**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»**

Факультет физико-математических и естественных наук

Кафедра информационных технологий

«УТВЕРЖДАЮ»  
Заведующий кафедрой  
информационных технологий  
д.ф.-м.н., проф.

\_\_\_\_\_\_\_\_\_\_\_\_Ю.Н. Орлов   
« » 2020г.

**ОТЧЕТ  
по лабораторной работе**

темА «Тема лабораторной работы»  
по дисциплине «Компьютерный практикум по информационным технологиям»

Выполнил:

Студент группы Н?Ибд-01-19

Студенческий билет № 1032189354

Ласурия Данил Рустанбеевич

(Подпись)

« 18 » июня 2020г.

Руководитель:

к.ф.-м.н., доцент кафедры информационных технологий

Виноградов Андрей Николаевич

(Подпись)

« » 2020г.

Оценка:

Москва 2020

**Цель работы**

Реализовать алгоритмы поисковых систем среди базы документов

**Код программы**

Reading.h

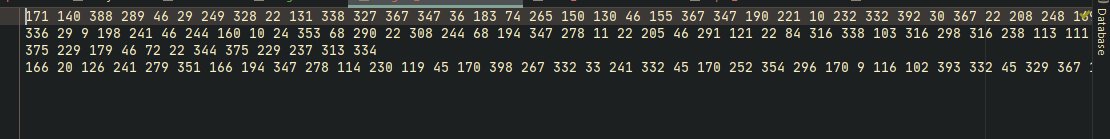
*//  
// Created by user on 08.06.2020.  
//*#ifndef **SEARCHSYSTEM\_READING\_H**#define **SEARCHSYSTEM\_READING\_H**#include <sstream>  
#include <fstream>  
#include <string>  
#include <vector>  
#include <map>  
#include <iostream>  
#include <set>  
#include <algorithm>  
#include <windows.h>  
  
using namespace std;  
*/\*Класс для открытия файлов\*/*class Reading {  
public:  
 Reading();  
 */\*Открытие файла\*/* explicit Reading(const string& filename)  
 {  
 */\*Если файла нет, то кидаем ошибку\*/* NameBook = create\_name(filename);  
 book.open(filename);  
 if (!book)  
 throw invalid\_argument("Invalid Filename");  
 }  
 */\*Возвращаем поток\*/* ifstream& GetFile()  
 {  
 return book;  
 }  
 */\*Возвращаем имя файля\*/* string GetName() const  
 {  
 return NameBook;  
 }  
 */\*Чтение октрытого файла\*/* void Read ()  
 {  
 string buffer;  
 while (getline(book,buffer)) *//проходимся по каждой СТРОКЕ файла* cout<<buffer<<endl;  
 }  
 */\*Деструктор\*/* ~Reading()  
 {  
 book.close();  
 }  
 */\*Открть инной файл\*/* ifstream& OPEN (string filename)  
 {  
 NameBook = create\_name(filename);  
 book.open(filename);  
 if (!book) *//если файла нет, кидаем ошибку* throw invalid\_argument("Invalid Filename");  
 return book;  
 }  
  
  
  
private:  
 ifstream book; *//поток файла* string NameBook; *//название файла  
 /\*Отбрасываем .txt\*/* string create\_name(const string& filename)  
 {  
 return filename.substr(0, filename.rfind('.'));  
 }  
};  
*/\*Блок операторов для логических операций\*/*inline bool operator==(char &lhs, const set <char> &rhs)  
{  
 for (auto symbol : rhs){  
 if (lhs == symbol) {  
 return true;  
 }  
 }  
 return false;  
}  
inline bool operator!=(char &lhs, const set <char> &rhs)  
{  
 for (auto symbol : rhs){  
 if (lhs != symbol) {  
 return true;  
 }  
 }  
 return false;  
}  
  
inline bool operator != (const vector <string>& lhs,const string &rhs )  
{  
 for (auto const &l:lhs)  
 {  
 if (l == rhs)  
 return false;  
 }  
 return true;  
}  
*/\*Конец блока\*/  
  
