Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Информационный поиск»

Студент: Комбаров В. А.
Преподаватель: Кухтичев А. А.
Группа: М8О-412Б-22
Дата:
Оценка:
Подпись:

Москва, 2026

Содержание

Лабораторная работа №1 «Добыча корпуса документов»

Необходимо подготовить корпус документов, который будет использован при выполнении остальных лабораторных работ:

- Скачать его к себе на компьютер. В отчёте нужно указать источник данных.
- Ознакомиться с ним, изучить его характеристики. Из чего состоит текст? Есть ли дополнительная мета-информация? Если разметка текста, какая она?
- Разбить на документы.
- Выделить текст.
- Найти существующие поисковики, которые уже можно использовать для поиска по выбранному набору документов (встроенный поиск Википедии, поиск Google с использованием ограничений на URL или на сайт). Если такого поиска найти невозможно, то использовать корпус для выполнения лабораторных работ нельзя!
- Привести несколько примеров запросов к существующим поисковикам, указать недостатки в полученной поисковой выдаче.

В результатах работы должна быть указаны статистическая информация о корпусе:

- Размер «сырых» данных.
- Количество документов.
- Размер текста, выделенного из «сырых» данных.
- Средний размер документа, средний объём текста в документе.

Описание

В рамках работы была выполнена подготовка корпуса документов, который в дальнейшем используется для построения поисковых индексов и реализации поисковых алгоритмов.

В качестве источников данных были выбраны открытые текстовые ресурсы, представленные в сети Интернет. Была выбрана тема «Информационная безопасность», по ней содержится много статей в Википедии (категория «Информационная безопасность» и обход её по подкатегориям) и на сайте Securitylab.ru (есть как новости разного размера, так и аналитические статьи). На портале Securitylab статьи расположены в классическом формате - набор страниц со статьями, на каждой странице максимум 15 статей.

Исходные документы представляли собой HTML-страницы, содержащие текстовую информацию, разметку, навигационные элементы, ссылки, служебные блоки и дополнительную мета-информацию. Разметка документов выполнена с использованием HTML и включает структурные элементы (заголовки, списки, таблицы), а также вспомогательные элементы, не относящиеся непосредственно к основному тексту статьи. К информации, которая есть в статьях у обоих источников, относятся название статьи, общая краткая информация (summary) и основной текст статьи.

Для обоих источников данных доступен встроенный поиск по сайту, а также поиск с использованием поисковых систем общего назначения (например, Yandex) с ограничением на сайт.

Примеры запросов:

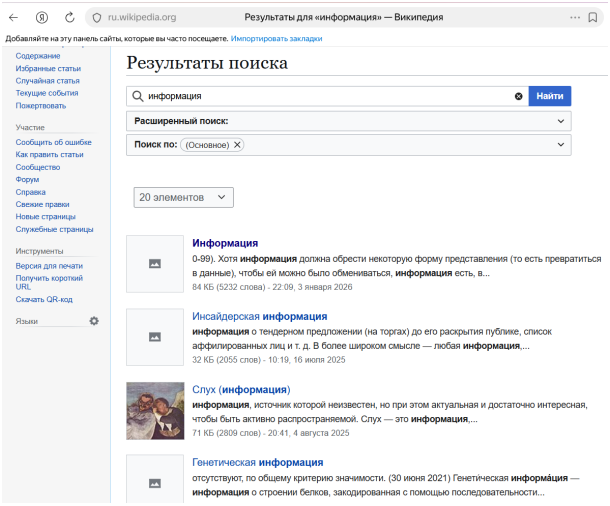

Рис. 1: Поиск в Википедии статей, содержащих слово «информация»

