**Московский авиационный институт**

**(национальный исследовательский университет)**

**Институт информационные технологии и прикладной математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №4 по курсу**

**«Информационный поиск»**

|  |  |
| --- | --- |
| Студент: | Е.М. Стифеев |
| Преподаватель: | А.А. Кухтичев |
| Группа: | М8О-109М-21 |
| Дата: | 10.11.21 |
| Оценка: |  |
| Подпись: |  |

**Москва, 2021**

# Лабораторная работа №4 «Булев поиск»

Нужно реализовать ввод поисковых запросов и их выполнение над индексом, получение поисковой выдачи.

Синтаксис поисковых запросов:

* Пробел или два амперсанда, «&&», соответствуют логической операции «И».
* Две вертикальных «палочки», «||» – логическая операция «ИЛИ»
* Восклицательный знак, «!» – логическая операция «НЕТ»
* Могут использоваться скобки.

Парсер поисковых запросов должен быть устойчив к переменному числу пробелов, максимально толерантен к введённому поисковому запросу.

Примеры запросов:

* [ московский авиационный институт ];
* [ (красный || желтый) автомобиль ];
* [ руки !ноги ].

Для демонстрации работы поисковой системы должен быть реализован веб-сервис, реализующий базовую функциональность поиска из двух страниц:

* Начальная страница с формой ввода поискового запроса.
* Страница поисковой выдачи, содержащая в себе форму ввода поискового запроса, 50 результатов поиска в виде текстов заголовков документов и ссылок на эти документы, а так же ссылку на получение следующих 50 результатов.

Так же должна быть реализована утилита командной строки, загружающая индекс и выполняющая поиск по нему для каждого запроса на отдельной строчке входного файла.

В отчёте должно быть отмечено:

* Скорость выполнения поисковых запросов.
* Примеры сложных поисковых запросов, вызывающих длительную работу.
* Каким образом тестировалась корректность поисковой выдачи.

# Описание

## Корпус и индекс

Напомню, что корпус документов имеет следующую структуру, полученную по результатам ЛР1 (доступен по ссылке <https://cloud.mail.ru/public/ZfkX/gccM7hnDR> ):

* Корпус документов
  + films1.txt (94 Мб, 15000 документов)
  + films2.txt (96 Мб, 15000 документов)
  + films3.txt (184 Мб, 15000 документов)
  + films4.txt (219 Мб, 15000 документов)
  + films5.txt (322 Мб, 15000 документов)
  + films6.txt (711 Мб, 15000 документов)
  + films7.txt (823 Мб, 15000 документов)
  + films8.txt (226 Мб, 15000 документов)
  + films9.txt (67 Мб, 15000 документов)
  + films10.txt (75 Мб, 15000 документов)
  + films11.txt (99 Мб, 15000 документов)
  + films12.txt (78 Мб, 15000 документов)
  + films13.txt (41 Мб, 6109 документов)

Также, напомню, что получение одного документа могло включать проход по нескольким html-страницам и обработку динамически подгружаемых страниц, поэтому общее количество обработанных страниц было >800’000.

В каждом файле \*.txt документы хранятся следующим образом:

* 1 строка 1 документ {….}
* 2 строка 2 документ {….}
* *n* строка *n* документ {….}

Каждый документ снабжён прямой ссылкой на источник, откуда был скачен, и хранит только выделенный из html-кода текст в кодировке UTF-8. Например, 234 строка файла films1.txt выглядит так:

{"page\_url": "https://www.kinopoisk.ru/media/article/1773537/", "title": "Артур Смольянинов: «Я сомневался, что смогу сыграть ангела»", "body": "2 января в российский прокат вышла романтическая комедия Веры Сторожевой „Мой парень — ангел“, главные роли в которой исполнили Артур Смольянинов и Анна Старшенбаум. Мы подготовили небольшой видеосюжет с участием создателей картины...Студентка Саша с большим трудом верит в чудеса. Ангелу Серафиму приходится приложить немало усилий, чтобы доказать ей, что ангелы существуют. Но он не учел одного: если девушка тебе поверит, она, скорее всего, тебя полюбит.\n\n\n\n\n\n\n\nАвтор: Дарико Цулая", "comments": ""}.

