Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа по курсу «Информационный поиск»

Студент: Н.Х.А. Нгуен
Преподаватель: А. А. Кухтичев
Группа: М8О-409Б-22
Дата:
Оценка:
Подпись:

Москва, 2025

1 Добыча корпуса документов

1 Источник и характеристика корпуса

В качестве источника данных был использован портал научных публикаций CyberLeninka (`cyberleninka.ru`). Сбор документов производился из двух тематических разделов: «Клиническая медицина» и «Науки о здоровье».

При анализе структуры HTML-документов было установлено, что страницы имеют семантическую разметку с использованием атрибутов микроразметки Schema.org. Заголовок статьи располагается в теге с атрибутом `itemprop="headline"`, а основной текст - в блоке `div` с `itemprop="articleBody"`. Данный подход позволил реализовать парсер, устойчивый к изменениям в верстке страниц.

В мета-тегах страниц также присутствует дополнительная информация: автор, год публикации, название журнала и др. Эти данные могут быть использованы в дальнейшем для реализации расширенного поиска. Тексты статей имеют научный стиль и стандартную структуру.

2 Анализ существующих поисковых систем

Был проведен сравнительный анализ двух поисковых систем: встроенного поиска на сайте CyberLeninka и поиска Google с оператором `site:cyberleninka.ru`.

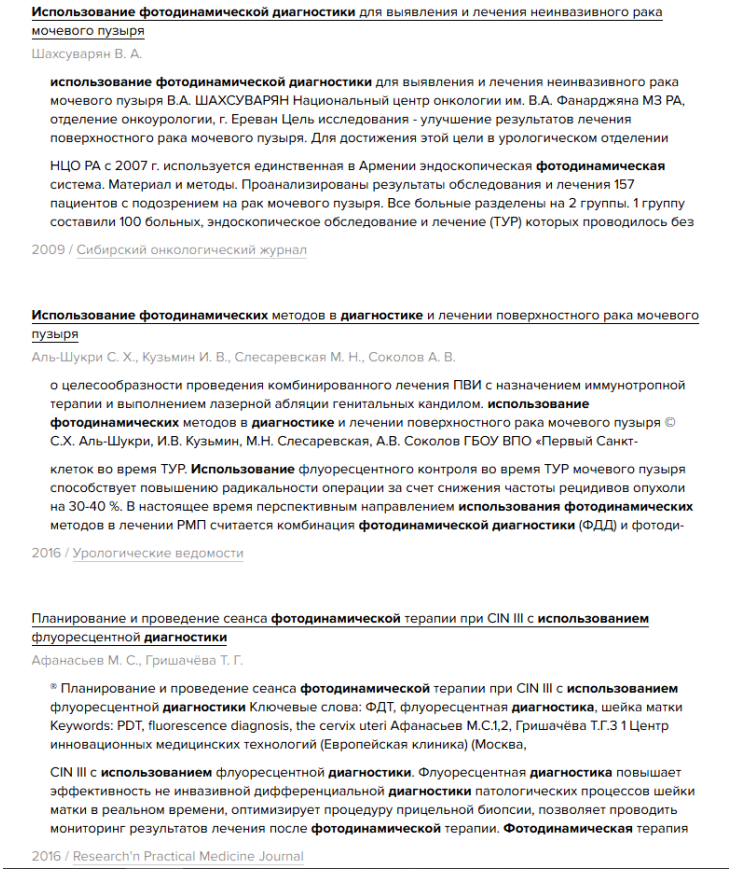

Рис. 1: Запрос "использование фотодинамической диагностики"во встроенном поисковике CyberLeninka

Сниппеты во встроенном поиске включают два фрагмента текста, содержащие запрос. Ранжирование отдает приоритет статьям, где запрос встречается в заголовке. Присутствует указание автора и года публикации.