/\*Парсинг файла\*/*class ParseBook : private Reading{  
public:  
 ParseBook (const string& filename) : Reading(filename)  
 {  
 Reading reading(filename);  
 start(reading);*//запускаем парсинг* }  
 */\*Функция для парсинга нового файла в области одного и того же объекта\*/* void new\_book(string filename)  
 {  
 Reading reading(filename);  
 start(reading);  
 }  
 */\*Функция парсинга\*/* void start(Reading &reading)  
 {  
 while (!reading.GetFile().eof()) *//пока не конец файла, читаем* {  
 reading.GetFile().get(letter); *// берем символ* if (letter == '\10' or letter == '\n') *// если он равен переходу на новую строку* {  
 if (!buffer.empty()) { *//если буффер не пустой* int temp = int(reading.GetFile().tellg())-1-buffer.size();*//позиция слова* list[buffer][reading.GetName()].push\_back(temp); *//закидываем в словарь слово (если его не было) [книга,  
 //закидываем в вектор еще позицию ]  
 //* if (line\_index != buffer )*//если слово не встречали закидываем его в прямой индекс* line\_index.push\_back(buffer);  
 }  
 buffer.resize(0); *//удаляем слово* }  
 else if (letter == symbols ) *//если встретили запретный символ* {  
 */\*тоже самое, что и выше\*/* if (!buffer.empty()) {  
 int temp = int(reading.GetFile().tellg())-1-buffer.size();  
 list[buffer][reading.GetName()].push\_back(temp);  
 if (line\_index != buffer )  
 line\_index.push\_back(buffer);  
 }  
 buffer.resize(0);  
 }  
 else *//иначе добавляем символ к нашему слову* {  
 buffer += tolower(letter);  
 }  
 }  
 sorting(); *//сортируем* }  
  
  
 */\*функция печати словаря\*/* void print ()  
 {  
 for (auto &[key, value] : list)  
 {  
 for (auto &[name, pos]: value) {  
 cout << "word: '" << key << "' book: '" << name << "' pos:[";  
 for (auto p:pos)  
 cout<<p<<", ";  
 cout<<"]";  
 cout<<endl;  
 }  
 }  
 }  
  
 map <string , map <string,vector<int>>>& GetMap () {return list;}*//функция возрата словаря  
 /\*Создание прямых индексов\*/* void book\_direct\_index( map <string , map <string,vector<int>>>& res)  
 {  
 for (auto const & [name, key] :res) { *//для каждой книги* ofstream output(name + "\_direct-index" + ".txt"); *//поток ввода в книгу* Reading r (name + ".txt"); *//октртия исходной книги* while (!r.GetFile().eof()) {  
 r.GetFile().get(letter); *//берем символ* if (letter == '\10' or letter == '\n') { *//если знак enter* if (!buffer.empty()) { *//если буфер не пустой* int \_temp = iter\_(buffer); *//получаем итератор(ключ) вектора с заданным словом* output << \_temp; *//вписываем этот итератор (ключ)* }  
 output << endl;*//и ставим знак eneter* buffer.resize(0); *//удаляем слово  
 /\*тоже самое, но только если есть запретный имвол\*/* } else if (letter == symbols) {  
 if (!buffer.empty()) {  
 int \_temp = iter\_(buffer);  
 output << \_temp << " ";  
 }  
 buffer.resize(0);  
 } else {  
 buffer += tolower(letter);*//доавбляем символ, формируя слово* }  
 }  
 output.close(); *//закрываем файл* }  
  
 }  
  
 */\*Функция читаем готовый словарь\*/* map <string, map<string,vector<int>>>& ReadMap ()  
 {  
 string temp;  
 while(getline(GetFile(),temp)) *//читаем строку* {  
 stringstream buffer(temp);  
 string trash,name,word;  
 vector <int> pos;  
 buffer>>trash; *//читаем пробел* buffer>>word; *//читаем слово* buffer>>trash;*// пробел* buffer>>name; *//книга* buffer>>trash; *//пробел  
 /\*читаем позиции\*/* while(!buffer.eof())  
 {  
 buffer>>trash;  
 if (trash!="]")  
 pos.push\_back(stoi(trash));  
 }  
 list[word][name]=pos; *//добавляем прочитанные позиции* }  
 return list; *//возвращаем словарь* }  
  
 vector<string> GetVec ()  
 {  
 return line\_index; *//возвращем вектор прямых индексов* }  
 ~ParseBook()  
 {}  
  
private:  
 string buffer; *//буфер слова* char letter; *//символ* set <char> symbols = {',', '!', '?', ')', '(', '/', '.', ':', ';', '"', '`', ' '}; *//запретные символы* map <string , map <string,vector<int>>> list; *//словарь* vector <string> line\_index; *//прямые индексы* void sorting()  
 {  
 sort(line\_index.begin(), line\_index.end()); *//функцяи сортировки* }  
  
 int iter\_(string word)  
 {  
 return (find(line\_index.begin(), line\_index.end(), word)-line\_index.begin()); *//возвращает индекс найденного эелмента* }  
};  
  
