#include <iostream>

#include <string>

#include <queue>

#include <Windows.h>

#include <unordered\_set>

#include <fstream>

#include <limits>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

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

};

class BST {

public:

Node\* root;

BST() : root(nullptr) {}

// Додати вузол до дерева

void insert(int value) {

root = insertRecursive(root, value);

}

// Рекурсивна функція для додавання вузла

Node\* insertRecursive(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insertRecursive(root->left, value);

}

else if (value > root->data) {

root->right = insertRecursive(root->right, value);

}

return root;

}

// Вивести дерево на екран

void printTree() {

printTreeRecursive(root, 0);

}

// Рекурсивна функція для виведення дерева

void printTreeRecursive(Node\* root, int level) {

if (root != nullptr) {

printTreeRecursive(root->right, level + 1);

for (int i = 0; i < level; i++) std::cout << " ";

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

printTreeRecursive(root->left, level + 1);

}

}

// Знайти вузол з заданим значенням

Node\* search(int value) {

return searchRecursive(root, value);

}

// Рекурсивна функція для пошуку вузла з заданим значенням

Node\* searchRecursive(Node\* root, int value) {

if (root == nullptr || root->data == value) {

return root;

}

if (value < root->data) {

return searchRecursive(root->left, value);

}

else {

return searchRecursive(root->right, value);

}

}

// Знайти максимальний елемент у дереві

int findMax() {

return findMaxRecursive(root);

}

// Рекурсивна функція для знаходження максимального елементу

int findMaxRecursive(Node\* root) {

if (root == nullptr) {

// Повертаємо дефолтне значення для порожнього дерева

return std::numeric\_limits<int>::denorm\_min();

}

while (root->right != nullptr) {

root = root->right;

}

return root->data;

}

};

struct TreeNode {

char data;

TreeNode\* left;

TreeNode\* right;

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

};

class BinarySearchTree {

public:

TreeNode\* root;

BinarySearchTree() : root(nullptr) {}

// Додати вузол до дерева

void insert(char value) {

root = insertRecursive(root, value);

}

// Рекурсивна функція для додавання вузла

TreeNode\* insertRecursive(TreeNode\* root, char value) {

if (root == nullptr) {

return new TreeNode(value);

}

if (value < root->data) {

root->left = insertRecursive(root->left, value);

}

else if (value > root->data) {

root->right = insertRecursive(root->right, value);

}

return root;

}

// Вивести дерево на екран у вигляді постфіксного обходу

void printPostOrder(TreeNode\* root) {

if (root != nullptr) {

printPostOrder(root->left);

printPostOrder(root->right);

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

}

}

// Знайти букви, що зустрічаються більше одного разу та видалити їх

void removeDuplicateLetters() {

std::unordered\_set<char> seen;

root = removeDuplicateLettersRecursive(root, seen);

}

// Рекурсивна функція для видалення букв, які зустрічаються більше одного разу

TreeNode\* removeDuplicateLettersRecursive(TreeNode\* root, std::unordered\_set<char>& seen) {

if (root == nullptr) {

return nullptr;

}

// Видаляємо літери, які зустрічаються більше одного разу

if (seen.count(root->data) > 0) {

return removeDuplicateLettersRecursive(root->left, seen);

}

seen.insert(root->data);

root->left = removeDuplicateLettersRecursive(root->left, seen);

root->right = removeDuplicateLettersRecursive(root->right, seen);

return root;

}

};

struct MyTreeNode {

std::string data;

MyTreeNode\* left;

MyTreeNode\* right;

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

};

class MyBinarySearchTree {

public:

MyTreeNode\* root;

MyBinarySearchTree() : root(nullptr) {}

// Додати вузол до дерева

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

root = insertRecursive(root, value);

}

// Рекурсивна функція для додавання вузла

MyTreeNode\* insertRecursive(MyTreeNode\* root, const std::string& value) {

if (root == nullptr) {

return new MyTreeNode(value);

}

if (value < root->data) {

root->left = insertRecursive(root->left, value);

}

else if (value > root->data) {

root->right = insertRecursive(root->right, value);

}

return root;

}

// Вивести дерево на екран

void printTree() {

printTreeRecursive(root, 0);

}

// Рекурсивна функція для виведення дерева

void printTreeRecursive(MyTreeNode\* root, int level) {

if (root != nullptr) {

printTreeRecursive(root->right, level + 1);

for (int i = 0; i < level; i++) std::cout << " ";

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

printTreeRecursive(root->left, level + 1);

}

}

// Підрахувати кількість вершин дерева, що містять слова, які починаються на задану букву

int countNodesStartingWith(char letter) {

return countNodesStartingWithRecursive(root, letter);