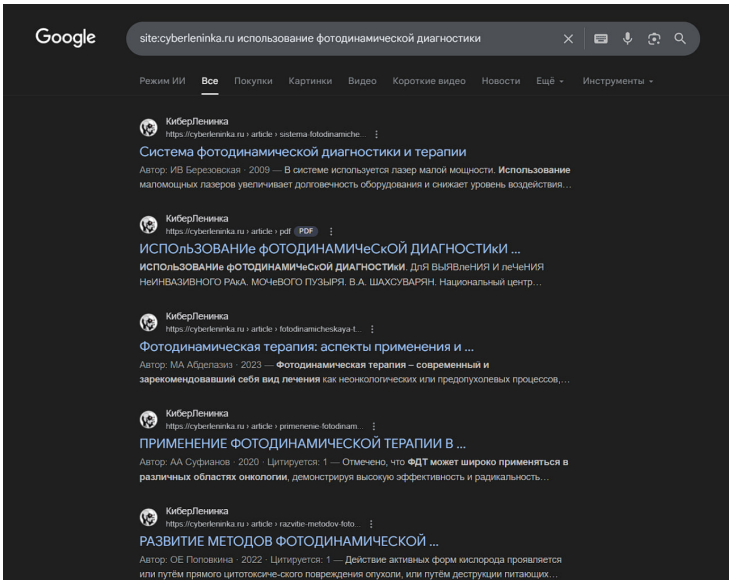

Рис. 2: Запрос "использование фотодинамической диагностики"в поисковике Google

Ранжирование, вероятно, отдает приоритет наиболее популярным статьям. Также присутствует указание автора и года публикации.

3 Итоговая статистика корпуса

По результатам работы был собран и обработан корпус из 30176 документов.

Параметр	**Значение**
Источник	cyberleninka.ru
Количество документов	30176
Общий размер «сырых» HTML	2.26 ГВ
Общий размер очищенного текста (TXT)	759.89 MB
Средний размер «сырого» документа	76.67 KB
Средний объём текста в документе	25.79 KB

2 Поисковый робот

Описание метода

Для сбора корпуса документов был разработан поисковый робот на языке Python. Архитектура робота включает следующие компоненты:

- **Управляющий конфигурационный файл** `config.yaml`, который задает стартовые URL, разрешенные домены и параметры работы.
- **База данных MongoDB** для хранения скачанных HTML-документов и очереди URL для обхода. Использование двух коллекций (для документов и для очереди) обеспечивает отказоустойчивость. Робот может быть остановлен и перезапущен, продолжая работу с места остановки.
- **Основной цикл**, который атомарно извлекает URL из очереди, скачивает страницу, сохраняет ее и извлекает новые ссылки.
- **Механизм «вежливости»** - реализована задержка между запросами для снижения нагрузки на сервер.
- **Детектор CAPTCHA**, который останавливает работу при обнаружении признаков защиты от автоматических запросов, чтобы избежать блокировки.

3 Токенизация

1 Правила токенизации

Процесс разбиения текста на токены реализован на C++ в виде утилиты командной строки, работающей как фильтр. Были выработаны следующие правила:

- **Приведение к нижнему регистру:** Все символы кириллицы и латиницы переводятся в нижний регистр.
- **Определение границ токена:** Разделителями считаются пробелы, знаки препинания и символы перевода строки. Они не включаются в состав токена.
- **Состав токена:** Токеном считается непрерывная последовательность букв, цифр и дефиса (если он не в начале или конце слова).
- **Фильтрация:** Отбрасываются токены, состоящие только из цифр или дефисов, а также токены длиной менее двух символов после всех преобразований.

Достоинства: Высокая скорость обработки за счет реализации на C++ и использования конечного автомата, простота правил. **Недостатки:** Возможна неверная обработка сложносоставных слов или аббревиатур (например, «С.-Петербург»).

2 Результаты и анализ производительности

- Количество токенов: **81 326 141**
- Средняя длина токена: **11.90**
- Скорость токенизации: **19.45 мБ/с**

4 Анализ распределения: Закон Ципфа

1 Графическое представление

Для анализа частотного распределения терминов в корпусе был построен график в логарифмических координатах (Рис. 3). На график наложено эмпирическое распределение и теоретическая кривая Закона Ципфа.