  
#endif *//SEARCHSYSTEM\_READING\_H*

Reading.cpp

*//  
// Created by user on 08.06.2020.  
//*#include "Reading.h"

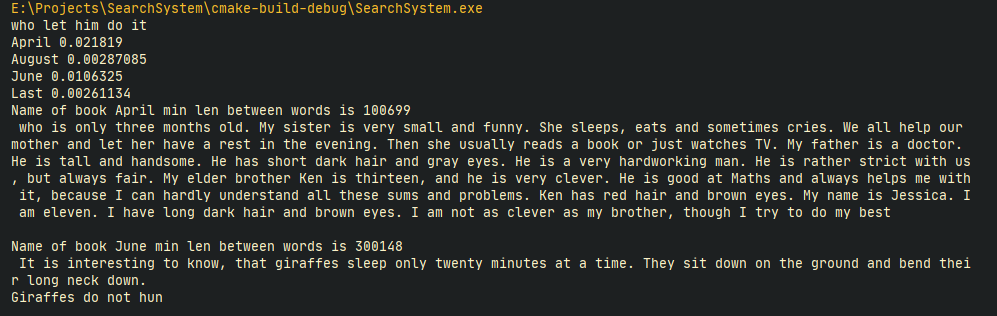
main.cpp

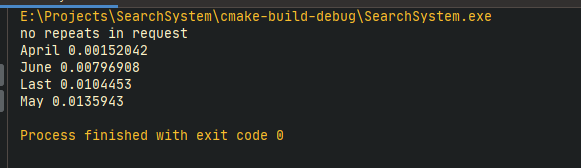
#include "Reading.h"  
#include <cmath>  
  
*/\*Класс работы со вловарем\*/*class DataBase {  
public:  
 */\*конструктор\*/* DataBase(string filename,  
 const map<string, map <string, vector<int>>> &list)  
 {  
 db.open(filename); *// октрываем файл на запись* work\_file = filename; *//даем ему имя* create\_bookword(list); *//создлаем "обратный словарь"* SetBookNames(list); *//вектор всех книг* count\_word\_in\_book(list); *//сколько слов в тексте* }  
 */\*запись словармя в текст\*/* void WriteDB\_inF(string mode, const map<string, map <string, vector<int>>> &list )  
 {  
 if (mode == "word-book") { *//если хотии вид книга-слово* for (auto const &[word, value] : list) {  
 for (auto const &[name, poses] : value) {  
 bool flag = true;  
 db << "'word': " << word << " 'book': " << name << " 'pos':[ " ;  
 for (auto const &pos : poses) {  
 if (flag)  
 db << pos;  
 else  
 db << ", " << pos;  
 flag = false;  
 }  
 db << " ]" << endl;  
 }  
 }  
 }  
 else if(mode == "book-word") *//вид слово-книга* {  
 for (auto const &[word, value] : list) {  
 for (auto const &[name, poses] : value) {  
 bool flag = true;  
 db << "book: " << word << " word: " << name << " pos:[ ";  
 for (auto const &pos : poses) {  
 if (flag)  
 db << pos;  
 else  
 db << ", " << pos;  
 flag = false;  
 }  
 db << "]" << endl;  
 }  
 }  
 }  
 }  
 */\*Тоже самое, только запись в консоль\*/* void WriteDB (string mode, const map<string,  
 const map <string, vector<int>>> &list)  
 {  
 if (mode == "book-word") {  
 for (auto const &[word, value] : bookword) {  
 for (auto const &[name, poses] : value) {  
 bool flag = true;  
 cout << "book: '" << word << "' word: '" << name << "' pos:[";  
 for (auto const &pos : poses) {  
 if (flag)  
 cout << pos;  
 else  
 cout << ", " << pos;  
 flag = false;  
 }  
 cout << "]" << endl;  
 }  
 }  
 }  
 else if(mode=="word-book")  
 {  
 for (auto const &[word, value] : list) {  
 for (auto const &[name, poses] : value) {  
 bool flag = true;  
 cout << "word: '" << word << "' book: '" << name << "' pos:[";  
 for (auto const &pos : poses) {  
 if (flag)  
 cout << pos;  
 else  
 cout << ", " << pos;  
 flag = false;  
 }  
 cout << "]" << endl;  
 }  
 }  
 }  
 }  
  
