МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №12**

**з дисципліни «Основи програмування»**

**на тему: «Бінарне дерево пошуку»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Нежуміра О. І.

Дніпро, 2020

**Лабораторна робота №12**

**Тема.** Бінарне дерево пошуку.

**Мета.** Засвоїти поняття ієрархічної динамічної структури. Оволодіти навичками створення бінарного дерева пошуку та виконання операцій над його елементами. Засвоїти рекурсивні методи аналізу бінарних дерев пошуку.

**Постановка задачі**

1.Загальна постановка: Розробити програму для заданого індивідуального завдання і обраного рівня складності для роботи з бінарними деревами пошуку. Структура вершини дерева містить такі поля: ключ, дані та два вказівника на піддерева–ліве і праве. Якщо піддерево відсутнє, вказівник дорівнює NULL. Дерево може мати вершини з однаковими ключами, але не може мати вершини з однаковими ключами і даними одночасно. При розробці функцій слід використовувати механізм рекурсії.

2.Індивідуальна постановка:

Рівень II (75-81 балів, добре).

Реалізувати функції:

1) додавання вершини до бінарного дерева пошуку по ключу,

2) друк вершин дерева за заданим порядком в форматі «(ключ, дані)»,

3) видалення всього дерева,

4) «дія» з Таблица 1,

5) створення бінарного дерева пошуку шляхом читання інформації з текстового файлу,

6) запису бінарного дерева пошуку в текстовий файл в такому ж форматі і порядку, як і для друку. Ім’я файлу задає користувач.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Варіант | Ключ | Дані | Дія | Друк вершин дерева |
| 9. | Об’єм ОЗУ | Модель смартфону  Діагональ | Визначення вершини з максимальним ключем | В зворотному порядку |

3.Вимоги до програми:

- валідація вхідних даних;

- програма повинна мати багатомодульну структуру, яка базується на реалізації функціонала програми.

**4.Вимоги до тексту програми:**

- специфікації програми і її функцій;

- самодокументованість коду: всі ідентифікатори повинні мати назви, що відповідають суті змінних.

**Зовнішні специфікації**

1. Вхідні дані

Вхідні дані: menu, RAM, model, diagonal, n

Формат вхідних даних

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Найменування даних | Умовне позначення | Вимоги до даних | Приклад |
| 1. | Меню | menu | ціле число | 2 |
| 2. | Ключ | RAM | ціле число | 12 |
| 3. | Назва смартфона | model | string | dfgcbdx |
| 4. | Діагональ | diagonal | float | 5 |
| 5. | Кількість елементів | n | ціле число | 3 |

2. Вихідні дані

Вхідні дані: RAM, model, diagonal

Формат вихідних даних

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Найменування даних | Умовне позначення | Вимоги до даних | Приклад |
| 1. | Ключ | RAM | ціле число | 2 |
| 2. | Назва смартфона | model | ціле число | 4 |
| 3. | Діагональ | diagonal | ціле число | 3 |

**3. Функціональні вимоги до програми**

Програма повинна реалізувати такі дії:

-друк структури;

-створення дерева;

-видалення дерева;

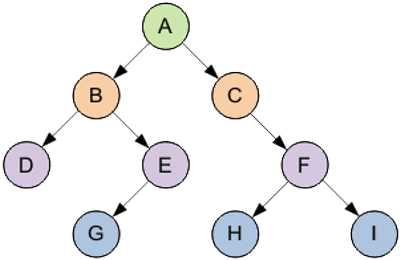
-робота с файлами;

-перевірка елемента, який ми додаємо в дерево.

**Опис структури даних, використаних для реалізації завдання**

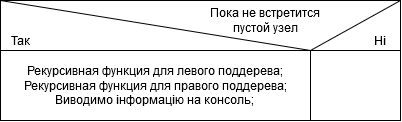
Бінарне дерево - це кінцеве безліч елементів, яке або порожня, або містить елемент (корінь), пов'язаний з двома різними бінарними деревами, званими лівим і правим піддеревами. Кожен елемент бінарного дерева називається вузлом. Зв'язки між вузлами дерева називаються його гілками.

