Advanced Data Structures Assignment

Name : Shrirang R Mhalgi

Roll number : 222006

Class : SE B

Batch : B1

--------------------------------------------------------------------------------------------------------------------------------------

Problem Statement:

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 Height balance tree and find the complexity for finding a keyword

Code:

#include "iostream"

using namespace std;

class node {

string key,mean;

node \*left,\*right;

int h;

friend class avl;

public :

node() {

left = right = NULL;

key = mean = "\0";

}

};

class avl {

public :

node \*root;

avl() {

root=NULL;

}

void create();

node \*insert(node \*,string);

void display(node \*);

void Delete();

node \*balance(node \*);

int height(node \*);

void search(node \*);

node \*rotLeft(node \*);

node \*rotRight(node \*);

int balFact(node \*);

node \*Delete(node \*,string);

node \*nextIn(node \*);

};

node \*avl :: rotLeft(node \*curr) {

node \*temp = NULL;

temp = curr->right;

curr->right = temp->left;

temp->left=curr;

return temp;

}

node \*avl :: rotRight(node \*curr) {

node \*temp = NULL;

temp = curr->left;

curr->left=temp->right;

temp->right=curr;

return temp;

}

node\* avl :: insert(node \*root,string temp) {

node \*New;

if(root == NULL) {

New = new node;

New->key = temp;

cout<<"Enter the Meaning for this Word : ";

cin>>temp;

New->mean = temp;

root = New;

}

else if(root->key > temp) {

root->left = insert(root->left,temp);

}

else if(root->key < temp) {

root->right = insert(root->right,temp);

}

else if(root->key == temp) {

cout<<"Word Already Present !!\n";

}

root = balance(root);

return root;

}

void avl :: create() {

string temp;

char ch;

do {

cout<<"Enter the Word to be inserted in Dictionary : ";

cin>>temp;

root = insert(root,temp);

cout<<"Do you wish to insert another node ?? (y/n) : ";

cin>>ch;

} while(ch == 'y');

}

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

if(root!=NULL) {

display(root->left);

cout<<"Word : "<<root->key<<"; Meaning : "<<root->mean<<";\n";

display(root->right);

}

}

void search(node \*root) {

}

int avl :: height(node \*root) {

int h=0;

if(root == NULL) return 1;

else if(root != NULL) {

h = 1+max(height(root->left),height(root->right));

}

return h;

}

int avl :: balFact(node \*temp) {

return (height(temp->left) - height(temp->right));

}

node \* avl :: balance(node \*root) {

int nowBal = balFact(root);

if(nowBal > 1) {

if(balFact(root->left) > 0) {

root = rotRight(root);

}

else {

root->left = rotLeft(root->left);

root = rotRight(root);

}

}

else if(nowBal < -1) {

if(balFact(root->right) > 0) {

root->right = rotRight(root->right);

root = rotLeft(root);

}

else root = rotLeft(root);

}

return root;

}

node \*avl :: nextIn(node \*temp) {

if(temp->right==NULL) return temp;

else {

temp=temp->right;

while(temp->left!=NULL) temp=temp->left;

return temp;

}

}

node \*prev=NULL;

node \*avl :: Delete(node \*root,string val) {

if(root!=NULL) {

if(root->key==val) {

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

if(prev == NULL) this->root = NULL;

else if(prev->left==root) prev->left=NULL;

else if(prev->right==root) prev->right=NULL;

delete root;

}

else if(root->left==NULL && root->right!=NULL) {

//right child present

if(prev == NULL) {

prev = root;

this->root = root->right;

prev->right = NULL;

delete prev;

}

else if(prev->left==root) {

prev->left=root->right;

delete root;

}

else if(prev->right == root){

prev->right=root->right;

delete root;

}

}

else if(root->left!=NULL && root->right==NULL) {

//left child present

if(prev == NULL) {

prev = root;

this->root = root->left;

prev->left = NULL;

delete prev;

}

else if(prev->left==root) {

prev->left=root->left;

delete root;

}

else if(prev->right==root){

prev->right=root->left;

delete root;

}

}

else if(root->left!=NULL && root->right!=NULL) {

//both children present

node \*temp = NULL;

temp = nextIn(root);

root->key = temp->key;

root->mean=temp->mean;

// root->v=temp->v; //all meanings exchange

root->right = Delete(root->right,root->key);

}

}

else if(root->key > val) {

prev=root;

root->left = Delete(root->left,val);

}

else if(root->key < val) {

prev=root;

root->right = Delete(root->right,val);

}

}

else cout<<"Word Not Present OR Already Deleted \n";

root = balance(root);

return root;

}

int main() {

avl a;

int op;

do {

cout<<"1.Insert\n2.Display\n3.Delete\n4.Exit\n";

cin>>op;

switch(op) {

case 1 : {

a.create();

break;

}

case 2 : {

a.display(a.root);

break;

}

case 3 : {

string word;

cout<<"Enter the Word to be Deleted : ";

cin>>word;

a.root = a.Delete(a.root,word);

break;

}

default : {

cout<<"Terminating Program !!\n";

return 0;

}

}

} while(1);

return 0;

}

Output:

1.Insert

2.Display

3.Delete

4.Exit

1

Enter the Word to be inserted in Dictionary : apple

Enter the Meaning for this Word : fruit

Do you wish to insert another node ?? (y/n) : y

Enter the Word to be inserted in Dictionary : sun

Enter the Meaning for this Word : star

Do you wish to insert another node ?? (y/n) : y

Enter the Word to be inserted in Dictionary : horse

Enter the Meaning for this Word : animal

Do you wish to insert another node ?? (y/n) : y

Enter the Word to be inserted in Dictionary : lotus

Enter the Meaning for this Word : flower

Do you wish to insert another node ?? (y/n) : y

Enter the Word to be inserted in Dictionary : potato

Enter the Meaning for this Word : vegetable

Do you wish to insert another node ?? (y/n) : n

1.Insert

2.Display

3.Delete

4.Exit

2

Word : apple; Meaning : fruit;

Word : horse; Meaning : animal;

Word : lotus; Meaning : flower;

Word : potato; Meaning : vegetable;

Word : sun; Meaning : star;

1.Insert

2.Display

3.Delete

4.Exit

4