}

// Рекурсивна функція для підрахунку кількості вершин, що містять слова, які починаються на задану букву

int countNodesStartingWithRecursive(MyTreeNode\* root, char letter) {

if (root == nullptr) {

return 0;

}

int count = 0;

if (root->data[0] == letter) {

count++;

}

count += countNodesStartingWithRecursive(root->left, letter);

count += countNodesStartingWithRecursive(root->right, letter);

return count;

}

// Видалити вершини, що містять слова, які починаються на задану букву

void removeNodesStartingWith(char letter) {

root = removeNodesStartingWithRecursive(root, letter);

}

// Рекурсивна функція для видалення вершин, що містять слова, які починаються на задану букву

MyTreeNode\* removeNodesStartingWithRecursive(MyTreeNode\* root, char letter) {

if (root == nullptr) {

return nullptr;

}

if (root->data[0] == letter) {

root = deleteNode(root);

// Після видалення поточної вершини, рекурсивно викликаємо функцію для лівого та правого піддерева

root = removeNodesStartingWithRecursive(root, letter);

}

else {

root->left = removeNodesStartingWithRecursive(root->left, letter);

root->right = removeNodesStartingWithRecursive(root->right, letter);

}

return root;

}

// Видалити вузол та повернути новий корінь піддерева

MyTreeNode\* deleteNode(MyTreeNode\* root) {

if (root->left == nullptr) {

MyTreeNode\* temp = root->right;

delete root;

return temp;

}

else if (root->right == nullptr) {

MyTreeNode\* temp = root->left;

delete root;

return temp;

}

// Якщо вузол має два дочірніх вузли

MyTreeNode\* successor = findMin(root->right);

root->data = successor->data;

root->right = removeNodesStartingWithRecursive(root->right, successor->data[0]);

return root;

}

// Знайти мінімальний елемент у піддереві

MyTreeNode\* findMin(MyTreeNode\* root) {

while (root->left != nullptr) {

root = root->left;

}

return root;

}

};

int choice = 0;

int main() {

BST bst;

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

Node\* foundNode = nullptr;

BinarySearchTree binarySearchTree;

std::string inputString;

MyBinarySearchTree myBST;

int count;

ifstream file("words.txt");

string word;

char letter;

do {

cout << "1. Ввід чисел в перше дерево\n";

cout << "2. Вивід першого дерева на екран\n";

cout << "3. Знайти певну вершину дерева\n";

cout << "4. Ввід рядка в друге дерево\n";

cout << "5. Вивід другого дерева на екран без букв, що зустрічаються більше одного разу в другому дереві\n";

cout << "6. Ввід слів в третє дерево через файл\n";

cout << "7. Вивід третього дерева на екран\n";

cout << "8. Визначити кількість вершин дерева, що містять слова, які починаються на зазначену букву\n";

cout << "9. Видалити з дерева вершини, що містять слова, які починаються на зазначену букву\n";

cout << "10. Вихід\n";

std::cin >> choice;

switch (choice) {

case 1: {

// Ввести числа та додати їх до дерева

int num;

std::cout << "Введіть цілі числа для додавання до дерева (введіть -1, щоб завершити): ";

while (cin >> num && num != -1) {

bst.insert(num);

}

break;

}

case 2:

// Вивести дерево на екран

std::cout << "\nПерше Дерево:\n";

bst.printTree();

break;

case 3: {

// Пошук вузла з заданим числом

int num;

std::cout << "\n\nВведіть число для пошуку в дереві: ";

std::cin >> num;

foundNode = bst.search(num);

if (foundNode != nullptr) {

std::cout << "Знайдено вершину з числом " << num << "\n";

// Знаходження максимального елементу

int maxElement = bst.findMax();

std::cout << "Максимальний елемент у дереві: " << maxElement << "\n";

}

else {

cout << "Не знайдено вершину з числом " << num << "\n";

}

break;

}

case 4: {

cout << "Введіть рядок для побудови дерева: ";

cin >> inputString;

cout << "Дерево створено" << "\n";

break;

}

case 5:

for (char c : inputString) {

binarySearchTree.insert(c);

}

// Вивести дерево на екран у вигляді постфіксного обходу після видалення дублікатів

cout << "Друге Дерево після видалення дублікатів: ";

binarySearchTree.printPostOrder(binarySearchTree.root);

cout << "\n";

break;

case 6:

// Зчитати слова з текстового файлу та додати їх до дерева

cout << "Зчитано дані з файлу" <<"\n";

while (file >> word) {

myBST.insert(word);

}

file.close();

break;

case 7:

// Вивести дерево на екран

cout << "Третє Дерево:\n";

myBST.printTree();

break;

case 8:

// Ввести букву для пошуку та видалення вершин

cout << "\n\nВведіть букву для пошуку та видалення вершин: ";

cin >> letter;

// Підрахувати та вивести кількість вершин

count = myBST.countNodesStartingWith(letter);

cout << "Кількість вершин, що містять слова, які починаються на букву '" << letter << "': " << count << "\n";

break;

case 9:

// Видалити вершини

cout << "Вершину видалено" << "\n";

myBST.removeNodesStartingWith(letter);

break;

default:

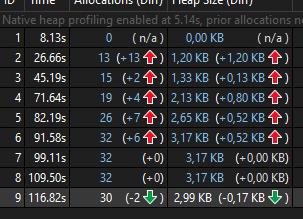
break;

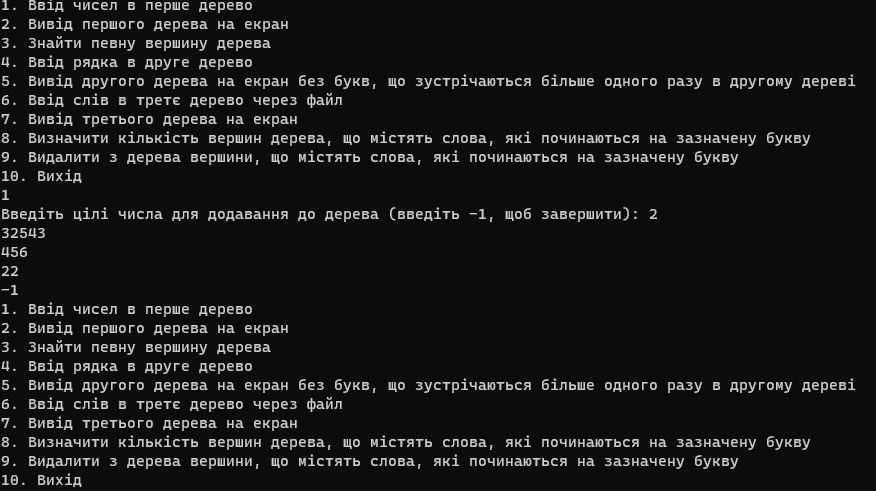
}

} while (choice != 10);

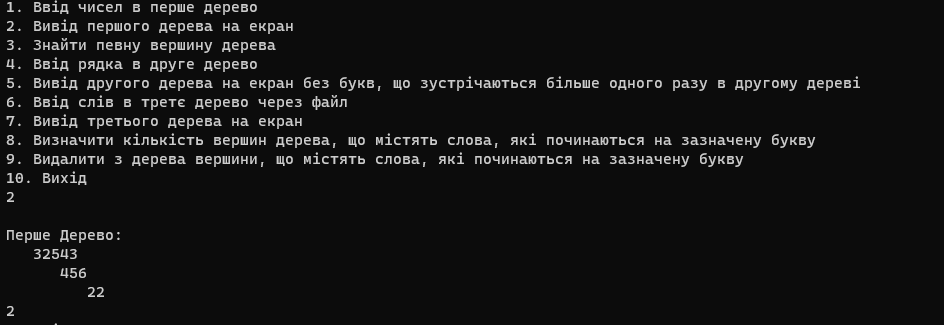
return 0;

