**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Algorytmy i struktury danych | **Zadanie** | 6\_01 |
| **Autor** | Kamil Borkowski | **Grupa** | WCY22IY1S1 |
| **Temat** | Drzewo BST | | |

1. Treść

Napisz program, który umożliwi: Budowanie drzewa BST

Program powinien:

Umożliwić wczytywanie danych z pliku wejściowego

Umożliwić wprowadzanie danych z klawiatury

Dodawanie węzła

Kasowanie węzła

Szukanie węzła (sciezka)

Kasowanie danych z całego drzewa

Narysowanie/ wyświetlenie drzewa (BST)

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

Struktura BSTNode reprezentuje węzeł drzewa, zawierając informacje o wartości klucza oraz wskaźnikiach na przodków, lewe i prawe potomki. Program oferuje interaktywny interfejs użytkownika umożliwiający dodawanie, wyszukiwanie, usuwanie węzłów oraz wyświetlanie drzewa w różnych porządkach (infiksowy, prefiksowy, postfiksowy). Funkcje operujące na drzewie, takie jak insertBST, deleteBST, findNodeAndPrintPath, zostały zaimplementowane zgodnie z zasadami drzewa binarnego poszukiwań. Program również zawiera funkcję DFSRelease do zwolniania pamięci po zakończeniu działania. Interfejs użytkownika jest obsługiwany przez pętlę while i instrukcje warunkowe, umożliwiając interaktywne zarządzanie drzewem binarnym.

* 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.Określenie struktury węzła drzewa binarnego poszukiwań (BST) z polami przechowującymi klucz (key) oraz wskaźniki na przodka (up), lewego (left) i prawego (right) potomka.

2. insertBST: Dodawanie nowego węzła do drzewa BST w odpowiednim miejscu zgodnie z zasadami drzewa binarnego poszukiwań.

3. LVR\_iterative: Iteracyjne wypisywanie węzłów drzewa w porządku infiksowym (LVR).

4. VLR\_iterative: Iteracyjne wypisywanie węzłów drzewa w porządku prefiksowym (VLR).

5. LRV: Rekurencyjne wypisywanie węzłów drzewa w porządku postfiksowym (LRV).

6. findNodeAndPrintPath: Wyszukiwanie węzła o określonym kluczu i wypisywanie ścieżki do tego węzła.

7. deleteBST: Usuwanie węzła o podanym kluczu zgodnie z zasadami drzewa binarnego poszukiwań.

8. Inicjalizacja zmiennej root jako wskaźnika na korzeń drzewa.

9. Określenie specjalnych znaków do wizualizacji drzewa w funkcji printBT.

10. Wyświetlenie interaktywnego menu dla użytkownika.

11. Umożliwienie użytkownikowi wyboru różnych operacji, takich jak dodawanie węzłów, wyświetlanie drzewa, wyszukiwanie, usuwanie i inne.

12. Wykonanie odpowiednich operacji w zależności od wyboru użytkownika.

13. Zwolnienie pamięci na koniec programu.

* 1. Kod źródłowy

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

//Zadanie 6\_01 jezyk C++

//Drzewo BST

//WCY22IY1S1 Borkowski Kamil

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

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

struct BSTNode

{

BSTNode \*up, \*left, \*right;

int key;

};

//MODUL WYSWIETLANIA DRZEWA

string cr, cl, cp;

void printBT(string sp, string sn, BSTNode\* 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 << endl;

s = sp;

if (sn == cl)

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

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

}

}

//Funkcja zwolnienia pamieci

void DFSRelease(BSTNode\* v)

{

if (v)

{

DFSRelease(v->left);

DFSRelease(v->right);

free(v);

}

}

//Funkcja dodajaca nowy wezel

void insertBST(BSTNode\*& root, int k)

{

BSTNode\* p;

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

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

w->key = k;

p = root;

if (!p)

root = w;

else

while (true)

if (k < p->key)//sprawdzamy czy przodek bedzie z lewej czy z prawej

{

if (!p->left)

{

p->left = w;

break;

}

else

p = p->left;

}

else

{

if (!p->right)

{

p->right = w;

break;

}

else

p = p->right;

}

w->up = p;//up wskazuje na przodka

}

//funkcja wypisujaca wezly infiksowo

void LVR\_iterative(BSTNode\* root)

{

if (!root) {

cout << "Drzewo jest puste." << endl;

return;

}

BSTNode\* current = root;

BSTNode\* predecessor;

while (current)

{

if (!current->left)

{

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

current = current->right;

}

else

{

predecessor = current->left;

while (predecessor->right && predecessor->right != current)

predecessor = predecessor->right;

if (!predecessor->right)

{

predecessor->right = current;

current = current->left;

}

else

{

predecessor->right = NULL;

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

current = current->right;

}

}

}

cout << endl;