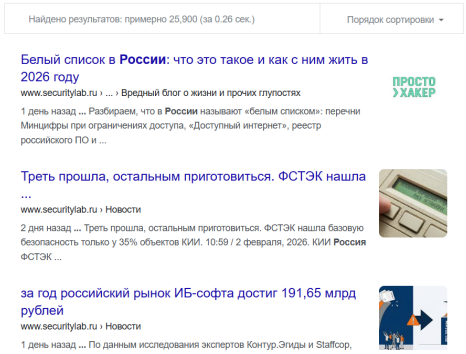

Рис. 2: Поиск статей по ключевому слову «Россия» на портале Securitylab (Оптимизирован Google)

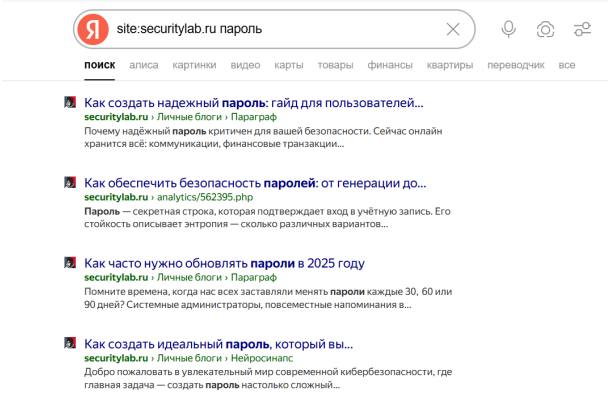

Рис. 3: Поиск в Яндекс с ограничением на сайт Securitylab.ru по слову «пароль»

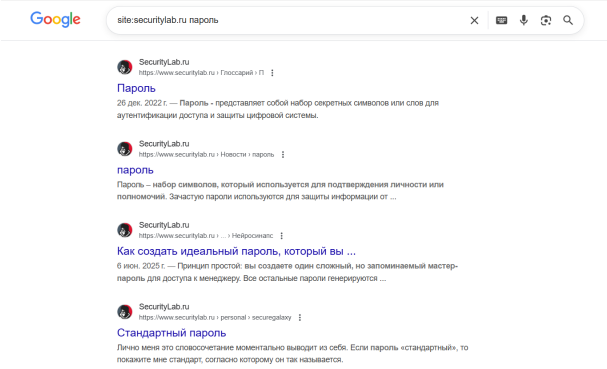

Рис. 4: Поиск в Google с ограничением на сайт Securitylab.ru по слову «пароль»

Анализ существующих поисковых систем выявил в них несколько недостатков. Поиск в Википедии по ключевому слову «Информация» возвращает не только статьи, включающие это слово, но и перенеправления с параграфов некоторых больших статей (например, о государствах). Поиск через Google с ограничением на сайт SecurityLab формирует результаты с приоритетом релевантности, а не актуальности - находятся в том числе очень старые статьи. Поиск внутри сайта предусматривает фильтрацию - по дате или по релевантности, однако в случае сортировки по дате статьи сортируются нестрого (скорее всего все равно учитывается релевантность - более старая статья может быть выше в списке, чем более свежая, что противоречит поиску исключительно по дате). Поиск в Яндекс с ограничением на Securitylab.ru держит баланс между датой и релевантностью (выдаются сравнительно новые статьи по теме), но не поддерживает четкой фильтрации.

Статистическая информация

Размер исходных (сырых) данных: 1.91 Гбайт.

Средний размер одного сырого документа: 59.67 Кбайт.

Общее количество сырых документов в корпусе — 33 616.

После этапа очистки и выделения текстового содержимого в ходе фильтрации в корпусе осталось 32 182 документа.

Совокупный размер выделенного текста: 199.64 Мбайт.

Средний объём текста в одном документе после очистки: 6.35 Кбайт.

Исходный код

В рамках выполнения работы была спроектирована архитектура извлечения и обработки текстовых данных, предназначенная для формирования корпуса документов, пригодного для последующих лабораторных работ.

На первом этапе выполняется загрузка исходных документов из открытых источников (Википедия и SecurityLab) с сохранением оригинального содержимого в базу данных. На втором этапе производятся разбор и очистка текстов: удаляются разметка, служебные элементы и неконтентные фрагменты, нормализуется кодировка и выделяется текстовое содержимое. Результаты обработки сохраняются в отдельной базе данных, содержащей только очищенный текст.