}

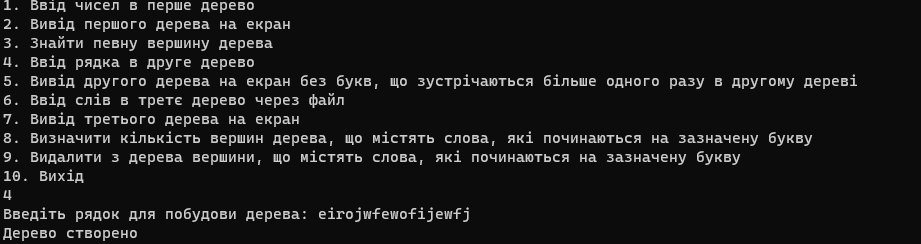




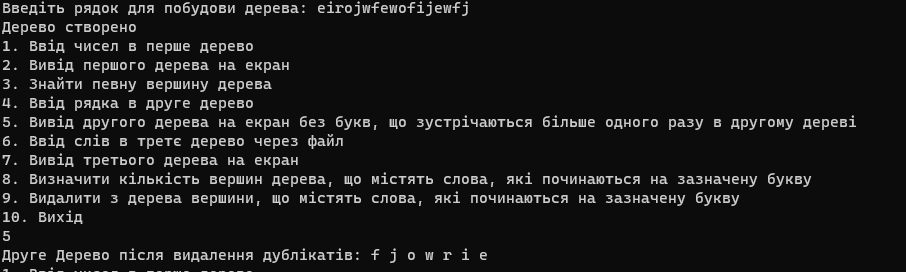
Ввід чисел в перше дерево



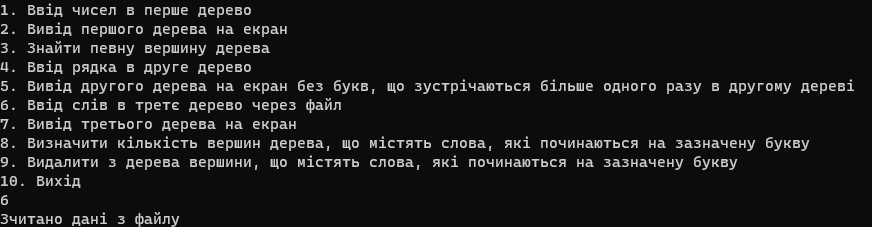
Вивід першого дерева



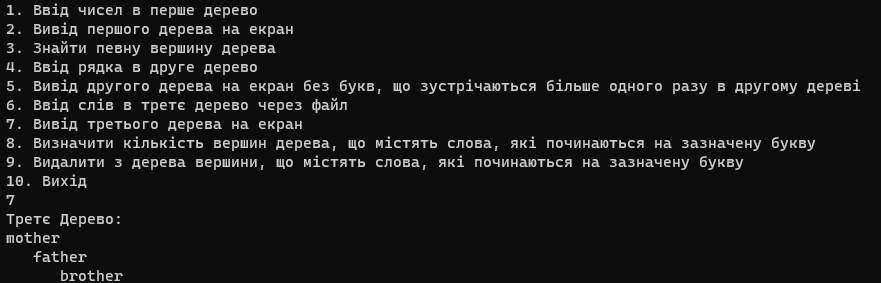
Введення рядка для другого дерева



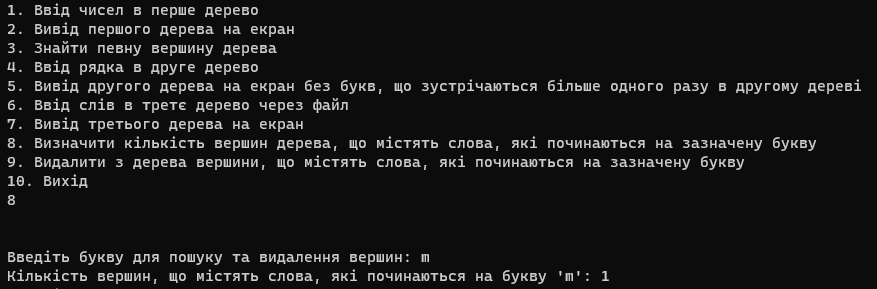
Вивід другого дерева після знищення дублікантів



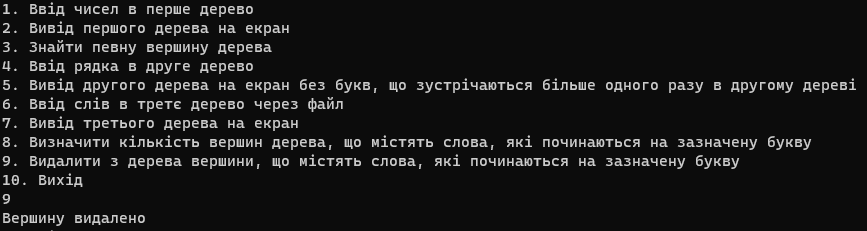
Отримання слів з файлу для третього дерева



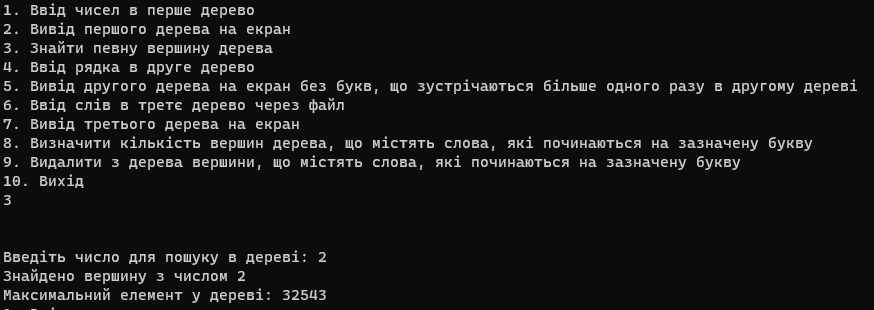
Вивід третього дерева



Визначення кількості вершин, що починаються на вказану літеру



Видалення вершини зі вказаною літерою



Отримання інформації про вказане число і найбільше число в першому дереві