Готовый индекс хранится в трёх файлах:

* **docs\_id.data** (35 Мб)

Файл служит для отображения индекса документа (doc\_id) в его текстовое представление в файлах \*txt. Поддерживается переменная длина пути до файлов с документами.

* **terms.data** (81 Мб)

Файл служит для хранения словаря с терминами и ссылок на файл с словопозициями. Поддерживается переменная длина термина. Термины упорядочены в лексикографическом порядке.

* **postings\_list.data** (889 Мб)

Файл служит для хранения словопозиций и частот терминов в документе. Слопозиции упорядочены по возрастанию идентификаторов документов.

## Алгоритм поиска

Скажу сначала пару слов о парсинге запроса из линейной строчной формы в форму, подходящую для обработки. Для начала входящая строка трансформируется в инфиксный запрос в виде последовательности операторов и операндов:

(самый лучший фильм) || (интересный сериал || !документалка)

->

( самый & лучший & фильм ) | ( интересный & сериал | ! документалка )

Далее, преобразуем в постфиксный запрос с помощью стандартного алгоритма Дейксты:

самый лучший фильм & & интересный сериал & документалка ! | |

Полученное выражение можно вычислять с помощью стека.

Основные операции со словопозициями запрограммированы согласно [1].

Приведу код операции пересечения слопозиций:

void intersect\_postings(const vector<int> &p1, const vector<int> &p2,

vector<int> &answer)

{

int i=0, j=0;

answer.resize(0);

while (i < p1.size() && j < p2.size())

{

if (p1[i] == p2[j])

{

answer.push\_back(p1[i]);

i++;

j++;

}

else if (p1[i] < p2[j])

{

i++;

}

else

{

j++;