Для дальнейшего анализа и построения поисковых индексов очищенные данные экспортируются в табличный формат `TSV`, где каждая строка соответствует одному документу и содержит уникальный идентификатор и текст документа, сформированный как конкатенация заголовка, краткого описания и основного текста в формате `id\t title + summary + text\n`. Дальнейшая работа ведётся исключительно с полученным TSV-файлом.

Лабораторная работа №2 «Поисковый робот»

Необходимо написать парсер на любом языке программирования.

Написать поисковый робот — компоненты обкачки документов, используя любой язык программирования;

- Единственным аргументом поисковому роботу подаётся путь до yaml-конфига, содержащий:
  - Данные для базы данные в секции db;
  - Данные для робота в секции logic: задержка между обкачкой страницы;
  - Любые другие данные, необходимые для реализовации логики поискового робота.
- Сохранять в базе данных (например, MongoDB) документы со следующими полями:
  - url, нормализованный;
  - «сырой» html-текст документа;
  - название источника;
  - Дата обкачки документа в формате Unix time stamp.
- Поисковый робот можно остановить в любой момент и при повторном запуске робот должен начать с того документа, с которого он остановился;
- Периодически он должен уметь переобкачивать документы, которые уже есть в базе, но только в том случае, если они изменились.

Описание

В рамках лабораторной работы реализован поисковый робот, предназначенный для автоматической обкачки документов из сети Интернет с последующим сохранением результатов в базе данных.

В качестве источников данных используются несколько типов веб-ресурсов. Для энциклопедических данных (Википедия) применяется иерархическая структура категорий, что позволяет организовать обход документов с учётом вложенности и глубины. Для новостных и аналитических ресурсов (Securitylab) используется постраничный обход с извлечением ссылок на отдельные публикации.

Все ссылки приводятся к нормализованному виду, что позволяет избежать дублирования документов и обеспечить целостность данных в базе. Для определения изменений содержимого документов применяется вычисление хэш-суммы HTML-текста, что позволяет выполнять переобкачку только в случае фактического изменения страницы.

Также нужна была дополнительная компонента, выделяющая чистый текст из скачанных роботом HTML-страниц. Основной задачей обработки является извлечение из HTML-страниц структурированных текстовых данных, таких как заголовок документа, краткая аннотация и основной текст статьи. Для различных источников данных используются специализированные парсеры, учитывающие особенности структуры страниц.

В процессе обработки выполняется удаление шумов, не относящихся к содержимому документа, включая служебные блоки, рекламные элементы, скрипты и стили. Также производится нормализация текста: устранение лишних пробелов, переносов строк, артефактов пунктуации и т.д.

Исходный код

Программа реализована на языке Python и состоит из набора функций, каждая из которых отвечает за отдельный аспект работы поискового робота.

В начале программы производится загрузка конфигурации из YAML-файла, а также инициализация подключения к базе данных MongoDB. Создаются необходимые коллекции и индексы, обеспечивающие уникальность URL-адресов и корректную работу очередей заданий.

Для обработки ссылок реализована функция нормализации URL, устраняющая различия в регистре доменных имён, фрагментах и завершающих слэшах. Это решение позволяет избежать сохранения дублирующихся документов.

Сохранение документов в базе данных выполняется с учётом проверки изменений. Для каждого HTML-документа вычисляется хэш-сумма, которая сравнивается с ранее сохранённым значением. Если содержимое документа не изменилось, обновляется только временная метка, в противном случае документ сохраняется повторно.

Логика обкачки Википедии построена на использовании API для получения элементов категорий. Реализован механизм обхода как статей, так и подкатегорий с учётом глубины вложенности. Для Securitylab реализованы отдельные функции обкачки, основанные на постраничном обходе. Состояние обработки каждой категории сохраняется в БД, что позволяет корректно восстанавливать процесс при прерывании выполнения.

