**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Algorytmy i struktury danych | **Zadanie** | 7\_02 |
| **Autor** | Kamil Borkowski | **Grupa** | WCY22IY1S1 |
| **Temat** | Drzewo RB | | |

1. Treść

Napisz program, który umożliwi: Budowanie drzewa RB

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 (RB)

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 wykorzystuje strukturę DrzewoRB, która reprezentuje węzeł drzewa. Operacje takie jak dodawanie, usuwanie, wyszukiwanie, wypisywanie oraz rotacje są realizowane w funkcjach pomocniczych. Interfejs użytkownika umożliwia wykonywanie tych operacji poprzez wprowadzanie numerów operacji. Dodatkowo, program obsługuje wczytywanie danych z pliku i zwalnianie pamięci po zakończeniu działania.

* 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 DrzewoRB.

2. Ustawienie początkowego korzenia (root) na NULL.

3. Implementacja funkcji DodajW do dodania węzła do drzewa.

4. Implementacja funkcji removeRBT do usunięcia węzła z drzewa.

5. Implementacja funkcji findNodeAndPrintPath do znalezienia i wypisania ścieżki.

6. Implementacja funkcji DFSRelease do zwolnienia pamięci drzewa

7. Implementacja funkcji printBT do wizualizacji struktury drzewa.

8.Implementacja funkcji VLR, LVR, LRV do wypisania wierzchołków w kolejności VLR, LVR i LRV.

9. Wyświetlenie prostego menu interakcji z użytkownikiem.

10. Oczekiwanie na wprowadzenie numeru operacji przez użytkownika.

11. Program utrzymuje pętlę, oczekując na kolejne operacje użytkownika.

12. Po zakończeniu działania programu zwalniana jest pamięć przy użyciu funkcji DFSRelease.

Kod źródłowy

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

//Zadanie 7\_02 jezyk C++

//Drzewo RB

//WCY22IY1S1 Borkowski Kamil

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

#include <iostream>

#include <string>

#include <cstdlib>

#include <fstream>

using namespace std;

struct DrzewoRB

{

DrzewoRB\* up;

DrzewoRB\* left;

DrzewoRB\* right;

int key;

char color;

};

//MODUL wizualizacji drzewa

string cr, cl, cp;

void printBT ( string sp, string sn, DrzewoRB \* 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->color << endl;

s = sp;

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

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

}

}

//funckja zwalniajaca pamiec

void DFSRelease(DrzewoRB\* p)

{

if (p != NULL)

{

DFSRelease(p->left);

DFSRelease(p->right);

free(p);

}

}

//funkcja pomocnicza zwracajaca wezel o podanym kluczu

DrzewoRB\* findRBT(DrzewoRB\* root, int k)

{

DrzewoRB\* p = root;

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

{

if (k < p->key)

p = p->left;

else

p = p->right;

}

return p;

}

//funkcja pomocnicza szukajaca najmniejszego wezla w poddrzewie

DrzewoRB\* minRBT(DrzewoRB\* p)

{

if (p != NULL)

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

return p;

}

//funkcja szukajaca zamiennika wezla, czyli najmniejszego wezla w podrzewie

DrzewoRB\* succRBT(DrzewoRB\* p)

{

DrzewoRB\* r;

if (p != NULL)

{

if (p->right != NULL)

{

return minRBT(p->right);

}

else

{

r = p->up;

while (r != NULL && p == r->right)

{

p = r;

r = r->up;

}

return r;

}

}

return NULL;

}

//funkcja rotujaca w lewo wzgledem A

void rot\_L(DrzewoRB\* A)

{

DrzewoRB\* B, \* p;

B = A->right;

if (B != NULL)

{

p = A->up;

A->right = B->left;

if (A->right != NULL)

{

A->right->up = A;

}

B->left = A;

B->up = p;

A->up = B;

if (p != NULL)

{

if (p->left == A)

{

p->left = B;

}

else p->right = B;

}

}

}

//funkcja rotujaca w prawo wzgledem A

void rot\_R(DrzewoRB\* A)

{

DrzewoRB\* B, \* p;

B = A->left;

if (B != NULL)

{

p = A->up;

A->left = B->right;

if (A->left != NULL)

{

A->left->up = A;

}

B->right = A;

B->up = p;

A->up = B;

if (p != NULL)

{

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

else p->right = B;

}

}

}

//funkcja dodajaca wezel do drzewa

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

{

DrzewoRB\* X, \* Y;

//X = new DrzewoRB;

X=(DrzewoRB\*)malloc(sizeof(DrzewoRB));

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

X->key = k;

if (root == NULL)

{

root = X;

}

else

{

Y = root;

while (true)

{

if (k < Y->key)

{

if (Y->left == NULL)

{

Y->left = X;

X->up = Y;

break;

}

Y = Y->left;

}

else

{

if (Y->right == NULL)

{

Y->right = X;

X->up = Y;

break;

}

Y = Y->right;

}

}

}

X->color = 'R';

while (X != root && X->up->color == 'R')

{

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

{

Y = X->up->up->right;

if (Y != NULL && Y->color == 'R')