}

//funkcja wypisujaca wezly prefiksowo

void VLR\_iterative(BSTNode\* root)

{

if (!root) {

cout << "Drzewo jest puste." << endl;

return;

}

BSTNode\* current = root;

while (current)

{

if (!current->left)

{

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

current = current->right;

}

else

{

BSTNode\* predecessor = current->left;

while (predecessor->right && predecessor->right != current)

predecessor = predecessor->right;

if (!predecessor->right)

{

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

predecessor->right = current;

current = current->left;

}

else

{

predecessor->right = NULL;

current = current->right;

}

}

}

cout << endl;

}

//funkcja wypisujaca wezly postfiksowo

void LRV(BSTNode\* v)

{

if (v)

{

LRV(v->left);

LRV(v->right);

cout << v->key << " ";

}

}

//funkcja szukajaca sciezki wezla

bool findNodeAndPrintPath(BSTNode\* root, int key)

{

if (!root) {

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

return false;

}

BSTNode\* current = root;

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

while (current)

{

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

if (key == current->key) {

cout << 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 usuwajaca wezwel

void deleteBST(BSTNode\*& root, int k)

{

BSTNode\* p = root;//wskaznik na usuwany wezel

BSTNode\* q = NULL;//wskaznik na przodka usuwanego wskaznika

while (p && p->key != k)//szukanie wezla o podanym kluczu

{

q = p;

if (k < p->key)

p = p->left;

else

p = p->right;

}

if (!p)//jesli nie znajdziemy danego klucza nic sie nie dzieje

return;

if (!p->left || !p->right)//usuwanie wezla z jednym albo zadnymi potomkami

{

BSTNode\* t;

if (!p->left)

t = p->right;

else

t = p->left;

if (!q)//polaczenie przodka z potomkiem usuwanego wezla

root = t;

else if (p == q->left)

q->left = t;

else

q->right = t;

if (t)

t->up = q;

free(p);

}

else//usuwanie wezla z dwoma potomkami

{

BSTNode\* r = p->right;//wskaznik na prawego potomka

while (r->left)

r = r->left;

p->key = r->key;//zamiana klucza usuwanego wezla z kluczem jego potomka

deleteBST(r, r->key);//usuniecie wszystkich potomkow usuwanego wezla z jednej strony

}

}

int main()

{

BSTNode\* root = NULL;//deklaracja korzenia drzewa

//czesc modulu wyswietlenia drzewa

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 << "Program do zarzadzania drzewem binarnym" << endl;

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

cout << "[1] - dodawanie nowego wezla do drzewa" << endl;

cout << "[2] - wyswietlenie drzewa" << endl;

cout << "[3] - wyswietlenie sciezki podanego wezla" << endl;

cout << "[4] - wyswietlanie LVR" << endl;

cout << "[5] - wyswietlanie VLR" << endl;

cout << "[6] - wyswietlanie LRV" << endl;

cout << "[7] - usuwanie wezla o podanym kluczu" << endl;

cout << "[8] - usuwanie wszystkich danych z drzewa" << endl;

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

while (pom) {

cout << "Wprowadz nr operacji: ";

cin >> n;

switch (n){

case 0:

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

pom = 0;

cout<<endl;

break;

case 1:

{

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)

{

insertBST(root,k);

}

}

break;

}

case 2:

{

if(root)printBT("", "", root);

break;

}

case 3:

{

int searchKey;

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

cin >> searchKey;

findNodeAndPrintPath(root, searchKey);

break;

}

case 4:

{

cout << "Kolejnosc IN-ORDER(LVR): ";

LVR\_iterative(root);

break;

}

case 5:

{

cout << "Kolejnosc PRE-ORDER(VLR): ";

VLR\_iterative(root);

break;

}

case 6:

{

cout << "Kolejnosc POST-ORDER(LRV): ";

LRV(root);

cout<<endl;

break;

}

case 7:

{

int k;

cout << "Podaj wartosc klucza do usuniecia: ";

cin>>k;

deleteBST(root,k);

break;

}

case 8:

{

cout << "Usunieto dane z calego drzewa"<<endl;

DFSRelease(root);

root = NULL;//ustawiamy root = NULL zeby moc wykonywac dalej operacje na drzewie

break;

}

}

}

DFSRelease(root);//zwolnienie pamieci na koniec programu

return 0;

}

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

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

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

Ostateczna złożoność programu zależy od sekwencji operacji wykonywanych przez użytkownika. W przypadku drzewa binarnego poszukiwań, optymalna złożoność występuje, gdy drzewo jest zrównoważone, co oznacza, że wysokość drzewa h jest proporcjonalna do logarytmu liczby węzłów n (h = O(log n)). Jednakże, w najgorszym przypadku, gdy drzewo jest nieskręcone, złożoność może wynieść O(n).