Обработка документов реализована в виде набора независимых модулей, каждый из которых отвечает за разбор HTML-страниц определённого источника. В основе парсеров используется библиотека BeautifulSoup, обеспечивающая удобную навигацию по DOM-структуре документа.

Для нормализации текста реализована функция очистки для каждого сайта отдельно, выполняющая удаление лишних пробелов, корректировку пунктуации и устранение служебных вставок, таких как ссылочные сноски. Это позволяет получить однородный текст, пригодный для дальнейшей обработки.

Финальный модуль обработки выполняет обход коллекции «сырых» документов, сохранённых поисковым роботом, и применяет соответствующий парсер в зависимости от источника данных. Результаты обработки сохраняются в отдельной коллекции базы данных, при этом выполняется фильтрация документов с недостаточным объёмом текста.

Как уже было упомянуто выше, далее корпус экспортируется из базы данных в TSV-файл в нужном формате.

Лабораторная работа №3 «Токенизация»

Нужно реализовать процесс разбиения текстов документов на токены, который потом будет использоваться при индексации. Для этого потребуется выработать правила, по которым текст делится на токены. Необходимо описать их в отчёте, указать достоинства и недостатки выбранного метода. Привести примеры токенов, которые были выделены неудачно, объяснить, как можно было бы поправить правила, чтобы исправить найденные проблемы.

В результатах выполнения работы нужно указать следующие статистические данные:

- Количество токенов.
- Среднюю длину токена.

Кроме того, нужно привести время выполнения программы, указать зависимость времени от объёма входных данных. Указать скорость токенизации в расчёте на килобайт входного текста. Является ли эта скорость оптимальной? Как её можно ускорить?

Описание

В рамках данной лабораторной работы реализован процесс токенизации текстового корпуса, предназначенный для подготовки данных к последующей индексации.

Входные данные представлены в виде текстового файла, содержащего корпус документов в табличном формате, где каждая строка соответствует отдельному документу. В процессе обработки игнорируются служебные идентификаторы документов, а анализу подвергается только текстовое содержимое.

Алгоритм токенизации реализован на уровне побайтового чтения входного файла. При разборе текста учитываются особенности кодировки UTF-8 и следующие правила токенизации:

– частями токенов являются только символы латинского и кириллического алфавитов;
– все символы приводятся в нижний регистр;
– слова с дефисом между их частей считаются как отдельный токен (дефис не увеличивает длину);
– корректная длина токена — от 3 до 40 символов;
– если символ не является буквой из латиницы/кириллицы или дефисом между частями токена, то он считается разделителем.

К достоинствам выбранного метода токенизации относятся простота реализации, предсказуемость результатов и корректная работа с многоязычными текстами. К недостаткам - некорректная обработка слов, содержащих цифры или специальные символы.

Примерами неудачной токенизации являются технические обозначения и составные термины со спецсимволами, такие как `covid-19`, `c++` или `tcp/ip`, для которых текущие правила приводят к потере части информации. Эти недостатки могут быть устранены путём расширения правил токенизации - добавлением математических знаков и добавлением некоторого количества цифр (можно было бы фильтровать длину цифровых последовательностей).

Исходный код

Реализация токенизатора выполнена на языке C++ и основана на потоковой обработке входного файла. Основной алгоритм реализован в функции `main (файл tokenizer.cpp)`, где осуществляется последовательное чтение байтов входного потока и формирование токенов в режиме реального времени.

Для распознавания символов используются отдельные функции, определяющие принадлежность байта к латинскому или кириллическому алфавиту. Для кириллических символов выполняется корректная обработка двухбайтовых последовательностей UTF-8. Для латиницы используется упрощённая ASCII-нормализация.

При встрече разделителя или недопустимого символа текущий токен считается завершённым и проходит проверку по длине. После этого выполняется учёт статистических характеристик.

