Лабораторная работа №4

Дерево поиска. Частотный словарь.

Составить программу, реализующую частотный словарь текста.

Программа должна производить следующие действия;

1. Выделять в выбранном текстовом файле отдельные слова.

2. Помещать слова в дерево поиска.

3. Выводить из дерева частотный словарь слов в алфавитном порядке.

4. Выводить частотный словарь слов, отсортированный в порядке частоты их употребления.

5. Производить поиск заданного слова.

6. Производить фильтрацию, оставляющую в словаре слова определенной длины.

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <Windows.h>

#include <algorithm>

using namespace std;

struct node {

string word;

int k;

node\* left, \* right;

node(const string& w) : word(w), k(1), left(nullptr), right(nullptr) {} // Constructor

};

// Функция для освобождения памяти, занимаемой деревом

void deleteTree(node\* root) {

if (root != nullptr) {

deleteTree(root->left);

deleteTree(root->right);

delete root;

}

}

void add(node\*& root, string word) {

if (root == nullptr) {

root = new node(word); // Using constructor

}

else if (word < root->word) {

add(root->left, word);

}

else if (word > root->word) {

add(root->right, word);

}

else {

root->k++;

}

}

void printTree(node\* root, string filename) {

ofstream f(filename, ios::app); //append to file

if (root != nullptr) {

printTree(root->left, filename);

f << root->word << " - " << root->k;

f << '\n';

printTree(root->right, filename);

}

}

void treeToVector(node\* root, vector<pair<string, int>>& vec) {

if (root != nullptr) {

treeToVector(root->left, vec);

vec.push\_back(make\_pair(root->word, root->k));

treeToVector(root->right, vec);

}

}

void printSortedByFrequency(node\* root) {

vector<pair<string, int>> wordFrequencies;

treeToVector(root, wordFrequencies);

sort(wordFrequencies.begin(), wordFrequencies.end(),

[](const auto& a, const auto& b) {

return a.second > b.second;

});

ofstream file("freq sorted.txt");

for (const auto& pair : wordFrequencies) {

file << pair.first << " - " << +pair.second;

file << '\n';

}

file.close();

}

bool search(node\* root, string word) {

if (root != nullptr) {

if (word < root->word) {

return search(root->left, word);

}

else if (word > root->word) {

return search(root->right, word);

}

else {

cout << root->word << "-" << root->k << " ";

return true;

}

}

return false;

}

node\* filter(node\* root, int count) {

node\* newTree = nullptr;

vector<pair<string, int>> wordFrequencies;

treeToVector(root, wordFrequencies);

for (const auto& pair : wordFrequencies) {

if (pair.first.length() == count) {

add(newTree, pair.first);

}

}

return newTree;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

ifstream file;

string w;

file.open("text.txt");

if (file.is\_open()) {

node\* Tree = nullptr;

while (file >> w) {

add(Tree, w);

}

file.close();

int choice;

string searchWord;

int filterLength;

do {

cout << "\nMenu:\n";

cout << "1. Print dictionary (alphabetical)\n";

cout << "2. Print dictionary (frequency)\n";

cout << "3. Search word\n";

cout << "4. Filter by length\n";

cout << "0. Exit\n";

cout << "Enter your choice: ";

cin.clear();

cin >> choice;

ofstream treeFile;

string filename;

string datafile;

switch (choice) {

case 1:

printTree(Tree, "abc sorted.txt");

cout << endl << "Successfully sorted! See at \"abc sorted.txt\". ";

cout << endl;

break;

case 2:

cout << endl << "Successfully sorted! See at \"freq sorted.txt\". ";

printSortedByFrequency(Tree);

cout << endl;

break;

case 3:

cout << "Enter word to search: ";

cin >> searchWord;

cout <<endl<< "Search result: ";

if (!search(Tree, searchWord)) {

cout << endl << "No match found." << endl;

}

else {

cout << endl;

}

break;

case 4: {

cout << "Enter length to filter: ";

cin >> filterLength;

if (cin.fail())

{

cout << endl << "incorrect length input!" << endl;

break;

}

node\* newTree = filter(Tree, filterLength);

deleteTree(Tree); // Free the old tree

Tree = newTree; // Assign the new tree

cout << endl << "Successfully filtred! See at \"filtred tree.txt\". ";

printTree(Tree, "filtered tree.txt");

cout << endl;

break;

}

case 0:

cout << endl << "Exiting...\n";

break;

default:

cout << endl << "Invalid choice.\n";

