Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Звіт

з лабораторної роботи №5

«Дерева»

з дисципліни «Алгоритми та структури даних»

Виконала:

ст.гр. ІП-22-4

Томин Є.В.

Перевірив:

Григорчук Л. І.

Івано-Франківськ

2023

# Тема. Дерева

**Мета роботи:** отримати навики застосування двійкових дерев, реалізувати основні операції над деревами: обхід дерев, включення, виключення та пошук вузлів

## **Теоретичні відомості**

Дерево - це нелінійна структура даних, що використовується для представлення ієрархічних зв'язків, що мають відношення «один до багатьох».

Дерево з базовим типом Т визначається рекурсивно або як порожня структура (порожнє дерево), або як вузол типу Т з кінцевою кількістю деревовидних структур цього ж типу, які називаються піддеревами.

Дерева використовуються при побудові організаційних діаграм, аналізі електричних ланцюгів, для представлення синтаксичних структур у компіляторах програм, для представлення структур математичних формул, організації інформації в СУБД тощо.

У пам'яті дерева можна представити у вигляді зв'язків з предками (батьками); зв'язного списку нащадків (дітей) або структури даних.

Якщо у кожної вершини дерева є не більше двох нащадків (ліві і праві піддерева), то таке дерево називається двійковим або бінарним. Двійкові дерева широко використовуються у програмуванні.

Основні операції з деревами: обхід дерева, пошук по дереву, включення в дерево, виключення з дерева.

Обхід (відвідування) вершин дерева можна здійснити наступним чином:

* Зліва направо: A, R, B   (інфіксний обхід, симетричний обхід) .
* Зверху вниз: R, A, B   (префіксній обхід, обхід в прямому порядку).
* Знизу вверх: A, B, R   (постфіксний обхід, обхід в зворотному порядку).

Для реалізації алгоритмів пошуку використовуються дерева двійкового пошуку. Дерево двійкового пошуку - це таке дерево, в якому всі ліві нащадки молодші за предка, а всі праві - старші. Ця властивість називається характеристичною властивістю дерева двійкового пошуку і виконується для будь-якого вузла, включаючи корінь.

## 

## **5.2. Порядок виконання роботи**

Побудувати дерево відповідно до варіанту. Вивести його на екран у вигляді дерева. Реалізувати основні операції роботи з деревом: обхід дерева, включення, виключення та пошук вузлів. Оформити їх у вигляді функцій.

**При розробці інтерфейсу програми слід передбачити:**

        вказати тип, формат і діапазон даних, що вводяться;

        вказати дії, що здійснює програма;

        наявність пояснень при виведенні результату;

       відображення дерева візуалізувати.

**Під час тестування програми необхідно:**

o      перевірити правильність введення і виведення даних (тобто їх відповідність необхідному типу і формату). Забезпечити адекватну реакцію програми на невірне введення даних;

o      забезпечити виведення повідомлень за відсутності вхідних даних («порожнє введення»);

o      перевірити правильність виконання операцій;

o     забезпечити можливість додавання вузла в порожнє дерево;

o     передбачити відображення повідомлення при спробі видалити вузол з порожнього дерева;

o      перевірити різні випадки включення і виключення вузла в існуючому дереві;

o      перевірити пошук існуючого вузла та пошук неіснуючого вузла в дереві.

## **5.3. Завдання**

1. Побудувати двійкове дерево пошуку з цілих чисел, що вводиться. Вивести його на екран у вигляді дерева. Знайти вершину, яка містить задане число. Визначити максимальний елемент в цьому дереві.

2. Побудувати двійкове дерево пошуку з букв рядка, що вводиться. Вивести його на екран у вигляді дерева. Знайти букви, що зустрічаються більше одного разу. Видалити з дерева ці літери. Вивести елементи дерева, що залишилися, при його постфіксному обході.

3. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться слова з текстового файлу. Вивести його на екран у вигляді дерева. Визначити кількість вершин дерева, що містять слова, які починаються на зазначену букву. Видалити з дерева ці вершини.