Спосіб подання бінарного дерева:



**Алгоритм розв’язання завдання**

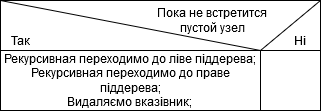
Друк дерева:



Додавання елемента до дерева:



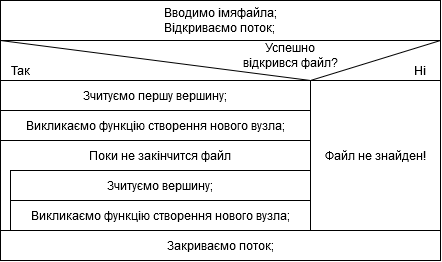
Повне видалення дерева:



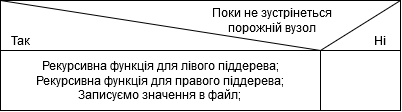
Визначити вершину з максимальним ключем:



Зчитати з файла:



Запис в файла:



**Текст програми**

#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

struct node {

int RAM;

float diagonal;

std::string smartphone\_model;

node \*left;

node \*right;

};

void push(int RAM, std::string model, float diagonal, node \*\*tree) { //Створення нової вершини дерева

if ((\*tree) == NULL) {//Якщо дерева не існує

(\*tree) = new node;

(\*tree)->RAM = RAM;

(\*tree)->smartphone\_model = model;

(\*tree)->diagonal = diagonal;

(\*tree)->left = (\*tree)->right = NULL;//Очищаємо пам'ять для наступного зростання

return;

}

if (RAM < (\*tree)->RAM)

push(RAM, model, diagonal, &(\*tree)->left);

else

push(RAM, model, diagonal, &(\*tree)->right);

}

void treeprint(node \*\*tree) {// постфіксний обхід дерева

if ((\*tree) != NULL) { //Пока не встретится пустой узел

treeprint(&(\*tree)->left); //Рекурсивная функция для левого поддерева

treeprint(&(\*tree)->right); //Рекурсивная функция для правого поддерева

std::cout << "Об'єм ОЗУ = " << (\*tree)->RAM << std::endl;

std::cout << "Модель смартфону = " << (\*tree)->smartphone\_model << std::endl;

std::cout << "Діагональ = " << (\*tree)->diagonal << std::endl << std::endl;

}

}

void freemem(node \*tree) {//Повне видалення дерева

if (tree != NULL) {

freemem(tree->left);

freemem(tree->right);

free(tree);

}

}

void file(node \*\*tree) { //Зчитуємо дерево з файла

int RAM;

float diagonal;

std::string model, namefile;

std::cout << "Введіть ім'я файла: ";

std::cin >> namefile;

std::ifstream ifs(namefile);

if (ifs.is\_open()) {

ifs >> RAM;

ifs >> model;

ifs >> diagonal;

push(RAM, model, diagonal, &(\*tree));

while (!ifs.eof()) {

ifs >> RAM;

ifs >> model;

ifs >> diagonal;

push(RAM, model, diagonal, &(\*tree));

}

}

else

std::cout << "Файл не знайден!" << std::endl;

ifs.close();

}

void file\_take(std::ostream& ifs, node \*\*tree) {//Запису бінарного дерева пошуку в текстовий файл в такому ж форматі і порядку, як і для друку

if ((\*tree) != NULL) { //Пока не встретится пустой узел

file\_take(ifs, &(\*tree)->left); //Рекурсивная функция для левого поддерева

file\_take(ifs, &(\*tree)->right); //Рекурсивная функция для правого поддерева

ifs << (\*tree)->RAM << ' ' << (\*tree)->smartphone\_model << ' ' << (\*tree)->diagonal << '\n'; //Записуємо значення в файл

}

}

node \*MaxValue(node \*tree) {//Визначення вершини з максимальним ключем

if (tree->right != NULL)

return MaxValue(tree->right);

else

return tree;

}

int input\_check(int choice) {//Перевірка значань тіпу int

int n;

while (true) {

switch (choice) {

case 1:

std::cout << "Введите количество элементов: ";

break;

case 2:

std::cout << "Об'єм ОЗУ = ";

break;

case 3:

std::cout << "Елемент який треба видалити: ";

break;

default:

break;

}

std::cin >> n;

if ((std::cin.peek() != '\n') || !(n > 0)) {

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cout << "Некоректне введення!" << std::endl;

}

else

return n;

}

}

float input\_check\_float() { //Перевірка значань тіпу float

float n;

while (true) {

std::cout << "Діагональ = ";

std::cin >> n;

if ((std::cin.peek() != '\n') || !(n > 0)) {

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cout << "Некоректне введення!" << std::endl;

}

else

return n;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

node \*tree = NULL;

int n, menu; //n - кількість елементів, menu - номер меню

int RAM;

float diagonal;

std::string model;

do {

system("cls");

std::cout << "1. Друк дерева" << std::endl;

std::cout << "2. Додати вершину до дерева" << std::endl;

std::cout << "3. Визначити вершину з максимальним ключем" << std::endl;

std::cout << "4. Зчитати дерево з файла" << std::endl;

std::cout << "5. Записати дерево в файла" << std::endl;

std::cout << "6. Вихід" << std::endl;

std::cin >> menu;

switch (menu) {

case 1: {

system("cls");

std::cout << "Дерево" << std::endl;

treeprint(&tree);

std::cout << std::endl;

system("pause");

break;