}

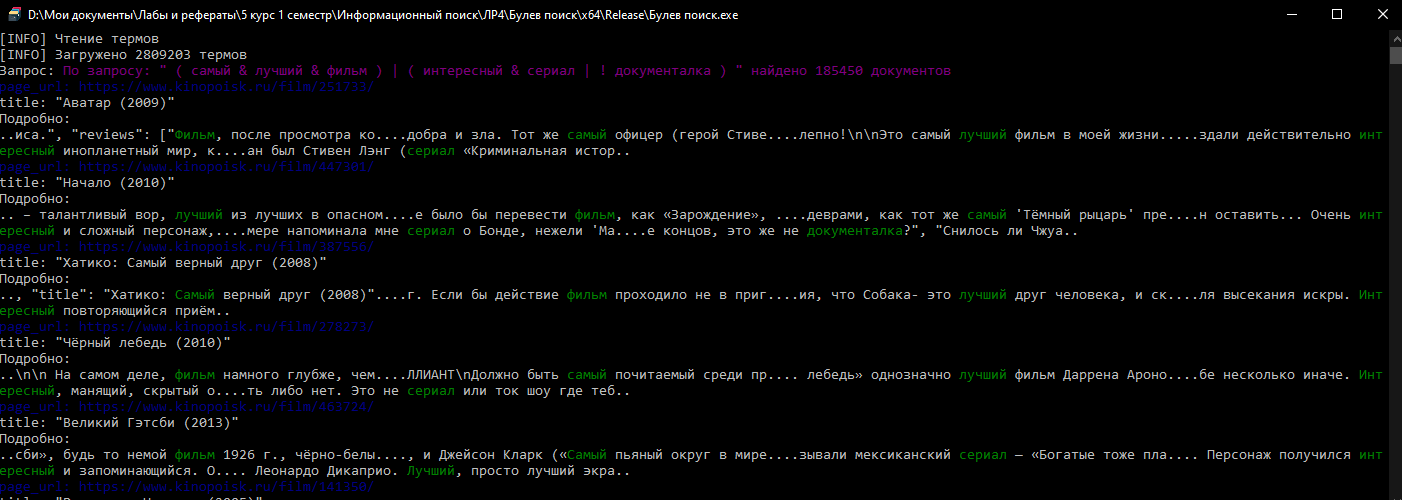
}

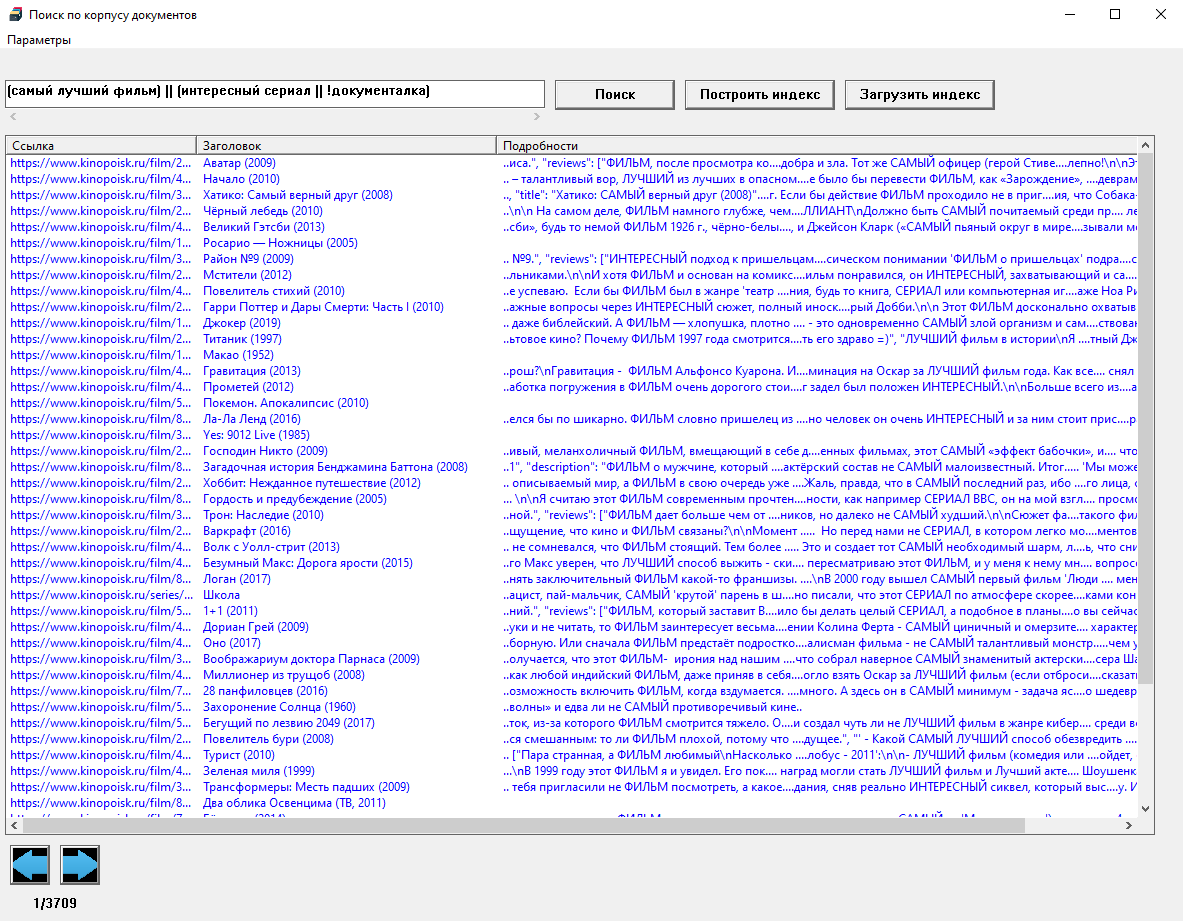
}

## Интерфейс

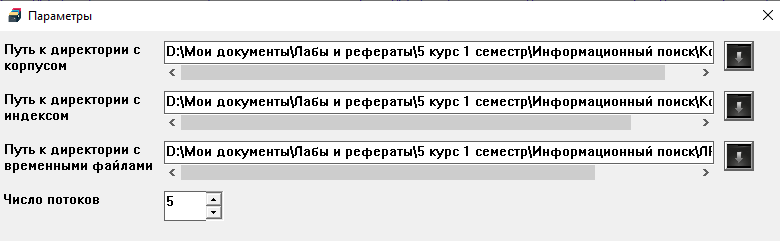
Программа была реализована в двух интерфейсах: консольном и оконном.