**Виконання**

**Завдання 1**

#include <iostream>

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) : data(value), left(nullptr), right(nullptr) {}

};

class BinarySearchTree {

public:

BinarySearchTree() : root(nullptr) {}

void Insert(int value) {

root = InsertRecursive(root, value);

}

bool Search(int value) {

return SearchRecursive(root, value);

}

int FindMax() {

return FindMaxRecursive(root);

}

void PrintTree() {

PrintTreeRecursive(root, 0);

}

private:

Node\* root;

Node\* InsertRecursive(Node\* node, int value) {

if (node == nullptr) {

return new Node(value);

}

if (value < node->data) {

node->left = InsertRecursive(node->left, value);

}

else if (value > node->data) {

node->right = InsertRecursive(node->right, value);

}

return node;

}

bool SearchRecursive(Node\* node, int value) {

if (node == nullptr) {

return false;

}

if (value == node->data) {

return true;

}

if (value < node->data) {

return SearchRecursive(node->left, value);

}

else {

return SearchRecursive(node->right, value);

}

}

int FindMaxRecursive(Node\* node) {

while (node->right != nullptr) {

node = node->right;

}

return node->data;

}

void PrintTreeRecursive(Node\* node, int level) {

if (node == nullptr) {

return;

}

PrintTreeRecursive(node->right, level + 1);

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

std::cout << " ";

}

std::cout << node->data << "\n";

PrintTreeRecursive(node->left, level + 1);

}

};

int main() {

BinarySearchTree tree;

char continueInput = 'y';

while (continueInput == 'y' || continueInput == 'Y') {

int inputValue;

std::cout << "Введіть число для вставки в дерево: ";

std::cin >> inputValue;

tree.Insert(inputValue);

std::cout << "Продовжити введення чисел? (y/n): ";

std::cin >> continueInput;

}

// Виведення дерева на екран

std::cout << "Binary Search Tree:" << std::endl;

tree.PrintTree();

int valueToSearch;

std::cout << "Введіть число для пошуку: ";

std::cin >> valueToSearch;

if (tree.Search(valueToSearch)) {

std::cout << "Знайдено " << valueToSearch << " у дереві." << std::endl;

}

else {

std::cout << valueToSearch << " не знайдено у дереві." << std::endl;

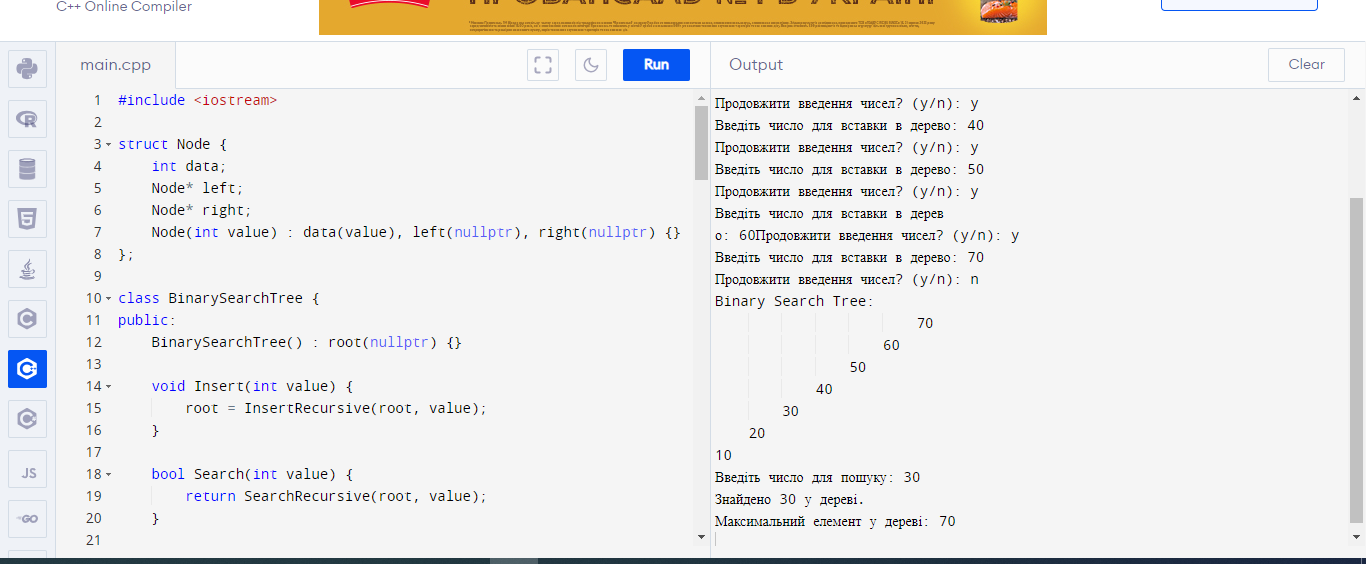
}

int maxValue = tree.FindMax();

std::cout << "Максимальний елемент у дереві: " << maxValue << std::endl;

return **0;**

**}**

 **Мал.1.** Двійкове дерево пошуку з цілих чисел, що вводиться

**Завдання 2**

#include <iostream>

#include <string>

#include <unordered\_map>

struct Node {

char data;