Анализ результатов токенизации

В результате выполнения программы токенизации были получены следующие показатели:

- Количество токенов: 12 014 711.
- Средняя длина токена: 7.595 символа.
- Объём входного текста: 200 989 KB.
- Время выполнения программы: 10.2274 секунды.
- Скорость токенизации: 19 652 KB/сек.

Время выполнения алгоритма линейно зависит от объёма входных данных, так как каждый символ входного текста обрабатывается ровно один раз. Это позволяет утверждать, что сложность алгоритма токенизации является линейной.

Полученная скорость токенизации является высокой для однопоточной реализации с побайтовой обработкой UTF-8 текста и может считаться достаточной для практического применения. Тем не менее, данная скорость не является предельной и может быть улучшена за счёт оптимизаций.

Возможные направления ускорения включают отображение файла в память и параллельную обработку текста.

Лабораторная работа №4 «Стемминг»

Добавить в созданную поисковую систему стемминг - поиск без учёта словоформ.

Описание

В рамках данной лабораторной работы в поисковую систему был добавлен этап стемминга, предназначенный для приведения слов к базовой форме с целью поиска без учёта словоформ. Стемминг применяется после токенизации и выполняется для каждого выделенного токена.

Реализованный подход основан на эвристическом удалении наиболее распространённых суффиксов для английского и русского языков. Для английских слов используются простые правила усечения типичных словообразовательных окончаний. Для русских слов применяется набор часто встречающихся окончаний, учитывающий особенности склонений и спряжений.

Использование стемминга позволяет сократить количество различных терминов в индексе и повысить полноту поиска за счёт объединения различных словоформ одного слова.

Исходный код

Стемминг реализован в виде отдельной функции, принимающей токен по ссылке и изменяющей его содержимое. В начале функции определяется длина обрабатываемого слова, после чего выполняется проверка набора правил для английского языка.

Для английских слов последовательно проверяется наличие характерных суффиксов, таких как `ing`, `ion`, `ed`, `er`, а также окончания множественного числа. При совпадении соответствующий суффикс удаляется, и обработка слова завершается.

Для русских слов используется массив строк, содержащий наиболее распространённые окончания в кодировке UTF-8. Для каждого окончания выполняется проверка совпадения с концом слова, после чего окончание удаляется при условии, что длина оставшейся основы в байтах превышает минимальный порог.

Лабораторная работа №5 «Закон Ципфа»

Для своего корпуса необходимо построить график распределения терминов по частотностям в логарифмической шкале, наложить на этот график закон Ципфа. Объяснить причины расхождения.

Описание

В рамках данного этапа работы проводится анализ распределения терминов по частотам. Для этого на основе обработанного корпуса строится распределение частот терминов, после чего выполняется его сравнение с теоретической моделью.

На этапе подсчёта частот используется собственная хэш-таблица, в которую для каждого термина накапливается количество его вхождений в корпусе. После завершения обработки все пары «термин–частота» сохраняются в текстовый файл, который используется как входные данные для последующего анализа.

Для визуализации распределения частот используется логарифмическая шкала по обеим осям. Термины сортируются по убыванию частот, после чего каждому термину сопоставляется его ранг. Теоретическая кривая закона Ципфа строится в виде зависимости $f(r) = C/r$, где C равна частоте самого популярного термина в корпусе.

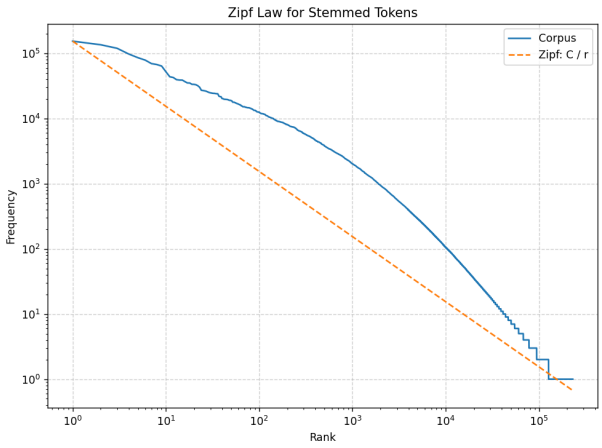

Рис. 5: График закона Ципфа

Полученный график демонстрирует общее соответствие распределения закону Ципфа, однако наблюдаются характерные отклонения. В начальной части графика расхождения связаны с высокой частотой служебных и общеупотребительных слов. В хвосте распределения отклонения объясняются наличием большого числа редких терминов.

