**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Algorytmy i struktury danych | **Zadanie** | 7\_01 |
| **Autor** | Kamil Borkowski | **Grupa** | WCY22IY1S1 |
| **Temat** | Drzewo AVL | | |

1. Treść

Napisz program, który umożliwi: Budowanie drzewa AVL

Program powinien:

Umożliwić wczytywanie danych z pliku wejściowego

Umożliwić wprowadzanie danych z klawiatury

Dodawanie węzła

Kasowanie węzła

Szukanie węzła (ścieżka)

Kasowanie danych z całego drzewa

Narysowanie/ wyświetlenie drzewa (AVL)

Wypisanie węzłów drzewa w kolejności przechodzenia

VLR – pre-order, przejście wzdłużne, prefiksowe

LVR – in-order, przejście poprzeczne, infiksowe

LRV – post-order, przejście wsteczne, postfiksowe

* 1. Metoda realizacji

Program implementuje strukturę danych drzewa AVL za pomocą dynamicznie alokowanych węzłów. Korzysta z interfejsu użytkownika, umożliwiając dodawanie nowych węzłów, usuwanie danego węzła oraz wyświetlanie drzewa w różnych porządkach (in-order, pre-order, post-order). Funkcje rekurencyjnie operują na węzłach drzewa, a program kończy się zwolnieniem pamięci zajmowanej przez węzły. Użytkownik może wybierać operacje poprzez wprowadzenie numeru komendy w prostym interfejsie tekstowym.

* 1. Założenia / ograniczenia dotyczące danych:
     1. Dane wejściowe *(opis)*

Wartości węzłów – wczytywana z pliku/klawiatury

* + 1. Dane wyjściowe *(opis)*

Graficzne przedstawienie drzewa – wyświetlone na ekranie i w pliku .txt

Przejscie po węzłach metodami VLR, LVR, LRV – wyświetlone na ekranie i w pliku .txt

1. Realizacja
   1. Algorytm *(lista kroków lub schemat blokowy)*

1. Utworzenie struktury DrzewoAVL zawierającej klucz, czynnik równoważenia i wskaźniki na rodzica, lewe i prawe dziecko.

2. Zaimplementowanie funkcji obsługujących rotacje (RR, LL, RL, LR) w celu utrzymania równowagi drzewa AVL po dodaniu lub usunięciu węzłów.

3. Stworzenie funkcji DodajW do dodawania nowych węzłów, równoważąc drzewo w trakcie operacji.

4. Implementacja funkcji usunW, usuwającej węzeł i równoważącej drzewo.

5. Implementacja funkcji printBT do wizualizacji struktury drzewa.

6. Utworzenie funkcji SciezkaW do wyszukiwania i wyświetlania ścieżki do konkretnego węzła.

7. Utworzenie funkcji zwolnij\_pamiec w celu zwolnienia pamięci zajmowanej przez drzewo po zakończeniu działania programu.

8. Implementacja funkcji LVR, VLR, LRV do wypisywania węzłów w różnych porządkach.

9. Stworzenie możliwości wczytywania danych z pliku (funkcja ifstream), co umożliwia łatwe inicjowanie drzewa danymi.

10. Zdefiniowanie interfejsu użytkownika w funkcji main, umożliwiającej wybór różnych operacji na drzewie AVL poprzez wprowadzanie numerów operacji.

11. Stworzenie pętli while w funkcji main, pozwalającej użytkownikowi iteracyjnie wykonywać różne operacje na drzewie AVL poprzez interakcję z menu.

Kod źródłowy

//================================================

//Zadanie 7\_01 jezyk C++

//Drzewo AVL

//WCY22IY1S1 Borkowski Kamil

//================================================

#include <iostream>

#include <string>

#include <cstdlib>

#include <fstream>

using namespace std;

struct DrzewoAVL

{

DrzewoAVL \* up;

DrzewoAVL\* left;

DrzewoAVL \* right;

int key, bf;

};

//MODUL wizualizacji drzewa

string cr, cl, cp;

void printBT ( string sp, string sn, DrzewoAVL \* v )

{

string s;

if( v )

{

s = sp;

if( sn == cr ) s [ s.length( ) - 2 ] = ' ';

printBT ( s + cp, cr, v->right );

s = s.substr ( 0, sp.length( )-2 );

cout << s << sn << v->key << ":" << v->bf << endl;

s = sp;

if( sn == cl ) s [ s.length( ) - 2 ] = ' ';

printBT ( s + cp, cl, v->left );