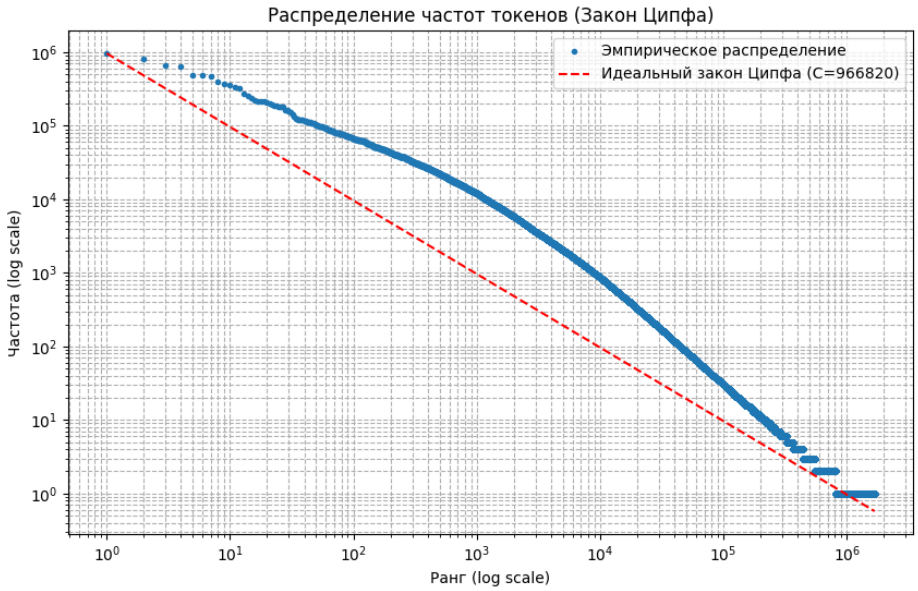

Рис. 3: Распределение частот токенов в логарифмических координатах

2 Объяснение расхождений

График демонстрирует, что распределение в целом следует закону Ципфа, однако наблюдаются характерные отклонения от идеальной модели:

- **«Голова» (начало графика):** Небольшое количество самых частотных терминов встречаются чаще, чем предсказывает теория. Это объясняется тематической однородностью корпуса (медицина), где ключевые термины («пациент», «лечение») доминируют.
- **«Тело» (середина графика):** Эмпирическая кривая идет почти параллельно теоретической, что подтверждает общую закономерность.

- **«Хвост» (конец графика):** Наблюдается большое количество слов с очень низкой частотой, включая множество слов, встретившихся всего один раз (гапаксы). Это приводит к появлению характерных «ступеней» на графике и является фундаментальным свойством естественного языка.

5 Нормализация: Стемминг

1 Описание метода

Для нормализации токенов и сокращения словаря был реализован и интегрирован в токенизатор стеммер Портера для русского языка. Стемминг применяется к каждому токену после приведения к нижнему регистру и до проверки на валидность.

2 Оценка качества поиска

Поскольку полноценная поисковая система еще не реализована, оценка качества проводилась качественно, на основе анализа гипотетических запросов.

Улучшение качества: Стемминг значительно повышает полноту поиска. Например, запрос `[современная терапия]` после обработки превратится в `[современ терап]`. Такой запрос найдет документы, содержащие словосочетания «современной терапии», «современную терапию» и т.д., которые не были бы найдены без нормализации.

Ухудшение качества (потеря точности): Агрессивный характер стемминга может приводить к ложным срабатываниям. Например, стеммер может ошибочно свести разные по смыслу слова к одной основе. Гипотетический пример: слова «универсальный» и «университет» могут быть сведены к общей основе «универс», что приведет к появлению нерелевантных документов в выдаче.

Способы улучшения: Для решения проблемы потери точности можно использовать более сложный метод нормализации - лемматизацию, которая приводит слово к его нормальной словарной форме (лемме) с использованием словарей. Однако этот метод значительно медленнее стемминга.

6 Булев индекс

1 Внутреннее представление данных

Для хранения индекса был разработан собственный бинарный формат, состоящий из трех файлов.

- **Прямой индекс** (`forward.idx`)**:** Хранит URL и заголовок для каждого документа. Файл состоит из таблицы смещений и области данных, что обеспечивает быстрый доступ к информации по `DocID`.
- **Словарь** (`dictionary.dat`)**:** Содержит лексикографически отсортированный список всех уникальных термов. Для каждого терма хранится его длина, частота встречаемости в документах (`doc_frequency`) и смещение в файле `postings.dat`.
- **Списки вхождений** (`postings.dat`)**:** Хранит последовательности `DocID` для каждого терма. Для экономии места и подготовки к сжатию хранятся не сами `DocID`, а их разницы (`d-gaps`).

