**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Algorytmy i struktury danych | **Zadanie** | 6\_02 |
| **Autor** | Kamil Borkowski | **Grupa** | WCY22IY1S1 |
| **Temat** | Kopiec | | |

1. Treść

Napisz program, który umożliwi: Budowanie drzewa - Kopiec

Program powinien:

Umożliwić wczytywanie danych z pliku wejściowego

Umożliwić wprowadzanie danych z klawiatury

Dodawanie węzła

Kasowanie węzła max

Kasowanie danych z całego drzewa

Narysowanie/ wyświetlenie drzewa (Kopiec)

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 kopca maksymalnego (max heap) za pomocą dynamicznie alokowanych węzłów. Korzysta z interfejsu użytkownika, umożliwiając dodawanie nowych węzłów, usuwanie węzła o największym kluczu 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. Zdefiniowanie struktury KOPNode reprezentującej węzeł kopca.

2. Zdefiniowanie struktury KOP reprezentującej sam kopiec.

3. printKOP: Wizualizacja drzewa kopca.

4. swapNodes: Zamiana miejscami kluczy dwóch węzłów.

5. freeTree: Rekurencyjne zwalnianie pamięci zajmowanej przez węzły drzewa.

6. LVR, VLR, LRV: Przechodzenie drzewa w różnych porządkach.

7. insert: Dodawanie nowego węzła do kopca, naprawa kopca po dodaniu.

8. popraw: Naprawa kopca po usunięciu elementu o największym kluczu.

9. removeRoot: Usuwanie korzenia kopca (elementu o największym kluczu) i naprawa kopca.

10. Inicjalizacja korzenia kopca.

11. Ustawienie znaków do wizualizacji drzewa.

12. Wyświetlenie menu interfejsu użytkownika z opcjami operacji.

Kod źródłowy

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

//Zadanie 6\_02 jezyk C++

//Drzewo kopiec max

//WCY22IY1S1 Borkowski Kamil

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

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

struct KOPNode {

KOPNode\* up;

KOPNode\* left;

KOPNode\* right;

int key;

};

struct KOP {

KOPNode\* root;

};

//MODUL WYSWIETLANIA DRZEWA

string cr, cl, cp;

void printKOP(string sp, string sn, KOPNode\* v)

{

string s;

if (v)

{

s = sp;

if (sn == cr)

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

printKOP(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] = ' ';

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

}

}

//funkcja zmieniajaca klucze dwoch wezlow

void swapNodes(KOPNode\* a, KOPNode\* b) {

int temp = a->key;

a->key = b->key;

b->key = temp;

}

void insert(KOP\* kop, int key){

KOPNode\* newNode = (KOPNode\*)malloc(sizeof(KOPNode));

newNode->left = NULL;

newNode->right = NULL;

newNode->up = NULL;

newNode->key = key;

if (kop->root == NULL) {//sprawdzenie czy kupiec jest pusty

kop->root = newNode;

return;

}

// Wstawianie wêzla na ostatnie miejsce

KOPNode\* TAB[100];

int start = -1;

int end = -1;

TAB[++end] = kop->root;

while (start != end) {

KOPNode\* temp = TAB[++start];

if (temp->left == NULL) {//jesli lewy potomek nie istnieje wstaw tam nowy wezel

temp->left = newNode;

newNode->up = temp;

break;

} else {

TAB[++end] = temp->left;

}

if (temp->right == NULL) {//jesli prawy potomek nie istnieje wstaw tam nowy wezel

temp->right = newNode;

newNode->up = temp;

break;

} else {

TAB[++end] = temp->right;

}

}

// Zamiana miejscami z rodzicem, jeœli nowy wêze³ jest wiêkszy

KOPNode\* current = newNode;

while (current->up != NULL && current->key > current->up->key) {

swapNodes(current, current->up);

current = current->up;

}

}

//funkcja zwalniajaca pamiec

void freeTree(KOPNode\* root) {

if (root != NULL) {

freeTree(root->left);

freeTree(root->right);

free(root);

}

}

//funkcja wypisujaca wezly infiksowo

void LVR(KOPNode\* root) {

if (root != NULL) {

LVR(root->left);

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

LVR(root->right);

}

}

//funkcja wypisujaca wezly prefiksowo

void VLR(KOPNode\* root) {

if (root != NULL) {

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

VLR(root->left);

VLR(root->right);

}

}

//funkcja wypisujaca wezly postfiksowo

void LRV(KOPNode\* root) {

if (root != NULL) {

LRV(root->left);

LRV(root->right);

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

}

}

//funkcja poprawiajaca kopeiec po usunieciu klucza maksymalnego

void popraw(KOPNode\* root) {

KOPNode\* largest = root;

KOPNode\* left = root->left;

KOPNode\* right = root->right;

if (left != NULL && left->key > largest->key)//sprawdzenie ktory potomek jest wiekszy

largest = left;

if (right != NULL && right->key > largest->key)

largest = right;

if (largest != root) {//zamiana miejscami wezlow jesli wezel nie jest najwiekszy

swapNodes(root, largest);

popraw(largest);

}

}

//funkcja usuwajaca wezel o najwiekszym kluczu

void removeRoot(KOP\* kop) {

if (kop->root == NULL) {//sprawdzenie czy kopiec jest pusty

cout << "Kopiec jest pusty." << endl;

return;

}

KOPNode\* lastNode = kop->root;//szukanie ostatniego wezla kopca

while (lastNode->right != NULL) {

lastNode = lastNode->right;

}

kop->root->key = lastNode->key;//zamiana klucza na klucz ostatniego wezla

if (lastNode->up != NULL) {//zaktualizowanie wskaznikow ostatniego wezla

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

lastNode->up->left = NULL;

else

lastNode->up->right = NULL;

}

free(lastNode);//zwolnienie pamieci po usunietym wezle

popraw(kop->root);//poprawienie kopca po usunieciu wezla

}

int main(){

//deklaracja korzenia drzewa

KOP\* kop = (KOP\*)malloc(sizeof(KOP));

kop->root = NULL;

//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 kopcem" << endl;

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

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

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

cout << "[3] - wyswietlanie LVR" << endl;

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

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

cout << "[6] - usuwanie wezla o najwiekszym kluczu" << 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)

{

insert(kop,k);

}

}

break;

}

case 2:

{

if(kop->root)printKOP("", "", kop->root);//wyswietlenie kopca

break;

}

case 3:

{

cout << "Kopiec wyswietlony jako LVR: " << endl;

LVR(kop->root);

cout << endl;

break;

}

case 4:

{

cout << "Kopiec wyswietlony jako VLR: " << endl;

VLR(kop->root);

cout << endl;

break;

}

case 5:

{

cout << "Kopiec wyswietlony jako LRV: " << endl;

LRV(kop->root);

cout << endl;

break;

}

case 6:

{

removeRoot(kop);//usuniecie wezla o maksymalnym kluczu

break;

}

}

}

freeTree(kop->root);//zwolnienie pamieci na koniec programu

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)***

Ogólna złożoność czasowa programu zależy od liczby operacji wstawiania, usuwania oraz rekurencyjnych przechodzeń drzewa. W przypadku typowych operacji w kopcu maksymalnym, złożoność czasowa wynosi O(log n) dla pojedynczej operacji. Warto jednak zauważyć, że operacje rekurencyjne takie jak freeTree mają złożoność czasową O(n), co może wpłynąć na ogólną złożoność w przypadku wielokrotnego użycia tych operacji.