Node\* left;

Node\* right;

Node(char value) : data(value), left(nullptr), right(nullptr) {}

};

class BinarySearchTree {

public:

BinarySearchTree() : root(nullptr) {}

void Insert(char value) {

root = InsertRecursive(root, value);

}

void PrintTree() {

PrintTreeRecursive(root, 0);

}

void FindAndRemoveDuplicateLetters() {

std::unordered\_map<char, int> letterCount;

FindAndRemoveDuplicateLettersRecursive(root, letterCount);

}

void PostorderTraversal() {

PostorderTraversalRecursive(root);

}

private:

Node\* root;

Node\* InsertRecursive(Node\* node, char value) {

if (node == nullptr) {

return new Node(value);

}

if (value < node->data) {

node->left = InsertRecursive(node->left, value);

}

else if (value > node->data) {

node->right = InsertRecursive(node->right, value);

}

return node;

}

void PrintTreeRecursive(Node\* node, int level) {

if (node == nullptr) {

return;

}

PrintTreeRecursive(node->right, level + 1);

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

std::cout << " ";

}

std::cout << node->data << "\n";

PrintTreeRecursive(node->left, level + 1);

}

void FindAndRemoveDuplicateLettersRecursive(Node\* node, std::unordered\_map<char, int>& letterCount) {

if (node == nullptr) {

return;

}

letterCount[node->data]++;

if (letterCount[node->data] > 1) {

node->data = '\0'; // Mark duplicate letters for removal

}

FindAndRemoveDuplicateLettersRecursive(node->left, letterCount);

FindAndRemoveDuplicateLettersRecursive(node->right, letterCount);

}

void PostorderTraversalRecursive(Node\* node) {

if (node == nullptr) {

return;

}

PostorderTraversalRecursive(node->left);

PostorderTraversalRecursive(node->right);

if (node->data != '\0') {

std::cout << node->data << " ";

}

}

};

int main() {

BinarySearchTree tree;

std::string input;

std::cout << "Введіть рядок букв: ";

std::cin >> input;

for (char c : input) {

tree.Insert(c);

}

// Виведення дерева на екран

std::cout << "Binary Search Tree:" << std::endl;

tree.PrintTree();

// Знайти та видалити дубльовані букви

tree.FindAndRemoveDuplicateLetters();

// Виведення букв, які залишилися після видалення дубльованих

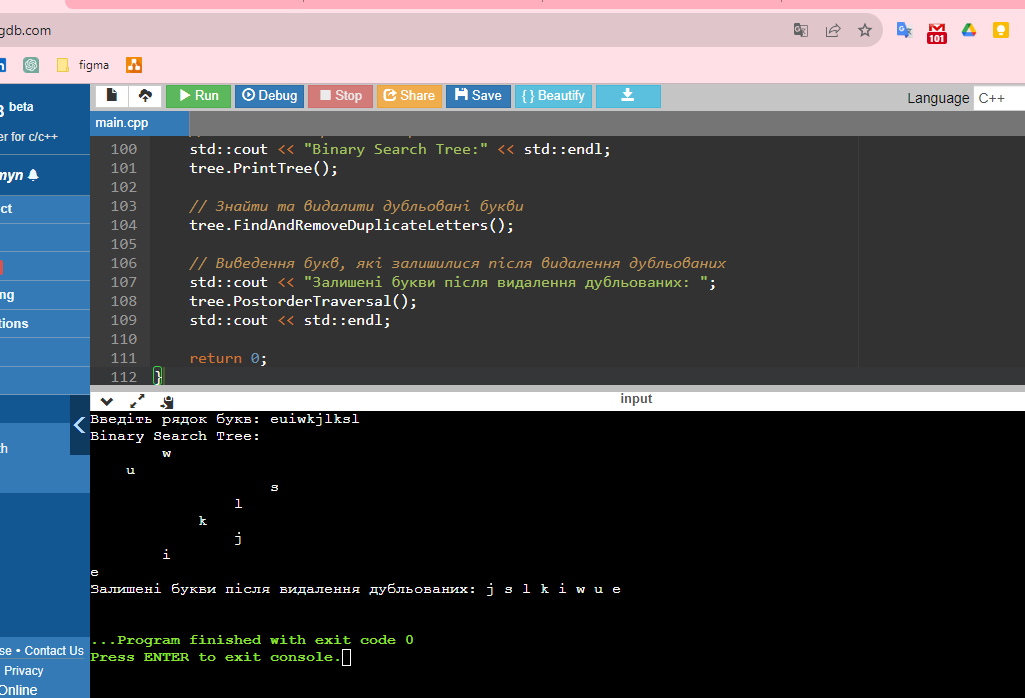
std::cout << "Залишені букви після видалення дубльованих: ";