}

}

// Procedura usuwająca drzewo

void zwolnij\_pamiec(DrzewoAVL\* pom)

{

if(pom)//jezeli wezel nie jest null to usuwamy go i jego prawe i lewe poddrzewo

{

zwolnij\_pamiec(pom->left);

zwolnij\_pamiec(pom->right);

free(pom);

}

}

// Rotacja RR

void RR(DrzewoAVL\* & root, DrzewoAVL\* A)

{

DrzewoAVL\* B=A->right, \*p=A->up;

A->right=B->left;

if(A->right)A->right->up=A;

B->left=A;

B->up=p;

A->up=B;

if(p)

{

if(p->left==A)p->left=B;

else p->right=B;

}

else root=B;

if(B->bf==-1)

{

A->bf=B->bf=0;

}

else

{

A->bf=-1;

B->bf=1;

}

}

// Rotacja LL

void LL(DrzewoAVL\* & root,DrzewoAVL\* A)

{

DrzewoAVL\* B=A->left,\*p=A->up;

A->left=B->right;

if(A->left)A->left->up=A;

B->right=A;

B->up=p;

A->up=B;

if(p)

{

if(p->left==A)p->left=B;

else p->right=B;

}

else root=B;

if(B->bf==1)

{

A->bf=B->bf=0;

}

else

{

A->bf = 1;

B->bf = -1;

}

}

// Rotacja RL

void RL(DrzewoAVL\* & root, DrzewoAVL\* A )

{

DrzewoAVL\* B=A->right,\*C=B->left,\*p=A->up;

B->left=C->right;

if(B->left)

{

B->left->up = B;

}

A->right=C->left;

if(A->right)

{

A->right->up=A;

}

C->left=A;

C->right=B;

A->up=B->up=C;

C->up=p;

if(p)

{

if(p->left==A)

{

p->left = C;

}

else p->right=C;

}

else root = C;

if(C->bf==-1)

{

A->bf=1;

}

else A->bf=0;

if(C->bf==1)

{

B->bf=-1;

}

else B->bf = 0;

C->bf = 0;

}

// Rotacja LR

void LR(DrzewoAVL\* & root,DrzewoAVL\* A)

{

DrzewoAVL\* B=A->left,\*C=B->right,\*p=A->up;

B->right=C->left;

if(B->right)

{

B->right->up=B;

}

A->left=C->right;

if(A->left)

{

A->left->up=A;

}

C->right=A;

C->left=B;

A->up=B->up=C;

C->up=p;

if(p)

{

if(p->left==A)

{

p->left=C;

}

else p->right=C;

}

else root=C;

if(C->bf==1)

{

A->bf=-1;

}

else A->bf = 0;

if( C->bf == -1 )

{

B->bf=1;

}

else B->bf = 0;

C->bf=0;

}

// funkcja dodajaca nowy wezel do drzewa

void DodajW(DrzewoAVL\* & root,int k )

{

DrzewoAVL \* w, \* p, \* r; //w to nowy wezel, p sluzy do przemieszczania sie po potomkach, r wskazuje na seniora wezla

bool t;//flaga kontrolna do sprawdzenia potrzeby rownowazenia drzewa

w = (DrzewoAVL\*)malloc(sizeof(DrzewoAVL));

w->left = w->right = w->up = NULL;

w->key = k;

w->bf = 0;

p = root;

if(!p)root=w;//jeśli drzewo jest puste, to nowy wezel jest korzeniem

else

{

while(true)

if(k<p->key)//jesli nowy klucz jest mniejszy od aktualnie sprawdzanego

{

if(!p->left)//jesli wezel nie ma lewego syna to nowy wezel sie nim staje

{

p->left=w;

break;

}

p=p->left;//jezeli wezel ma lewego syna to idziemy po nim

}

else//jesli nowy klucz jest wiekszy od aktualnie sprawdzanego

{

if(!p->right)//jesli wezel nie ma prawego syna to nowy wezel sie nim staje

{

p->right=w;

break;

}

p=p->right;//inaczej przechodzimy do prawego syna

}

w->up=p;//jezeli wezel ma prawego syna to idziemy po nim

//po wstawieniu wezla trzeba zrownowazyc drzewo

if(p->bf) p->bf=0;//jesli drzewo przed dodaniem wezla bylo niezrownowazone to po dodaniu wezla bedzie zrownowazone

else

{

if( p->left == w )//zmiana wagi zaleznie czy nowy wezel dodano z lewej czy prawej strony

p->bf = 1;

else

p->bf = -1;

r=p->up;//poruszanie sie w gore drzewa

t = false;//nie trzeba rownowazyc drzewa

while(r)//szukamy niezrownowazonych wezlow

{

if(r->bf)