 */\*пишем имен книг\*/* void GetBookNames ()  
 {  
 cout<<"Size of vector: "<<BookNames.size()<<"\nNames of books:"<<endl;  
 for (auto const& name:BookNames)  
 {  
 cout<<name<<endl;  
 }  
 }  
 */\*закртыие файла\*/* void CLOSE ()  
 {  
 db.close();  
 cout<<work\_file<<" closed";  
 }  
 */\*если нужно дописать файл\*/* void Add (string filename)  
 {  
 db.open(filename, ios\_base::app);  
 if (!db)  
 throw invalid\_argument("Invalid Filename");  
 work\_file = filename;  
 }  
 */\*узнать состояние файла , октрыт или нет\*/* bool GetState()  
 {  
 if(db)  
 return true;  
 return false;  
 }  
 */\*создание веса TF-IDF\*/* void make\_weight (const vector <string>& request,  
 const map<string, map <string, vector<int>>> &list)  
 {  
 map <string, vector<double>> sumTI; *//вспомогательный словарь* calc\_weight(request, list, sumTI); *//считаем вес* for (auto const&[book, value]:sumTI) *//складывает промежуточные веса* {  
 double temp = 0;  
 for(auto const &v:value)  
 {  
 temp+=v;  
 }  
 weight[book]=temp;  
 }  
 }  
 */\*вывести кол-во слов\*/* void get\_count()  
 {  
 for (auto const &[name,count] : word\_count)  
 cout<<name<<" " <<count<<endl;  
 }  
 */\*печать веса\*/* void print\_weights()  
 {  
 *//cout<<weight.size();* for (auto const &[name, w]:weight)  
 {  
 cout<< name << " "<< w<<endl;  
 }  
 }  
  
 void length (const vector <string> &request)  
 {  
 map <int, string> min\_dist; *//контейрер книга-мин\_дист* for (auto const& name: BookNames) { *// по каждой книге* vector<string> real\_word; *//слова которые есть в книге* for (auto const &r: request) { *//создаем вектор для работы со словами которые есть в тексте* for (auto const&[word, pos]: bookword[name]) {  
 if (r == word)  
 {  
 real\_word.push\_back(r);*//если есть слово в нкиге то добавляем его* }  
 }  
 }  
 sort(real\_word.begin(), real\_word.end()); *//сортируем наш вектор, т.к map по умолчанию сортируется;* map <string, vector<int>> temp;  
 for (auto const &r: real\_word)  
 {  
 for(auto const&[word, pos]: bookword[name])  
 {  
 if (r == word)  
 temp[word] = pos; *//создаем словарь из тех слов которые реально есть* }  
 }  
  
 int total = 1; *//кол-во комбинаций* int distance = 0; *//конечная дистанция* int temp\_distance = 0; *//рабочая дистанция* vector <int> cord\_temp; *//кол-во позиций* vector <int> cord; *//вспомогательный* for (auto const &[word, pos] : temp)  
 {  
 total \*= pos.size(); *//считаем кол-во комбинаций* cord\_temp.push\_back(pos.size()); *//закидываем кол-во позиций для каждого слова* cord.push\_back(pos.size());  
 }  
 vector <int> temp\_pos;  
 if (real\_word.size() > 1) { *//если в книге больше двух слов* for (int k = 0; k < total; k++) { *//прозодимся по каждому возможному случаю* if (k == 0) { */\*задаем начальную длину\*/* for (int j = 0; j < real\_word.size(); j++) *//считаем длину между словами* {  
 temp\_pos.push\_back(temp[real\_word[j]][cord\_temp[j] - 1]); *//позции слова* }  
  
 for (int j = 0; j < real\_word.size() - 1; j++) {  
 distance += abs(temp[real\_word[j]][cord\_temp[j] - 1]  
 - temp[real\_word[j + 1]][cord\_temp[j + 1] - 1]); *//считаем позицию между 1 и 2 словом* temp\_distance = distance; *//задаем рабочую длину* }  
 } else {  
  
 for (int j = 0; j < real\_word.size() - 1; j++) {  
 temp\_distance += abs(temp[real\_word[j]][cord\_temp[j] - 1]  
 - temp[real\_word[j + 1]][cord\_temp[j + 1] - 1]);*//считаем позицию между 1 и 2 словом* }  
 }  
  