tree.PostorderTraversal();

std::cout << std::endl;

return 0;

}



**Мал.2.** Двійкове дерево пошуку з букв рядка, що вводиться.

**Завдання 3**

#include <iostream>

#include <fstream>

#include <string>

class Node {

public:

std::string data;

Node\* left;

Node\* right;

Node(const std::string& value) : data(value), left(nullptr), right(nullptr) {}

};

class BinarySearchTree {

public:

BinarySearchTree() : root(nullptr) {}

void Insert(const std::string& value) {

root = InsertRecursive(root, value);

}

void PrintTree() {

PrintTreeRecursive(root, 0);

}

int CountWordsStartingWith(char letter) {

return CountWordsStartingWithRecursive(root, letter);

}

void RemoveWordsStartingWith(char letter) {

root = RemoveWordsStartingWithRecursive(root, letter);

}

private:

Node\* root;

Node\* InsertRecursive(Node\* node, const std::string& value) {

if (node == nullptr) {

return new Node(value);

}

if (value < node->data) {

node->left = InsertRecursive(node->left, value);

}

else if (value > node->data) {

node->right = InsertRecursive(node->right, value);

}

return node;

}

void PrintTreeRecursive(Node\* node, int level) {

if (node == nullptr) {

return;

}

PrintTreeRecursive(node->right, level + 1);

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

std::cout << " ";

}

std::cout << node->data << "\n";

PrintTreeRecursive(node->left, level + 1);

}

int CountWordsStartingWithRecursive(Node\* node, char letter) {

if (node == nullptr) {

return 0;

}

int count = 0;

if (node->data.front() == letter) {

count = 1;

}

count += CountWordsStartingWithRecursive(node->left, letter);

count += CountWordsStartingWithRecursive(node->right, letter);

return count;

}

Node\* RemoveWordsStartingWithRecursive(Node\* node, char letter) {

if (node == nullptr) {

return nullptr;

}

if (node->data.front() == letter) {

// Remove this node and reorganize the tree

Node\* left = node->left;

Node\* right = node->right;

delete node;

if (right == nullptr) {

return left;

}

Node\* minNode = FindMin(right);

minNode->right = RemoveMin(right);

minNode->left = left;

return BalanceTree(minNode);

}

if (node->data.front() < letter) {

node->right = RemoveWordsStartingWithRecursive(node->right, letter);

}

else {

node->left = RemoveWordsStartingWithRecursive(node->left, letter);

}

return BalanceTree(node);

}

Node\* FindMin(Node\* node) {

while (node->left != nullptr) {

node = node->left;

}

return node;

}

Node\* RemoveMin(Node\* node) {

if (node->left == nullptr) {

return node->right;

}

node->left = RemoveMin(node->left);

return BalanceTree(node);

}

Node\* BalanceTree(Node\* node) {

// Implement tree balancing logic here (e.g., AVL or Red-Black balancing)

return node;

}

};

int main() {

std::ofstream file("words.txt"); // Відкрити файл для запису

if (!file) {

std::cerr << "Failed to open the file." << std::endl;

return 1;