{

t=true;

break;

};

if(r->left==p)

{

r->bf=1;

}

else

{

r->bf = -1;

}

p=r;//przejscie w gore drzewa

r=r->up;

}

if(t)//sprawdzenie czy drzewo wymaga zrownowazenia

{

if(r->bf==1)

{

if(r->right==p)

{

r->bf=0;

}

else if( p->bf == -1 ) LR ( root, r );//rotacja LR

else LL ( root, r );//rotacja LL

}

else

{

if(r->left==p)

{

r->bf=0;

}

else if(p->bf==1) RL(root,r);//rotacja RL

else RR(root, r );//rotacja RR

}

}

}

}

}

// Funkcja znajduje poprzednik węzła p

DrzewoAVL \* zwroc\_up(DrzewoAVL \* p)

{

DrzewoAVL \* r;

if(p)

{

if(p->left )

{

p=p->left;

while(p->right) p = p->right;

}

else

do

{

r=p;

p=p->up;

} while( p && p->right != r );

}

return p;

}

//funkcja pomocnicza dla funkcji usuwajacej wezel

DrzewoAVL \* podaj\_klucz(DrzewoAVL \*p, int k) {

while (p != NULL && p->key != k) {

if (k < p->key) {

p = p->left;

} else {

p = p->right;

}

}

return p;

}

// Funkcja usuwajaca węzeł

DrzewoAVL \* usunW (DrzewoAVL \* & root, DrzewoAVL \* x )

{

DrzewoAVL \*t, \*y, \*z;

bool zrow;//flaga sprawdzajaca zrownowazenie po usunieciu wezla

if( x->left && x->right )//jesli ma dwoch przodkow

{

y=usunW(root,zwroc\_up(x));

zrow=false;

}

else

{

if( x->left )//jesli ma tylko jednego przodka to oznaczamy je y

{

y = x->left;

x->left = NULL;

}

else

{

y = x->right;

x->right = NULL;

}

x->bf = 0;

zrow = true;

}

if(y)//przypisujemy do y atrybuty x

{

y->up = x->up;

y->left = x->left;

if(y->left)

{

y->left->up=y;

}

y->right = x->right;

if( y->right )

{

y->right->up = y;

}

y->bf=x->bf;

}

if(x->up)//aktualizacja seniora x

{

if( x->up->left == x )

{

x->up->left = y;

}

else x->up->right = y;

}

else root = y;

if(zrow)//jesli drzewo moze byc niezrownowazone

{

z = y;

y = x->up;

while( y )

{

if(!y->bf )//sprawdzenie czy drzewo jest juz zrownowazone

{

if( y->left == z )

{

y->bf = -1;

}

else y->bf = 1;

break;

}

else//kiedy drzewo nie jest zrownowazone

{

if(((y->bf == 1 ) && ( y->left == z ) ) || ( ( y->bf == -1 ) && ( y->right == z ) ) )

{

y->bf = 0;

z = y;

y = y->up;

}

else//wybor odpowiedniego poddrzewa do zrownowazenia

{

if( y->left == z )

{

t = y->right;

}

else t = y->left;

if( !t->bf )

{

if( y->bf == 1 )

{

LL ( root, y );

}

else RR ( root, y );

break;

}

else if( y->bf == t->bf )

{

if( y->bf == 1 )

{

LL ( root, y );

}

else RR ( root, y );

z = t;

y = t->up;

}

else

{

if( y->bf == 1 )

{

LR ( root, y );

}

else RL ( root, y );

z = y->up;

y = z->up;

}

}

}

}

}

return x;

}

//funkcja szukajaca sciezki wezla

bool SciezkaW(DrzewoAVL\* root, int key)

{

if (!root) {

cout << "Drzewo jest puste. Wezel nie zostal znaleziony." << endl;

return false;

}

DrzewoAVL\* current = root;

cout << "Sciezka do wezla " << key << ": ";

while (current)

{

cout << current->key << " ";

if (key == current->key) {

cout<<endl<<endl;

return true;

}

if (key < current->key)

current = current->left;

else

current = current->right;

}

cout << endl;

cout << "Wezel o kluczu " << key << " nie zostal znaleziony." << endl;

return false;