 if (temp\_distance < distance) { *//если новая дистанция меньше* distance = temp\_distance; *//меняем мин длину* temp\_pos.resize(0); *//удаляем пред набор мин позиций* for (int j = 0; j < real\_word.size(); j++) { *//задаем заново набор мин позиций* temp\_pos.push\_back(temp[real\_word[j]][cord\_temp[j] - 1]);  
 }  
 }  
 temp\_distance = 0; *//обнуялем рабочую дилну  
 /\*Основа алогритма, переход к следю случаю комбинации\*/* cord\_temp[cord\_temp.size() - 1]--;  
 for (int i = cord\_temp.size() - 1; i > 0; --i) {  
 if (cord\_temp[i] <= 0) { *//обработав все возможные комбинации самой вложенной позиции, возвращем ее, уменшая менее вложенные* cord\_temp[i] = cord[i];  
 cord\_temp[i - 1]--;  
 }  
 }  
 }  
  
 if (real\_word.size() < request.size() and distance != 0) {  
  
 distance += 100000 \* (request.size() - real\_word.size()); *//если слово не встретили в тексте* }  
 snip[name] = temp\_pos; *//закидываем в словарь (книга - позиции) позции для книги* min\_dist[distance] = name; *//закидываем мин длину для этой книги* }  
 }  
 snipet(snip,min\_dist); *//вывваем функцию для снипета* }  
  
 void snipet (map <string, vector <int>>& snip, map <int, string> &min\_dist)  
 {  
 for (auto const& [len, name] : min\_dist)  
 {  
 if (len > 0) {  
 cout << "Name of book " << name << " min len between words is " << len << endl; *//выводим данные* Reading reading(name + ".txt");  
 char c;  
 reading.GetFile().seekg(\*min\_element(snip[name].begin(), snip[name].end())-1);*//задем положение каретки чтения самой минимальной позицией  
 /\*Пока каретка не дойдет до послденго слова из запроса\* будем читать книгу\*/* while (reading.GetFile().tellg() < \*max\_element(snip[name].begin(), snip[name].end()) + 10) *//т.к изначально не учитывалась длинна, добалвем еще 10 символов* {  
 reading.GetFile().get(c);  
 cout<<c;  
 }  
 reading.GetFile().close();  
 }  
 cout<<endl<<endl;  
 }  
 }  
  
 map <string, map <string, vector<int>>> & Getbookword()  
 {  
 return bookword; *//возвращает словарь в инном виде* }  
  
private:  
 ofstream db; *//поток раб файла* string work\_file; *//имя работника* vector <string> BookNames; *//все книги* map <string, double> TF; *//индексы TF* map <string, int> word\_count; *//все слова для книги* map <string, double> weight; *//веса для книги* map <string, map <string, vector<int>>> bookword; *//словарь в инном виде* map <string, vector <int>> snip; *//длины снипета для книги  
 /\*функци устанавливаем названия книг\*/* void SetBookNames(const map<string, map <string, vector<int>>> &list)  
 {  
 bool flag = true;  
 for (auto const&[word, value]: list)  
 {  
 for (auto const &[NameBook, p]: value) {  
 if (flag) {  
 BookNames.push\_back(NameBook);*//кидаем первую книгу* flag = false;  
 }  
 if (BookNames != NameBook) { *//если такой книги не блыо, то кидаем ее* BookNames.push\_back(NameBook);  
 }  
 }  
 }  
 }  
 */\*Подсчет слов в книге\*/* void count\_word\_in\_book(const map<string, map <string, vector<int>>> &list)  
 {  
 for (auto const &name : BookNames) *//для каждой ниги* {  
 for (auto const&[word, value]: list) *//для пары из словаря [слово, пара]* {  
 for(auto const& [book, pos]:value) *//для пары из [книга, вектор поз]* {  
 if(book == name)  
 {  
 word\_count[book]+=pos.size(); *//кол-во позиций и есть кол-во опрд слова* }  
 }  
 }  
 }  
 }  
  
 void create\_bookword(const map<string, map <string, vector<int>>> &list)  
 {  
 bookword=new\_map(list); *//осздаем инной словарь* }  
  
 void make\_sum(double temp, map <string, vector<double>> &sumTI)  
 {  
 for (auto const&[name, value]:TF)  
 {  
 sumTI[name].push\_back(value\*temp); *//закидываем в сумму TF\*IDF для каждой книги* }  
 }  
  
