. Display tree
4. Delete word
5. Search word
6. Modify meaning
7. Exit
Enter choice: 7

--------------------------------
```