Для последнего использовалась библиотека Windows.h (winapi). Примеры ниже:





Замечу, что для вывода «подробностей» требуется повторный проход по документу, что замедляет поисковую выдачу. Чтобы немного сгладить ситуацию, процесс был распараллелен. Лучшим решением данной ситуации является применение координатного индекса.



Также был написан автотестировщик.

# Исходный код

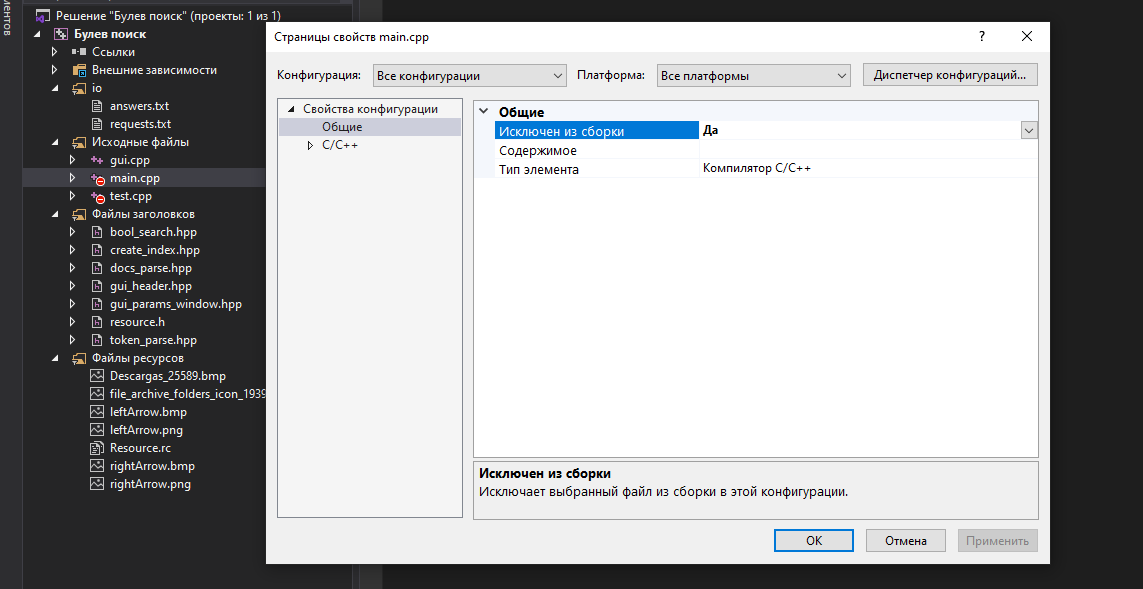
## Структура проекта

* include
  + bool\_search.hpp (булев поиск)
  + create\_index.hpp (создание, чтение индекса)
  + docs\_parse.hpp (извлечение полей из корпуса)
  + gui\_header.hpp (главный хэдер для оконного интерфейса)
  + gui\_params\_window.hpp (окно с выбором параметров)
  + token\_parse.hpp (функции для преобразования токенов в термы)
* io
  + anwers.txt
  + requests.txt
* src
  + gui.cpp (точка входа в оконный интерфейс)
  + main.cpp (точка входа в консольный интерфейс)
  + test.cpp (утилита для тестирования программы)
* resourses (файлы ресурсов для оконного приложения)