}

} while (choice != 0);

deleteTree(Tree); // Free memory before exiting

}

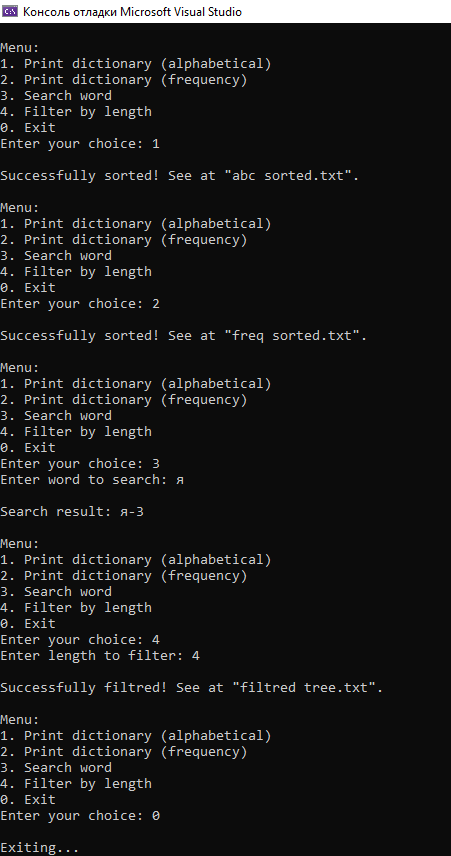
else {

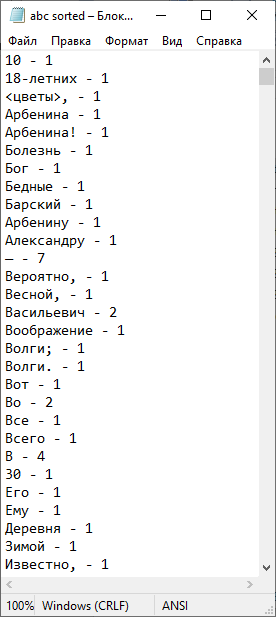
cout << "File is empty." << endl;

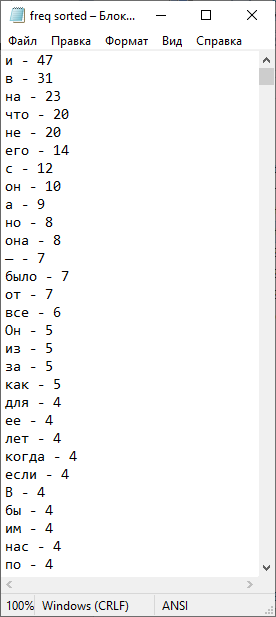
}

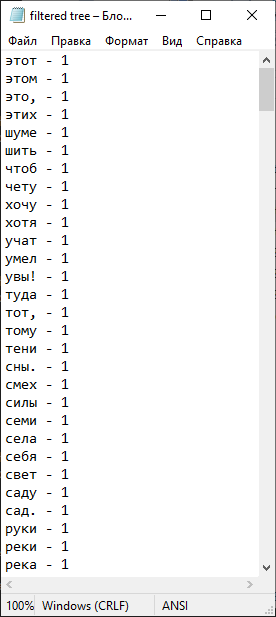
return 0;

}









Алгоритм работы программы

1. Открывается текстовый файл text.txt, содержащий исходные данные.
2. Слова из файла считываются и заносятся в бинарное дерево поиска (БДП).
3. Далее программа предлагает пользователю несколько функций:
   * Вывести список слов в алфавитном порядке.
   * Вывести список слов, отсортированных по частоте встречаемости.
   * Найти определённое слово.
   * Отфильтровать слова по длине.
4. В зависимости от выбора пользователя производится соответствующая обработка данных.
5. После завершения работы дерево удаляется для освобождения памяти.

Структура данных

Программа использует бинарное дерево поиска (BST, Binary Search Tree). Узел дерева (struct node) содержит:

* word — само слово.
* k — количество его вхождений в текст.
* left и right — указатели на левый и правый дочерние узлы.

Добавление слова в дерево (add)

* Если узел пуст, создаётся новый.
* Если слово меньше текущего узла — добавляется в левое поддерево.
* Если больше — в правое.
* Если слово уже есть, увеличивается счётчик k.

Функционал программы

1. Вывод слов в алфавитном порядке (printTree)

* Рекурсивный обход бинарного дерева (левый узел → корень → правый узел).
* Результат сохраняется в abc sorted.txt.

2. Вывод слов по частоте встречаемости (printSortedByFrequency)

* Дерево конвертируется в вектор (treeToVector).
* Вектор сортируется по убыванию частоты.
* Результат сохраняется в freq sorted.txt.

3. Поиск слова (search)

* Поиск в бинарном дереве (идём влево или вправо в зависимости от значения).
* Если слово найдено, выводится вместе с количеством вхождений.

4. Фильтрация по длине слова (filter)

* Все слова заносятся в вектор.
* Оставляются только слова заданной длины.
* Создаётся новое дерево с отфильтрованными словами.
* Результат сохраняется в filtered tree.txt.

Меню программы

Пользователь взаимодействует через консольное меню:

1. Вывести слова в алфавитном порядке.
2. Вывести слова по частоте.
3. Найти слово.
4. Отфильтровать слова по длине.
5. Выход.

При завершении работы память освобождается с помощью deleteTree.