}

case 2: {

system("cls");

n = input\_check(1);

for (int i = 0; i<n; ++i) {

RAM = input\_check(2); //Ввод ключа(ОЗУ)

std::cin.ignore(32767, '\n');

std::cout << "Модель смартфону = ";

std::getline(std::cin, model);

diagonal = input\_check\_float(); //Ввод даних(діагональ)

std::cout << std::endl;

push(RAM, model, diagonal, &tree);

}

system("pause");

break;

}

case 3: {

system("cls");

std::cout << "Максимальный элемент дерева" << std::endl;;

node\* max = MaxValue(tree);

std::cout << "Об'єм ОЗУ = " << max->RAM << std::endl;

std::cout << "Модель смартфону = " << max->smartphone\_model << std::endl;

std::cout << "Діагональ = " << max->diagonal << std::endl;

system("pause");

break;

}

case 4: {

system("cls");

file(&tree); //Зчитуємо дерево з файла

system("pause");

break;

}

case 5: {

system("cls");

std::string namefile;

std::cout << "Введіть ім'я файла: ";

std::cin >> namefile;

std::ofstream ifs(namefile); //Відкритя файлу

if (ifs.is\_open())

file\_take(ifs, &tree); //Запис дерева в файл

else

std::cout << "Файл не знайден!" << std::endl;

ifs.close(); //Закриття файлу

system("pause");

break;

}

case 6:

freemem(tree); //Видалення всего дерева

return 0;

default:

break;

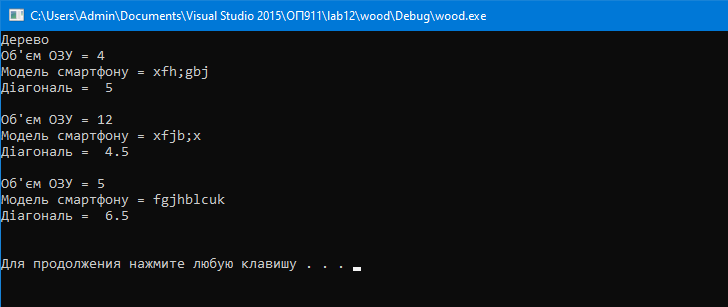
}

} while(menu != 6);

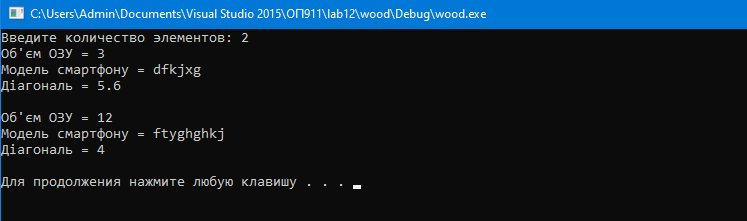
}

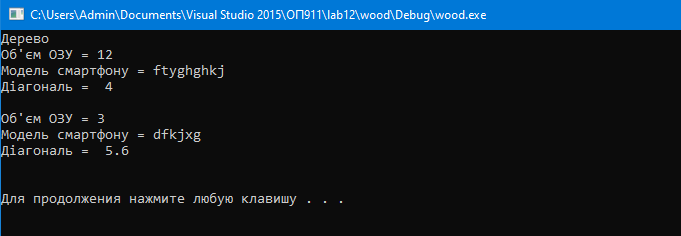
**Результати тестування програми**

Друк дерева:

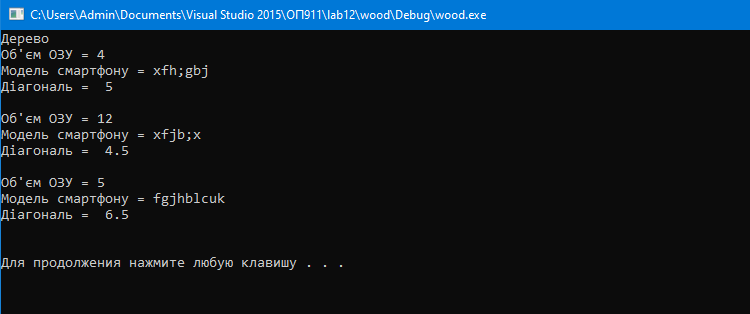


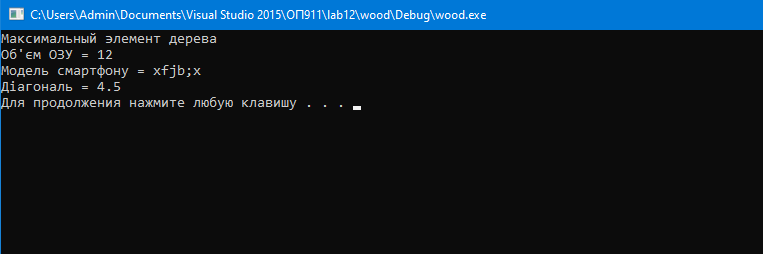
Додавання елемента до дерева:



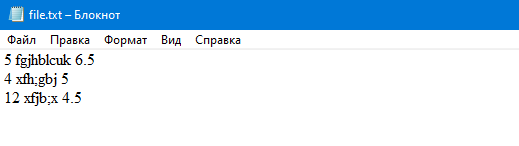


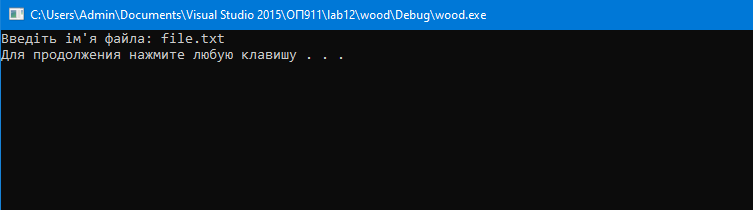
Визначити вершину з максимальним ключем:

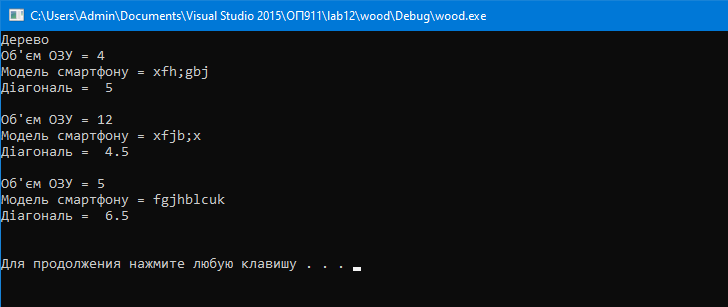




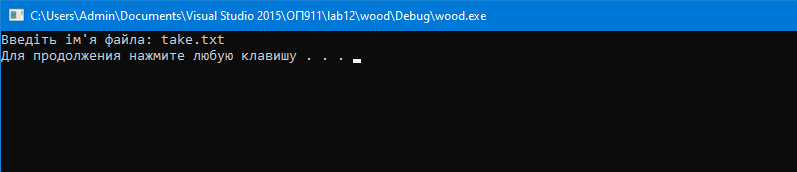
Зчитати з файла:

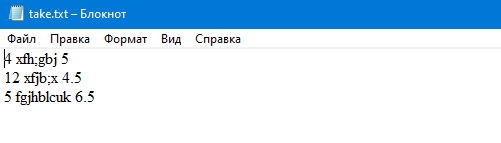






Записати дерево в файл





Висновок: Дерево пошуку представляє собою структуру даних, яка підтримує багато операцій з динамічними множинами, включаючи пошук елемента, визначення мінімального і максимального значення, попереднього і наступного елемента, операції вставка та видалення. Таким чином, дерево пошуку може використовуватися і як словник, і як черга з пріоритетами.

Основні операції в бінарному дереві пошуку виконуються за час, пропорційний його висоті. Для повного бінарного дерева з n вузлами ці операції виконуються за час θ (log2n) в найгіршому випадку. Математичне сподівання висоти побудованого випадковим чином бінарного дерева має значення О (log2n), так що всі основні операції  в такому дереві виконуються в середньому за час в (log2n).

Як випливає з назви, бінарне дерево пошуку в першу чергу є бінарним деревом, як показано на рис. 11.1. Таке дерево може бути представлене за допомогою зв'язаної структури даних, в якій кожен вузол є об'єктом. На додаток до полів ключа key і супутніх даних, кожен вузол містить поля left, right і р, які вказують на лівий і правий дочірні вузли і на батьківський вузол відповідно. Якщо дочірній або батьківський вузол відсутні, відповідне поле містить значення nil. Єдиний вузол, покажчик р якого дорівнює nil, - це кореневий вузол дерева. Ключі в бінарному дереві пошуку зберігаються таким чином, щоб в будь-який момент задовольняти наступній властивості бінарного дерева пошуку:*я*кщо х - вузол бінарного дерева пошуку, а вузол у знаходиться в лівому піддереві х, то

*key[y]≤key[x]*

*.*

*Якщо* вузол у знаходиться в правому піддереві х*,* то

*key[y]≥key[x]0*

*.*