2 Метод сортировки

Для лексикографической сортировки словаря в памяти использовалась стандартная функция C `qsort`.

- **Достоинства:** Реализация быстрой сортировки (в среднем O(N log N)), доступная в стандартной библиотеке C, не требует написания с нуля.
- **Недостатки:** Алгоритм работает только с данными, полностью помещающимися в оперативную память. Для корпусов большего размера он неприменим.

3 Результаты

- Количество термов: **1 672 481**
- Средняя длина терма: **14.71 символов**
- Скорость индексации (общая): **202.82 мс**
- Скорость индексации (на документ): **6.76 мс**

Анализ производительности и масштабирования: Текущая реализация ограничена объемом оперативной памяти. Малый размер хэш-таблицы к длинным цепочкам коллизий, что замедляет поиск. Для ускореения можно увеличить этот размер.

При увеличении объема данных в 10-1000 раз программа может завершиться с ошибкой нехватки памяти.

7 Булев поиск

1 Скорость выполнения запросов

Скорость измерялась на наборе из 1000 тестовых запросов разной сложности.

- Средняя скорость выполнения запроса: **1339.13 мс**

2 Примеры сложных запросов

- **Запросы с большим количеством операторов OR:** Запрос вида `[результат || исследование || данные]` вызывает длительную работу, так как требует слияния нескольких очень длинных списков документов, что создает большие промежуточные результаты.
- **Запросы с оператором NOT от редкого слова:** Запрос `[!гапакс]` является дорогим, так как требует создания списка почти всех документов корпуса и последующего исключения из него одного элемента. В реальных системах такие запросы часто запрещают или выполняют в связке с `AND`.

3 Тестирование корректности

Корректность поисковой выдачи проверялась в два этапа:

1. **Модульное тестирование:** Были созданы небольшие, заранее определенные списки `DocID`, на которых проверялась корректность работы функций пересечения (`AND`), объединения (`OR`) и инверсии (`NOT`).
2. **Интеграционное тестирование:** Был создан тестовый мини-корпус из 5 документов, для которого индекс был построен «вручную». Затем было выполнено 10-15 булевых запросов разной сложности. Результаты, выданные программой, сравнивались с ожидаемыми, вычисленными вручную. Этот метод позволил проверить корректность всей цепочки: от парсинга запроса до выполнения операций.