{

X->up->color = 'B';

Y->color = 'B';

X->up->up->color = 'R';

X = X->up->up;

continue;

}

if (X == X->up->right)

{

X = X->up;

rot\_L(X);

}

if (X->up != NULL)

{

X->up->color = 'B';

if (X->up->up != NULL)

{

X->up->up->color = 'R';

rot\_R(X->up->up);

}

}

break;

}

else

{

Y = X->up->up->left;

if (Y != NULL && Y->color == 'R')

{

X->up->color = 'B';

Y->color = 'B';

X->up->up->color = 'R';

X = X->up->up;

continue;

}

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

{

X = X->up;

rot\_R(X);

}

if (X->up != NULL)

{

X->up->color = 'B';

if (X->up->up != NULL)

{

X->up->up->color = 'R';

rot\_L(X->up->up);

}

}

break;

}

}

if (root != NULL)

root->color = 'B';

}

// Funkcja usuwaj¹ca z drzewa wêze³ X

void removeRBT(DrzewoRB\*& root, DrzewoRB\* X)

{

DrzewoRB\* W, \* Y, \* Z;

if (X == NULL) return;

if (X->left == NULL || X->right == NULL) Y = X;

else Y = succRBT(X);

if (Y->left != NULL) Z = Y->left;

else Z = Y->right;

if (Z != NULL) Z->up = Y->up;

if (Y->up == NULL) root = Z;

else if (Y == Y->up->left) Y->up->left = Z;

else Y->up->right = Z;

if (Y != X) X->key = Y->key;

if (Y->color == 'B')

{

while (Z != root && Z->color == 'B')

{

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

{

W = Z->up->right;

if (W != NULL && W->color == 'R')

{

W->color = 'B';

Z->up->color = 'R';

rot\_L(Z->up);

W = Z->up->right;

}

if ((W->left == NULL || W->left->color == 'B') && (W->right == NULL || W->right->color == 'B'))

{

W->color = 'R';

Z = Z->up;

}

else

{

if (W->right == NULL || W->right->color == 'B')

{

if (W->left != NULL) W->left->color = 'B';

W->color = 'R';

rot\_R(W);

W = Z->up->right;

}

W->color = Z->up->color;

Z->up->color = 'B';

if (W->right != NULL) W->right->color = 'B';

rot\_L(Z->up);

Z = root;

}

}

else

{

W = Z->up->left;

if (W != NULL && W->color == 'R')

{

W->color = 'B';

Z->up->color = 'R';

rot\_R(Z->up);

W = Z->up->left;

}

if ((W->right == NULL || W->right->color == 'B') && (W->left == NULL || W->left->color == 'B'))

{

W->color = 'R';

Z = Z->up;

}

else

{

if (W->left == NULL || W->left->color == 'B')

{

if (W->right != NULL) W->right->color = 'B';

W->color = 'R';

rot\_L(W);

W = Z->up->left;

}

W->color = Z->up->color;

Z->up->color = 'B';

if (W->left != NULL) W->left->color = 'B';

rot\_R(Z->up);

Z = root;

}

}

}

if (Z != NULL) Z->color = 'B';

}

delete Y;

}

//funkcja szukajaca sciezki wezla

bool findNodeAndPrintPath(DrzewoRB\* root, int key)

{

if (!root) {

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

return false;

}

DrzewoRB\* 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(DrzewoRB\* root) {

if (root != NULL) {

LVR(root->left);

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

LVR(root->right);

}

}

//funkcja wypisujaca wezly prefiksowo

void VLR(DrzewoRB\* root) {

if (root != NULL) {

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

VLR(root->left);

VLR(root->right);

}

}

//funkcja wypisujaca wezly postfiksowo

void LRV(DrzewoRB\* root) {

if (root != NULL) {

LRV(root->left);

LRV(root->right);

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

}

}

int main()

{

DrzewoRB\* root = NULL; // Korzeñ drzewa czerwono-czarnego

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 czerwono-czarnym" << 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;

removeRBT(root, findRBT(root, k));

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

break;

}

case 4:

{

int k;

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

cin >> k;

findNodeAndPrintPath(root, k);

break;

}

case 5:

{

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

DFSRelease(root); // Usuwamy drzewo AVL z pamiêci

root=NULL;

break;

}

case 6:

{

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

printBT ( "", "", root ); // Wyœwietlamy zawartoœæ drzewa AVL

break;

}

case 7:

{

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

VLR(root);

cout<<endl;

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

LVR(root);

cout<<endl;

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

LRV(root);

cout<<endl<<endl;

break;

}

default:

{

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

break;

}

}

}

DFSRelease( 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

Opis wygenerowany automatycznie

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

Złożoność obliczeniowa programu będzie zależała od wykonywanych operacji. Dla pojedynczych operacji na drzewie czerwono-czarnym (dodawanie, usuwanie, wyszukiwanie), złożoność będzie ograniczona logarytmicznie przez liczbę węzłów w drzewie (O(log n)). W przypadku operacji, które odwiedzają wszystkie węzły (np. wizualizacja drzewa), złożoność będzie zależała liniowo od liczby węzłów (O(n)).