}

// Записати дані до файлу

file << "Mother" << std::endl;

file << "dad" << std::endl;

file << "brother" << std::endl;

file << "me" << std::endl;

file << "grandma" << std::endl;

file << "grandpa" << std::endl;

file.close(); // Закрити файл

BinarySearchTree tree;

std::ifstream inputFile("words.txt");

if (!inputFile) {

std::cerr << "Failed to open the file." << std::endl;

return 1;

}

std::string word;

while (inputFile >> word) {

tree.Insert(word);

}

// Виведення дерева на екран

std::cout << "Binary Search Tree:" << std::endl;

tree.PrintTree();

char letterToSearch = 'b'; // Змініть літеру на потрібну

int wordCount = tree.CountWordsStartingWith(letterToSearch);

std::cout << "Кількість слів, які починаються з букви '" << letterToSearch << "': " << wordCount << std::endl;

tree.RemoveWordsStartingWith(letterToSearch);

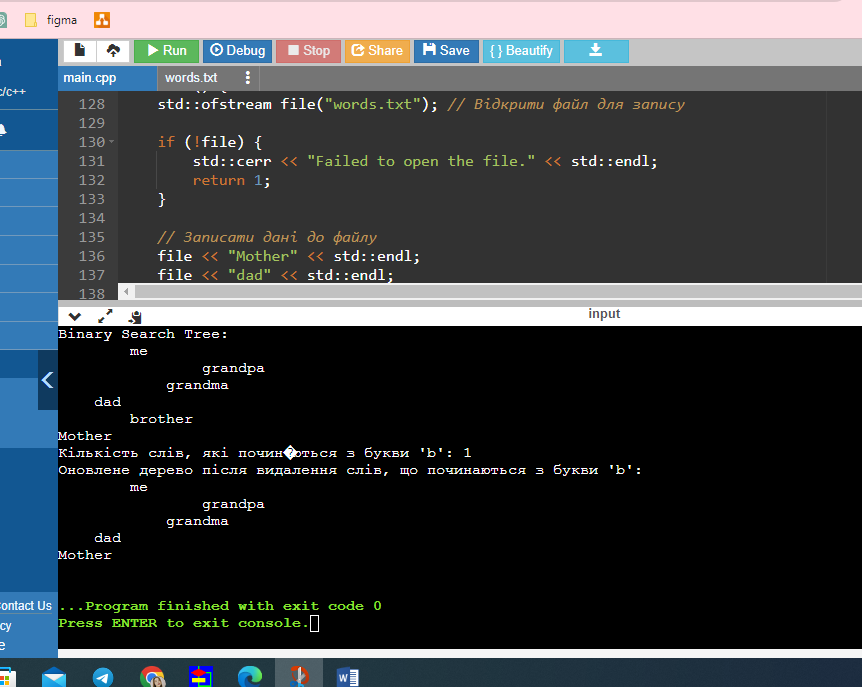
// Виведення оновленого дерева

std::cout << "Оновлене дерево після видалення слів, що починаються з букви '" << letterToSearch << "':" << std::endl;

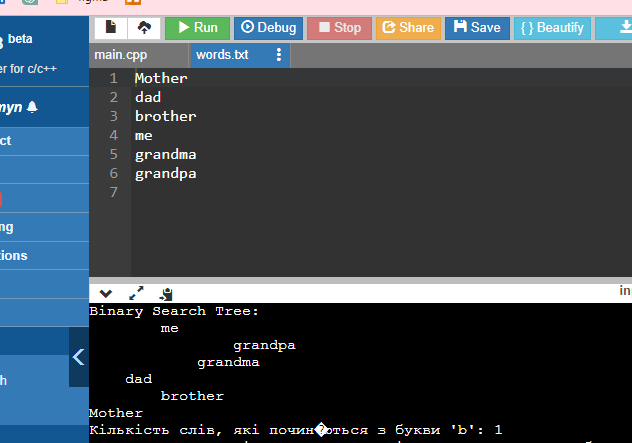
tree.PrintTree();

return 0;

}

****

**Мал.3.** Двійкове дерево пошуку, в вершинах якого знаходяться слова з текстового файлу «words.txt.»

****

**Мал.3.1.** Файл із записаними словами.

**Висновок.**

Виконуючи лабораторну роботу, я навчилася застосовувати двійкові дерева, реалізувати основні операції над деревами: обхід дерев, включення, виключення та пошук вузлів.