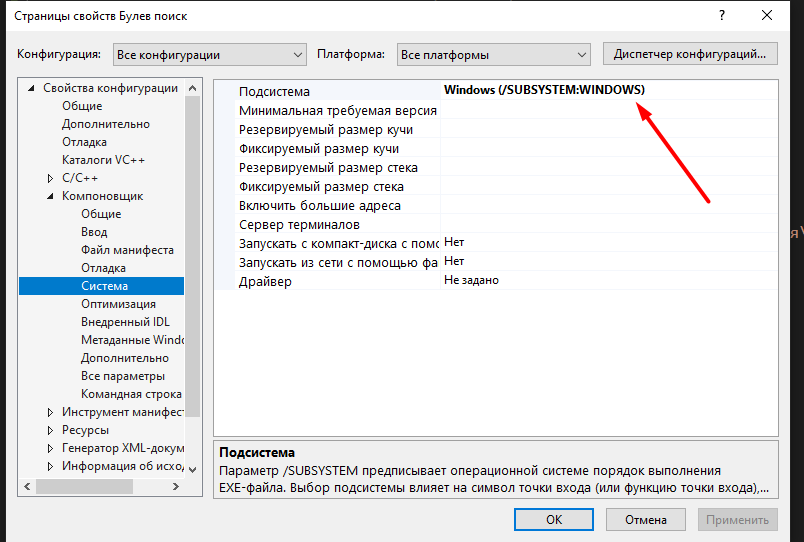
Проект был написан с помощью Microsoft Visual Studio 2019 эксклюзивно для ОС семейства Windows.

## Запуск

Переключение между тремя точками входа осуществляется с помощью флага «исключить из сборки»:



Не забудь при переключении между консольными и оконными приложениями менять подсистему в настройке проекта:



Консольное приложение поддерживает флаги запуска:

* -i 'абс. путь к корпусу'
* -o 'абс. путь к индексу'
* -t 'абс. путь к директории с блочным индексом'
* -p кол-во\_процессов\_для\_распараллеливания
* -create : создать блочный индекс
* -merge : выполнить слияние блочного индекса
* -clear : очистить папку с временными файлами после слияния
* -search : выполнить поиск

Утилита тестирования поддерживает следующие ключи:

* -i 'абс. путь к корпусу'
* -o \'абс. путь к индексу'
* -n1 число\_запросов
* -n2 длина\_запросы\_в\_термах

**Пример создания индекса из корпуса:**

$ ./Булев индекс.exe -merge -create -clear -i "D:\Мои документы\Лабы и рефераты\5 курс 1 семестр\Информационный поиск\Корпус" -o "D:\Мои документы\Лабы и рефераты\5 курс 1 семестр\Информационный поиск\Корпус\_index" -t "D:\Мои документы\Лабы и рефераты\5 курс 1 семестр\Информационный поиск\ЛР3\Булев индекс\tmp"

[INFO] Создание индекса для блоков

[INFO] Thread 0 processing block 1/13 : D:\Мои документы\Лабы и рефераты\5 курс 1 семестр\Информационный поиск\Корпус\films13.txt

[INFO] Thread 1 processing block 2/13 : D:\Мои документы\Лабы и рефераты\5 курс 1 семестр\Информационный поиск\Корпус\films9.txt

[INFO] Thread 2 processing block 3/13 : D:\Мои документы\Лабы и рефераты\5 курс 1 семестр\Информационный поиск\Корпус\films10.txt

[INFO] Thread 3 processing block 4/13 : D:\Мои документы\Лабы и рефераты\5 курс 1 семестр\Информационный поиск\Корпус\films12.txt