Таким образом, эксперимент подтверждает применимость закона Ципфа к рассматриваемому корпусу, при этом наблюдаемые расхождения являются ожидаемыми и обусловлены особенностями реальных текстовых данных.

Исходный код

Подсчёт частот терминов реализован на языке C++ с использованием собственной хэш-таблицы фиксированного размера. В качестве ключей используются строки терминов, а значения представлены счётчиками количества их появлений. Для разрешения коллизий используется хранение нескольких элементов в одной ячейке хэш-таблицы в виде связанного списка.

В процессе обработки корпуса для каждого токена вычисляется хэш-значение, после чего соответствующая запись либо обновляется, либо создаётся заново. По завершении обработки содержимое хэш-таблицы последовательно выгружается в файл, где каждая строка содержит термин и его частоту, разделённые символом табуляции. Из этого файла и строится рассматриваемый график.

Лабораторная работа №6 «Булев индекс»

Требуется построить поисковый индекс, пригодный для булева поиска, по подготовленному в ЛР1 корпусу документов.

Описание

Реализован булев поисковый индекс, предназначенный для выполнения логического поиска по корпусу документов. Индекс строится в виде инвертированной структуры, в которой каждому термину сопоставляется список идентификаторов документов, содержащих данный термин.

В процессе построения индекса корпус документов последовательно обрабатывается построчно. Для каждого документа формируется набор терминов, после чего для каждого термина в индекс добавляется идентификатор соответствующего документа. Повторное добавление одного и того же идентификатора для одного термина не допускается.

Исходный код

Булев индекс реализован в виде собственной инвертированной структуры данных. В основе индекса используется хэш-таблица фиксированного размера, в которой каждому термину сопоставляется список идентификаторов документов. Хэш-функция применяется для распределения терминов по ячейкам таблицы.

Каждый элемент индекса представлен структурой, содержащей строку термина и динамический список документов, в которых данный термин встречается. При добавлении нового термина в индекс выполняется поиск соответствующей записи; в случае её отсутствия создаётся новый элемент, иначе список документов дополняется новым идентификатором.

Лабораторная работа №7 «Булев поиск»

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

Описание

Реализован механизм булевого поиска по ранее построенному инвертированному индексу. Булев поиск позволяет формировать запросы с использованием логических операторов `AND`, `OR` и `NOT` и получать список документов, удовлетворяющих заданному логическому выражению.

Поисковый запрос предварительно разбивается на последовательность токенов, среди которых выделяются термины и логические операторы. Операторы представлены как управляющие элементы запроса и определяют способ объединения или исключения результатов поиска по отдельным терминам.

Для каждого термина из запроса из индекса извлекается список идентификаторов документов. Далее над этими списками последовательно выполняются операции пересечения, объединения или разности в зависимости от текущего логического оператора.

Итогом выполнения булевого запроса является список идентификаторов документов, которые удовлетворяют заданным условиям поиска и выводятся пользователю в качестве поисковой выдачи.

Исходный код

Реализация булевого поиска основана на последовательной обработке токенов запроса. Для выполнения поиска используется функция, принимающая строку запроса и возвращающая список идентификаторов документов, соответствующих результату булевого выражения.

Для поддержки логических операций реализованы отдельные функции для выполнения операций `AND`, `OR` и `NOT`. Каждая из этих функций принимает два списка идентификаторов документов и возвращает новый список, полученный в результате соответствующей логической операции. Операция `AND` реализуется как пересечение списков, `OR` — как объединение, а `NOT` — как разность.