}

//funkcja wypisujaca wezly infiksowo

void LVR(DrzewoAVL\* root) {

if (root != NULL) {

LVR(root->left);

cout << root->key << " ";

LVR(root->right);

}

}

//funkcja wypisujaca wezly prefiksowo

void VLR(DrzewoAVL\* root) {

if (root != NULL) {

cout << root->key << " ";

VLR(root->left);

VLR(root->right);

}

}

//funkcja wypisujaca wezly postfiksowo

void LRV(DrzewoAVL\* root) {

if (root != NULL) {

LRV(root->left);

LRV(root->right);

cout << root->key << " ";

}

}

int main( )

{

DrzewoAVL \* root = NULL;

cr = cl = cp = " ";

cr [ 0 ] = 218; cr [ 1 ] = 196;

cl [ 0 ] = 192; cl [ 1 ] = 196;

cp [ 0 ] = 179;

int pom=1,n;//pom oblusguje interfejs, n= nr komendy z interfejsu

cout << "WCY22IY1S1 Borkowski Kamil" << endl;

cout << "Program do zarzadzania drzewem AVL" << endl;

cout << "[0] - wyjscie z programu" << endl;

cout << "[1] - wczytaj wezly z pliku(we.txt)" << endl;

cout << "[2] - dodaj wezel przez klawiature" << endl;

cout << "[3] - usuwanie wezla o podanym kluczu" << endl;

cout << "[4] - wyswietlenie sciezki podanego wezla" << endl;

cout << "[5] - kasowanie danych z drzewa" << endl;

cout << "[6] - wyswietlenie drzewa" << endl;

cout << "[7] - przeglad wierzcholkow" << endl;

cout << "=========================================" << endl;

while(pom){

cout << "Wprowadz nr operacji: ";

cin >> n;

switch (n){

case 0:

cout<<"Wychodzenie..."<<endl;

pom = 0;

break;

case 1:

{

ifstream in;

in.open("we.txt");

if(!in.is\_open())cout<<"Nie ma takiego pliku"<<endl<<endl;

int p;

while(in>>p)

{

DodajW(root,p);

}

cout<<"Drzewo zostalo usupelnione wezlami z pliku we.txt"<<endl<<endl;

in.close();

break;

}

case 2:

{

int k=0;//gdy wywola sie ponownie ta komende trzeba zmienic wartosc k z -1

cout<<"Podaj wezel(liczba,-1 konczy dodawanie wezlow)"<<endl;

while(k!=-1)

{

cin>>k;

if(k!=-1)

{

DodajW(root,k);

}

}

break;

}

case 3:

{

cout<<"Wskaz klucz wezla do usuniecia"<<endl;

int k;

cin>>k;

usunW ( root, podaj\_klucz ( root, k ) );

cout<<"Usuwanie wskazanego wezla..."<<endl<<endl;

break;

}

case 4:

{

int k;

cout << "Podaj klucz wezla do wyszukania: ";

cin >> k;

SciezkaW(root, k);

break;

}

case 5:

{

cout<<"Kasowanie danych drzewa..."<<endl<<endl;

zwolnij\_pamiec(root);

root=NULL;

break;

}

case 6:

{

if(root==NULL)cout<<"Drzewo jest puste"<<endl<<endl;;

printBT ( "", "", root );

cout<<endl;

break;

}

case 7:

{

cout << "Wezly w kolejnosci VLR: " << endl;

VLR(root);

cout<<endl<<endl;

cout << "Wezly w kolejnosci LVR: " << endl;

LVR(root);

cout<<endl<<endl;

cout << "Wezly w kolejnosci LRV: " << endl;

LRV(root);

cout<<endl<<endl;

break;

}

default:

{

cout<<"Podano bledny nr operacji"<<endl<<endl;

break;

}

}

}

zwolnij\_pamiec(root);//zwolnienie pamieci

return 0;

}

* 1. Dane wejściowe *(ekran uruchomienia programu)*

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* 1. Dane wyjściowe *(ekran uruchomienia programu)*

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

1. Wnioski *(złożoność asymptotyczna algorytmu przy użyciu notacji* ***O lub innej i inne wnioski)***

Złożoność obliczeniową programu można ogólnie określić jako O(log n) dla operacji dodawania, usuwania i wyszukiwania, gdzie n to liczba węzłów w drzewie. Operacje wypisywania mają złożoność O(n), a zwalnianie pamięci również O(n), co wynika z potrzeby odwiedzenia każdego węzła.