4 Интерфейсная часть

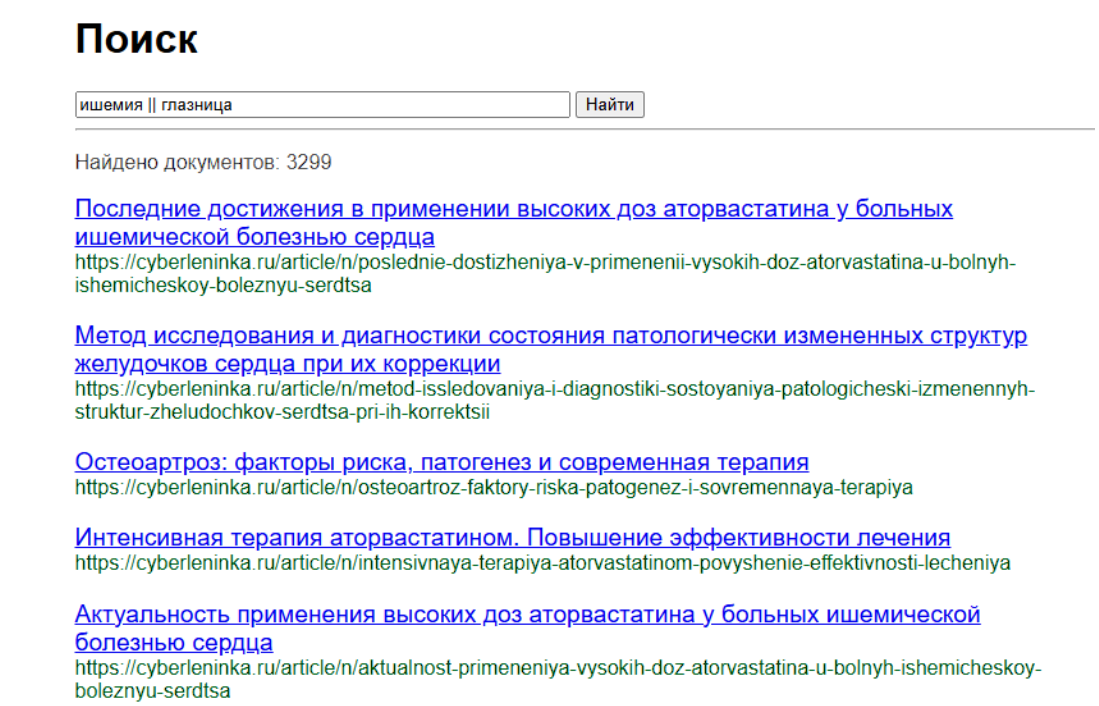

Рис. 4: Пример запроса

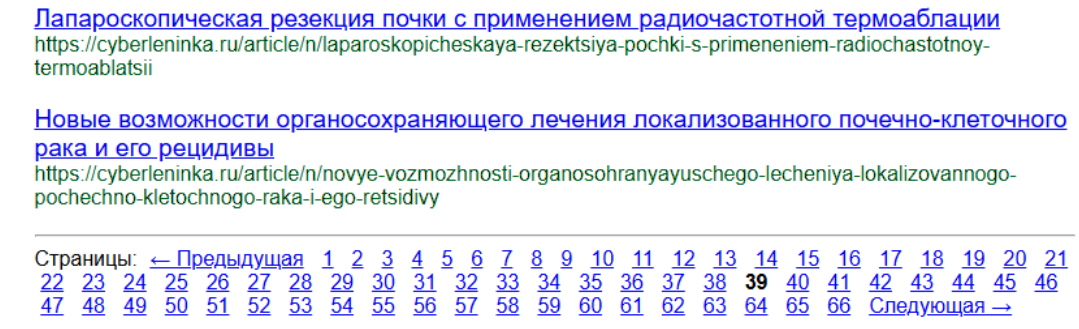

Рис. 5: Пагинация

8 Требования к компиляции и запуску

1 Требования к окружению

- ОС: Linux-совместимая (например, Ubuntu в WSL)
- Компилятор C++: g++
- Интерпретатор Python 3.8+
- Система управления пакетами Python: pip
- База данных: MongoDB (запущенная через Docker)

2 Последовательность сборки и запуска

1. Установить зависимости Python из файла `requirements.txt`:

```
pip install -r requirements.txt
```

2. Скомпилировать C++-утилиты (токенизатор, индексатор, поисковик):

```
g++ -O2 -o tokenizer tokenizer.cpp
g++ -O2 -o indexer indexer.cpp
g++ -O2 -o searcher searcher.cpp
```

3. Запустить Docker-контейнер с MongoDB:

```
docker start mongo_container
```

4. Запустить процесс построения индекса с помощью скрипта-оркестратора:

```
./run_indexing.sh
```

5. Запустить веб-сервис для демонстрации поиска:

```
python3 app.py
```

6. Открыть в браузере адрес `http://127.0.0.1:5000`.

9 Выводы

В ходе выполнения лабораторных работ была спроектирована и реализована поисковая система с поддержкой булева поиска. Был пройден весь цикл разработки: от сбора корпуса документов с помощью поискового робота до создания веб-интерфейса для взаимодействия с пользователем.

Ключевые технические задачи, решенные в ходе работы:

- Реализация основных компонентов (токенизатор, индексатор, поисковик) на языке C++.
- Проектирование собственного бинарного формата для хранения прямого и обратного индексов.
- Реализация классических алгоритмов информационного поиска: стеммера Портера, алгоритма «Сортировочная станция» для парсинга запросов и алгоритмов слияния для выполнения булевых операций.

Основным ограничением текущей реализации является работа индексатора только с данными, помещающимися в оперативную память.

Разработанная система успешно выполняет поставленные задачи, демонстрирует корректную работу булевой логики и является прочной основой для дальнейшего расширения функциональности.

Список литературы

[1] Маннинг К., Рагхаван П., Шютце Х. *Введение в информационный поиск.* — М.: Вильямс, 2011.

[2] Ахо А., Ульман Дж. *Теория синтаксического анализа, перевода и компиляции. Том 1: Синтаксический анализ.* — М.: Мир, 1978.

Приложение

Исходный код проекта: https://github.com/tng00/information-retrieval