В процессе выполнения поиска текущий результат накапливается по мере обработки терминов запроса. При встрече логического оператора изменяется режим обработки, после чего результат поиска по очередному термину комбинируется с текущим результатом согласно выбранной операции.

Дополнительно при реализации программы учитывались особенности ввода и обработки текста в различных операционных системах. В процессе разработки на ОС Windows было выявлено, что корректную обработку кириллического ввода из стандартного потока (`stdin`) существенно тяжелее реализовать, чем в Linux из-за различий в используемых системных кодировках (кириллица в Linux обрабатывается корректно изначально в кодировке UTF-8).

В связи с этим итоговое решение было разделено на две версии:

- версия для Windows, использующая системные средства преобразования кодировок и обработку аргументов командной строки;
- версия для Linux, ориентированная на работу со стандартными потоками ввода и вывода в кодировке UTF-8.

Разделение реализации по операционным системам позволило обеспечить корректную обработку поисковых запросов, содержащих кириллические символы, а также выполнить требования задания по поддержке утилиты командной строки с использованием стандартного ввода и вывода. С точки зрения логики булевого поиска обе версии программы идентичны и отличаются исключительно способом получения входных данных и настройками кодировки.

Примеры запросов к системе и поисковой выдачи:

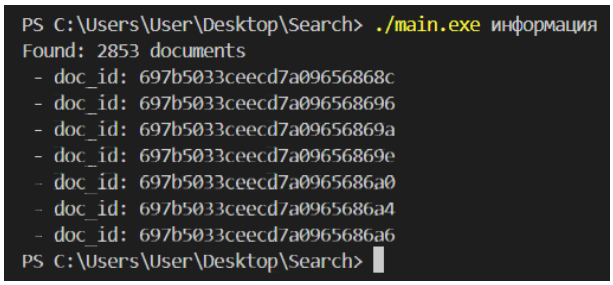

Рис. 6: Пример 1

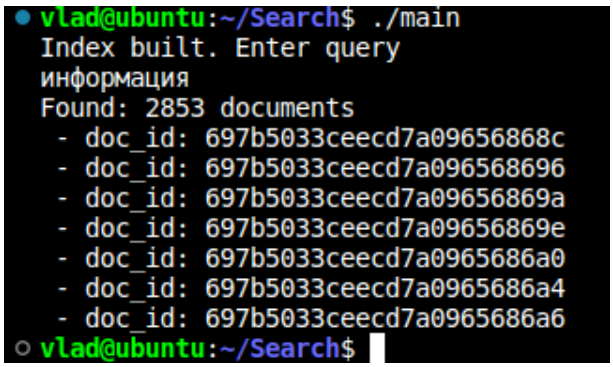

Рис. 7: Пример 1 через стандартный ввод (Linux) - как видим, смена способа ввода и ОС не повлияла на результат работы программы. А значит, примеры ввода и выдачи через Linux и Windows с точки зрения конечного результата будут идентичны