[INFO] Block 1 has 232216 terms

<…>

[INFO] Block 10 has 597482 terms

[INFO] Block 11 has 669497 terms

[INFO] Block 12 has 921883 terms

[INFO] Block 13 has 1383148 terms

[INFO] Создание очередей термов: 13 блок из 13

[INFO] Слияние docs\_id: 13 блок из 13

[INFO] Слияние слопозиций термов

[INFO] Осталось термов: 0

[INFO] Очистка временных файлов

[INFO] Общее число термов в словаре = 2809203

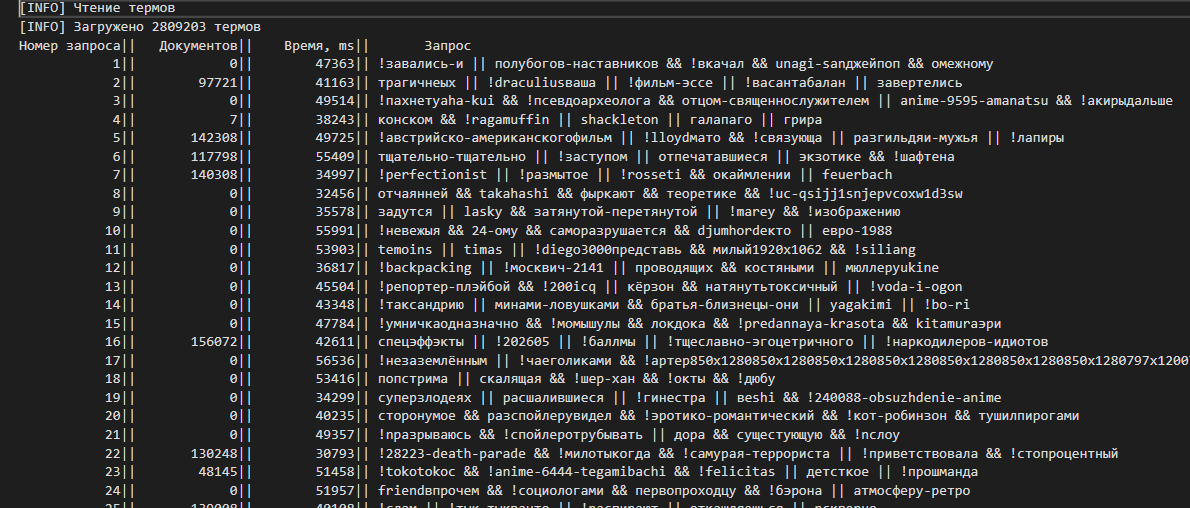
Время выполнения = 145,5 sec, размер корпуса = 2,899 Gb, документов = 186109

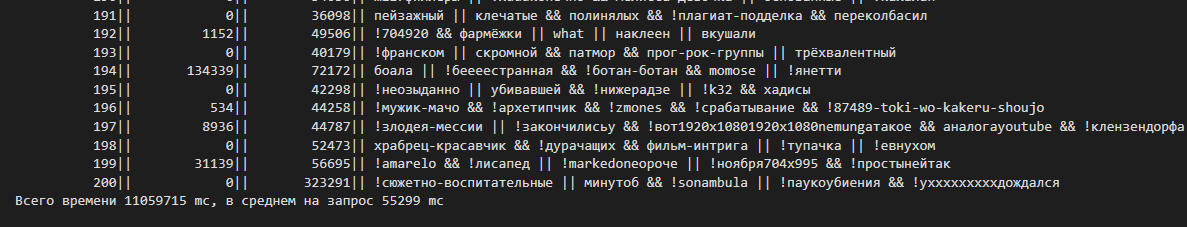
Средняя скорость на документ = 0,782 ms

Средняя скорость на килобайт = 0,048 ms

**Пример работы тестировщика:**

$ ./Булев индекс.exe -n1 200 -n2 5 -search -i "..\..\Корпус" -o "..\..\Корпус\_index" -t "tmp" <io/requests.txt >io/answers.txt





# Выводы

В целом процесс булевого поиска работает достаточно быстро. Самые тяжёлые запросы для системы выглядят следующим образом:

фильм && фильм && фильм ...

сериал && сериал && сериал ...

Для их обработки выполняется пересечение очень длинных одинаковых списков словопозиций. Возможное решение – символьная оптимизация запроса на уровне операторов. Также ещё раз отмечу, что для вывода блока с подробностями по запросу идёт повторный анализ документов с целью поиска термов. Это является узким местом производительности. Решения – распараллеливание (реализовано), координатный индекс.

В ходе выполнения лабораторной работы я научился выполнять булев поиск для коллекции документов. Познакомился с winapi. Научилcя разрабатывать оконный интерфейс на языке C/C++.

# Литература

1. Кристофер Д.Маннинг, Прабхакар Рагхаван, Хайнрих Шютце. Введение в информационный поиск. 2020, изд. Вильямс.