 void make\_TF (const map <string, int> &temp, const string &request)  
 {  
 for (auto const&[key, value]:temp)  
 {  
 TF[key] = 1.\*value / word\_count[key]; *//для книги считаем TF* }  
 }  
  
  
 double calc\_IDF(string request,  
 const map<string, map <string, vector<int>>> &list)  
 {  
 int counter = 0;  
 for(auto const&[word, value]: list) {  
 if (word == request)  
 counter = value.size(); *//считаем скольк раз встретили слово во всех книгах* }  
 return log10((1.\*BookNames.size()/counter)+exp(0.5)); *//возвращаем IDF* }  
  
 void calc\_weight(const vector <string>& request,  
 const map<string, map <string, vector<int>>> &list,  
 map <string, vector<double>> &sumTI)  
 {  
 map <string , int> tf;  
 for(auto const& r: request)  
 {  
 for (const auto &[word,value]:list)  
 {  
 double \_temp = 0;  
 if (word == r) {  
 for (const auto &[book, p]:value) {  
 tf[book] += p.size(); *//сколько слов которые нам подходят из запроса для опр книги* }  
 \_temp=calc\_IDF(r,list); *//считаем idf* make\_TF(tf, r); *//считаем tf* make\_sum(\_temp, sumTI); *//умножаем* }  
 }  
 }  
 }  
 */\*создание инного словаря\*/* map <string, map <string, vector<int>>> new\_map (const map<string, map <string, vector<int>>> &list)  
 {  
 */\*Меняем местами слово и книгу\*/* map <string, map <string, vector<int>>> temp;  
 for(auto const&[word,value]:list)  
 {  
 for (auto const&[book,pos]:value)  
 {  
 temp[book][word]=pos;  
 }  
 }  
 return temp;  
 }  
  
};  
  
void parse\_request(vector<string> &request)  
{  
 string str;  
 string temp;  
 getline(cin,str);*//считаем строку* stringstream buffer(str); *//превращяем ее в поток* while(!buffer.eof()) *//пока не конец потока* {  
 buffer>>temp; *//читаем по словам* if (request != temp) *//если слова не было до этого* request.push\_back(temp); *//закдываем это слово* }  
}  
  
int main() {  
  
 SetConsoleOutputCP(65001); *//язык работы консоли* vector <string> request;  
 parse\_request(request);*//функция создания запроса* ParseBook book("August.txt");*// первая книга* book.new\_book("April.txt"); *//вторая книга* book.new\_book("May.txt"); *//тертья кнгиа* book.new\_book("June.txt"); *//четвертая кнгиа* book.new\_book("Last.txt"); *//пятая книга* DataBase db("Retail.txt", book.GetMap()); *//создлаем обхет для работы со словарем* book.book\_direct\_index(db.Getbookword());*//создаем файлы прямых индексов* db.WriteDB\_inF("word-book", book.GetMap()); *//записываем в файл в виде слово - книга* db.make\_weight(request, book.GetMap()); *//считаем вес* db.print\_weights(); *//печатаем вес* db.length(request); *//подсчет длины и вывод снипета* return 0;  
}



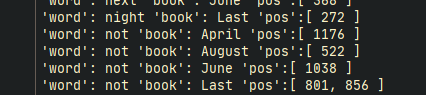
Прямые индексы

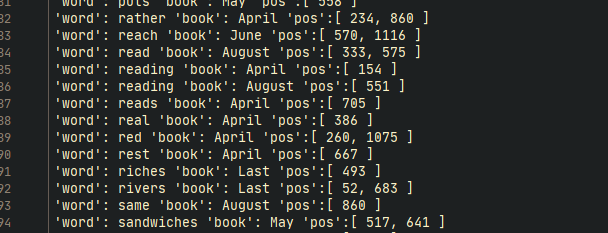






Из запроса только одно слово есть тексте, поэтому сниппета и длинны для них нет





Программа работает только с английским текстом, а точнее с ASCII. Т.к в с++ char ограничен размером.

Все исходные файлы и словари будут прикреплены к письму

Как можно заметить, оценка релевантности методом TF-IDF крайне близка к методу поиска мин. длины. Но при этом мин. длинна выдаст файл, который будет более похож, так он учитывает порядок слов.

**Вывод**

были реализованы алгоритмы поисковых систем