МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Череповецкий государственный университет»

**Лабораторная работа № 3**

**«Дерево поиска. Частотный словарь»**

**Выполнил:**

студент гр. 1ИВТпб-01-31оп

Климов А.Г.  
**Проверил:** преподаватель

Пышницкий К.М.  
Отметка о зачете:

Череповец

2018 год

**Задание:**

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.

**Двоичные деревья** – это структуры данных, состоящие из узлов, которые хранят значение, а также ссылку на свою левую и правую ветвь. Каждая ветвь, в свою очередь, является деревом. Узел, который находится в самой вершине дерева принято называть корнем (root), узлы, находящиеся в самом низу дерева и не имеющие потомков называют листьями (leaves). Ветви узла называют потомками (descendants). По отношению к своим потомкам узел является родителем (parent) или предком (ancestor). Также, развивая аналогию, имеются сестринские узлы (siblings – родные братья или сёстры) – узлы с общим родителем. Аналогично, у узла могут быть дяди (uncle nodes) и дедушки и бабушки (grandparent nodes). Такие названия помогают понимать различные алгоритмы.

**Двоичное дерево поиска** (ДДП) – это несбалансированное двоичное дерево, в котором элементы БОЛЬШЕ корневого размещаются справа, а элементы, которые МЕНЬШЕ размещаются слева.

Такое размещение – слева меньше, справа больше – не обязательно, можно располагать элементы, которые меньше, справа. Отношение БОЛЬШЕ и МЕНЬШЕ – это не обязательно естественная сортировка по величине, это некоторая бинарная операция, которая позволяет разбить элементы на две группы.

Для реализации бинарного дерева поиска будем использовать структуру Node, которая содержит значение и ссылку на правое и левое поддерево.

**Быстрая сортировка:**

1. Выбираем опорный элемент;

2. Разбиваем массив на 3 части;

* Создаём переменные l и r — индексы соответственно начала и конца рассматриваемого подмассива;
* Увеличиваем l, пока l-й элемент меньше опорного;
* Уменьшаем r, пока r-й элемент больше опорного;
* Если l всё ещё меньше r, то меняем l-й и r-й элементы местами, инкрементируем l и декрементируем r;
* Если l вдруг становится больше r, то прерываем цикл;

3. Повторяем рекурсивно, пока не дойдём до массива из 1 элемента.

**Поиск элемента:**

Для поиска элемента в бинарном дереве поиска можно воспользоваться следующей функцией, которая принимает в качестве параметров корень дерева и искомый ключ. Для каждого узла функция сравнивает значение его ключа с искомым ключом. Если ключи одинаковы, то функция возвращает текущий узел, в противном случае функция вызывается рекурсивно для левого или правого поддерева. Узлы, которые посещает функция образуют нисходящий путь от корня, так что время ее работы O(h), где h — высота дерева.

**Текст программы:**

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <string>

#include <windows.h>

struct node;

void FormTree(node \*&p, std::string word);

void PrintTree(node \*&p, std::string &s);

void Sort(int \*a, std::string \*word, int first, int last);

node\* Search(node\* p, std::string word);

int count = 0; //Счётчик слов

struct node {

std::string word;

int k;

node \*left, \*right;

};

void FormTree(node \*&p, std::string word) {

if (p == NULL) {

p = new node;

p->word = word; p->k = 1;

p->left = NULL;

p->right = NULL;

}

else if (word < p->word) FormTree(p->left, word);

else if (word > p->word) FormTree(p->right, word);

else { p->k++; count--; }

}

void PrintTree(node \*&p, std::string &s) {

char c[6]; //Строка, в которую преобразуется число

if (p != NULL) {

PrintTree(p->left, s);

s += p->word + " - " + itoa(p->k, c, 10) + "\n";

PrintTree(p->right, s);

}

};

void Sort(int \*a, std::string \*word, int first, int last)

{

int i = first, j = last, key = a[(first + last) / 2];

do {

while (a[i]<key) i++;

while (a[j]>key) j--;

if (i <= j) {

std::swap(a[i], a[j]); swap(word[i], word[j]);

i++; j--;

}

} while (i <= j);

if (i<last) Sort(a, word, i, last);

if (j>first) Sort(a, word, first, j);

}

node\* Search(node \*p, std::string word1) {

if (p == NULL) return NULL;

if (p->word == word1) return p;

if (word1 < p->word) return Search(p->left, word1);

else

return Search(p->right, word1);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int word\_max\_length; //Максимальная длина слова

std::cout << "Оставить слова длина которых больше: ";

std::cin >> word\_max\_length;

std::string stringWord; //Слово

char c; //Буква

std::ifstream f\_input; //Файловый ввод

node\* p = NULL;

f\_input.open("input1.txt");

while (!f\_input.eof())

{

stringWord = "";

while (!f\_input.eof())

{

c = f\_input.get();

if (f\_input.eof()) break;

if (c >= -64 && c <= -1) stringWord += c; //65-90 97-122 eng

else break;

}

if (stringWord.length()>word\_max\_length)

{

FormTree(p, stringWord); count++;

}

}

f\_input.close();

std::ofstream f1("output.txt"); //Файловый вывод

std::string stringWord2;

PrintTree(p, stringWord2);

f1 << stringWord2;

f1.close();

system("notepad.exe output.txt");

std::ifstream f("output.txt");

std::string\* strMas = new std::string[count + 1];

int\* freqMas = new int[count + 1];

for (int i = 0; i<count + 1; i++)

{

if (f >> strMas[i] >> c >> freqMas[i]);

}

f.close();

Sort(freqMas, strMas, 0, count);

std::ofstream f3("output1.txt");

for (int j = 1; j<count + 1; j++)

{

f3 << strMas[j] << " - " << freqMas[j] << std::endl;

}

system("notepad.exe output1.txt");

std::string key = "сказал";

std::cout << "Введите слово для поиска: ";

std::cin >> key;

node\* p3 = Search(p, key);

if (p3 == NULL)

std::cout << "Элемент не найден";

else

std::cout << "Ваше слово найдено: " << p3->word;

std::cout << std::endl;

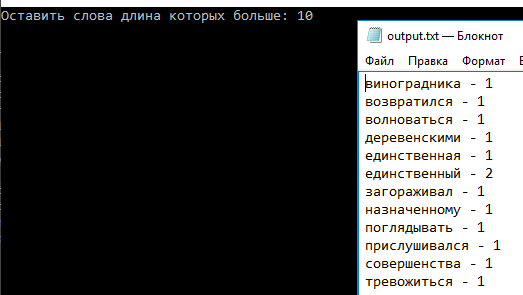
system("pause");

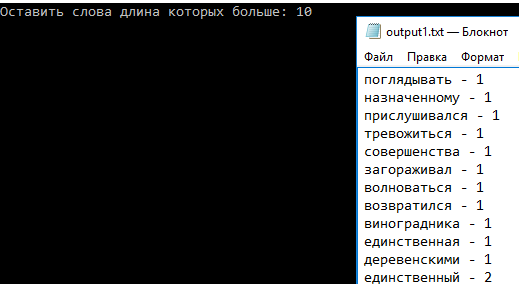
return 0;

}

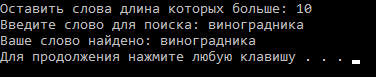
**Результаты тестирования:**

Частотный словарь слов в алфавитном порядке, содержащий слова определённой длины:

****

Частотный словарь слов, отсортированный в порядке частоты их употребления:****

Поиск заданного слова:

****