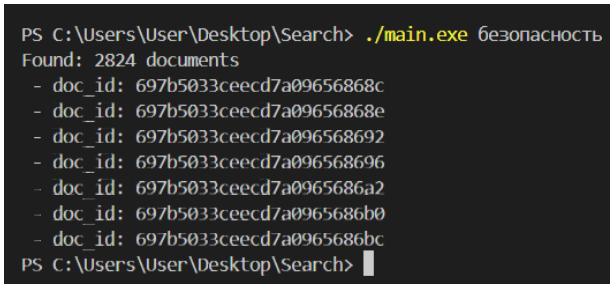

Рис. 8: Пример 2

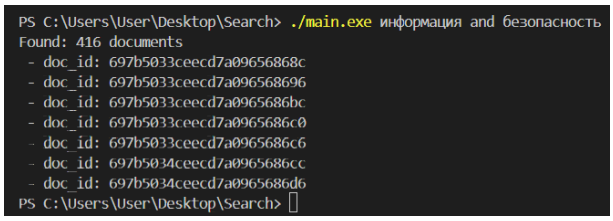

Рис. 9: Пример 3

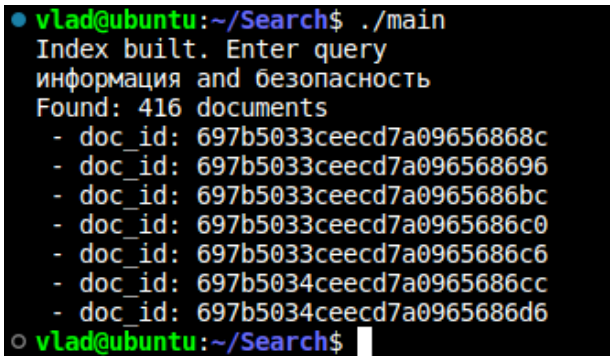

Рис. 10: Пример 3 в ОС Linux - результат идентичный Windows-функции

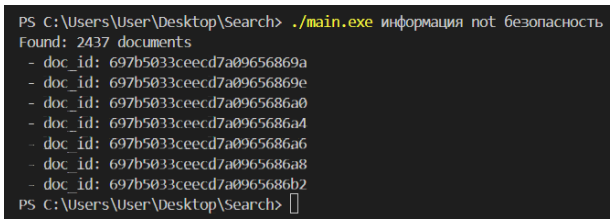

Рис. 11: Пример 4

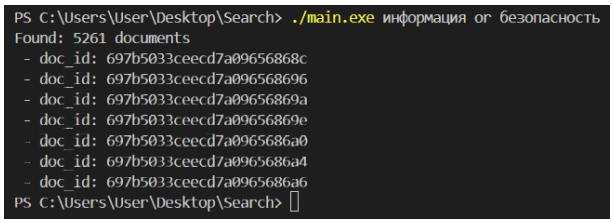

Рис. 12: Пример 5

Выводы

В ходе выполнения работы я последовательно изучил полный цикл построения простой поисковой системы, начиная с этапа добычи корпуса документов и заканчивая реализацией булевого поиска по собственному инвертированному индексу.

Я получил практический опыт реализации токенизации текстов с учётом особенностей латинского и кириллического алфавитов, а также осознал влияние выбранных правил токенизации на дальнейшую поисковую выдачу.

В рамках работы над частотным анализом корпуса я изучил распределение терминов без учета словоформ и на практике проверил закон Ципфа в практических условиях, что позволило лучше понять статистическую природу текстовых данных и особенности естественных языков.

Реализация инвертированного индекса и булевого поиска дала понимание принципов организации поисковых структур данных и алгоритмов обработки логических запросов. Я убедился, что даже простая реализация индекса позволяет эффективно выполнять поиск по большому корпусу документов.

Изначально разработка и тестирование системы выполнялись в среде Windows, однако в процессе работы были выявлены сложности, связанные с обработкой кириллического ввода и особенностями кодировок при использовании стандартного ввода. Благодаря тому, что взаимодействие между компонентами системы было организовано через простой текстовый формат `TSV`, а не через привязку к базе данных, переход на среду Linux не потребовал изменения логики работы программы. Это позволило устранить проблемы с кодировкой и получить идентичные по корректности и содержанию результаты работы поисковой системы. Также была создана отдельная `main`-функция для среды Windows, в которой ввод поискового запроса осуществляется через аргументы командной строки, получаемые с помощью функции `CommandLineToArgvW` (массив `argvW`). Строка запроса далее явно преобразуется из UTF-16 в UTF-8, что позволяет корректно обрабатывать кириллический ввод в консоли Windows. Эта проблема показала, что следует тщательно подходить к выбору ОС для выполнения работ, ограничивающих выбор кодировок и способов ввода.

Полученные знания и навыки могут быть использованы при разработке более сложных поисковых систем, систем анализа текстов, а также при работе с большими объёмами неструктурированных данных в прикладных и исследовательских задачах.