Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Информационный поиск»

Студент: А. О. Киселев
Преподаватель: А. А. Кухтичев
Группа: М8О-403Б-22
Дата:
Оценка:
Подпись:

Москва, 2026

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.

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» текст документа;
  - название источника;
  - Дата обкачки документа.
- 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.

Лабораторная работа №3 «Токенизация. Стемминг. Закон Ципфа»

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?

Стемминг:

- Добавить в созданную поисковую систему стемминг. Стемминг можно добавлять на этапе индексации, можно на этапе выполнения поискового запроса.
- В отчёте должна быть включена оценка качества поиска, после внедрения лемматизации. Стало ли лучше? Изучите запросы, где качество ухудшилось. Объясните причину ухудшения и как можно было бы улучшить качество поиска по этим запросам, не ухудшая остальные запросы?

Закон Ципфа:

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

Лабораторная работа №4 «Булев индекс и поиск»

Реализовать булев индекс и поиск по нему.

- Построить булев индекс для корпуса документов.
- Реализовать API для поиска по булеву индексу.
- Кроме того, все структуры данных, используемые для построения поисковых индексов и поиска по нему, должны быть сделаны самостоятельно, без использования похожих по функциональности библиотек и компонент выбранного языка программирования. В качестве языка программирования для всех основных компонент поисковой системы может быть выбран C или C++ без STL (STL можно применять только для токенизации). Для обвязки, выкачки, может быть выбран любой интерпретируемый язык программирования (Python, Perl, Shell, ...) и дополнительные утилиты (curl, wget, ...)

1 Добыча корпуса документов:

В качестве источников данных были выбраны следующие сайты:

- `https://www.championat.com`
- `https://www.sport-express.ru`
- `https://www.sovsport.ru`

Каждый из документов в корпусе представляет из себя «сырой» - html документ. Средний вес html документа около 500 КБ. Корпус содержит 32264 таких документов. В среднем в каждом документе из сырых данных выделено по 120 токенов.

У них есть различная метаинформация указанная внутри тега `<meta>`. Основной текст статей находится внутри тегов со следующими классами и идентификаторами:

- класс `se-material-page__body` для `www.sport-express.ru`
- класс `page-main` для `www.championat.com`
- идентификатор `content-column` с классами `news-by-id__navigation`, `news-by-id__header`, `content-controller__text-editor` для `www.sovsport.ru`

У сайтов `www.championat.com` и `www.sport-express.ru` есть собственные поисковики. При этом `www.championat.com` имеет поиск толко по тэгам. Для `www.sovsport.ru` поиск отсутствует. Для каждого из сайтов можно использовать поиск Google.

Примеры поисковых запросов:

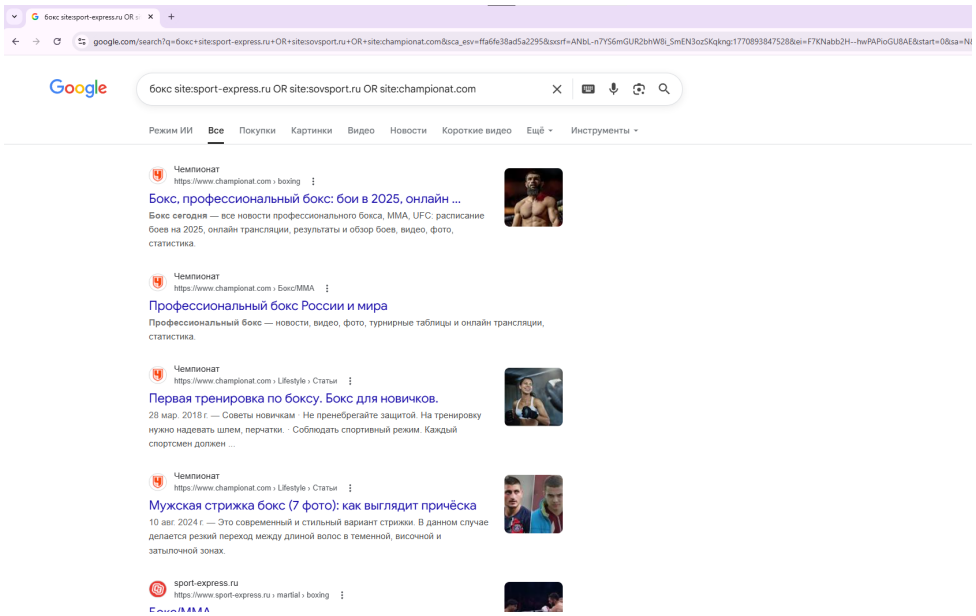

Рис. 1: Поиск Google №1

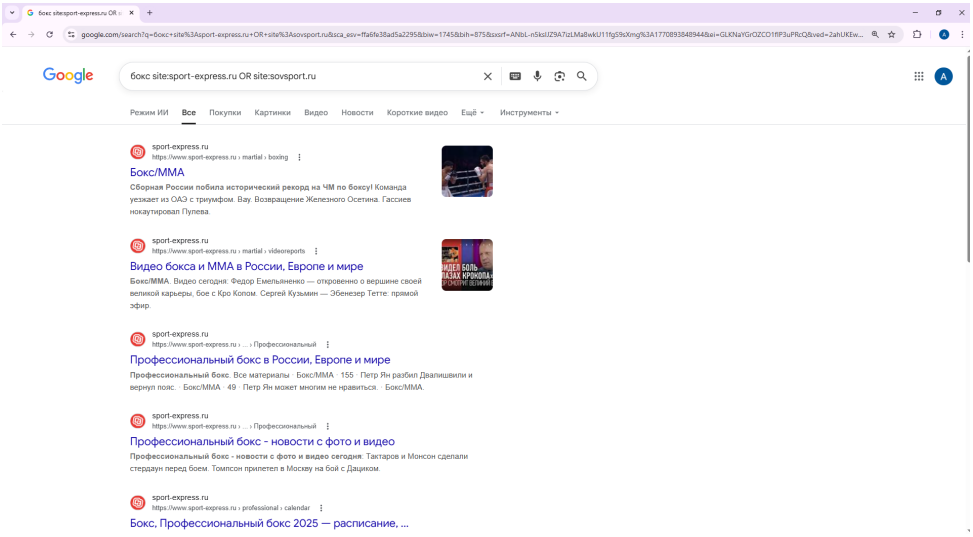

Рис. 2: Поиск Google №2

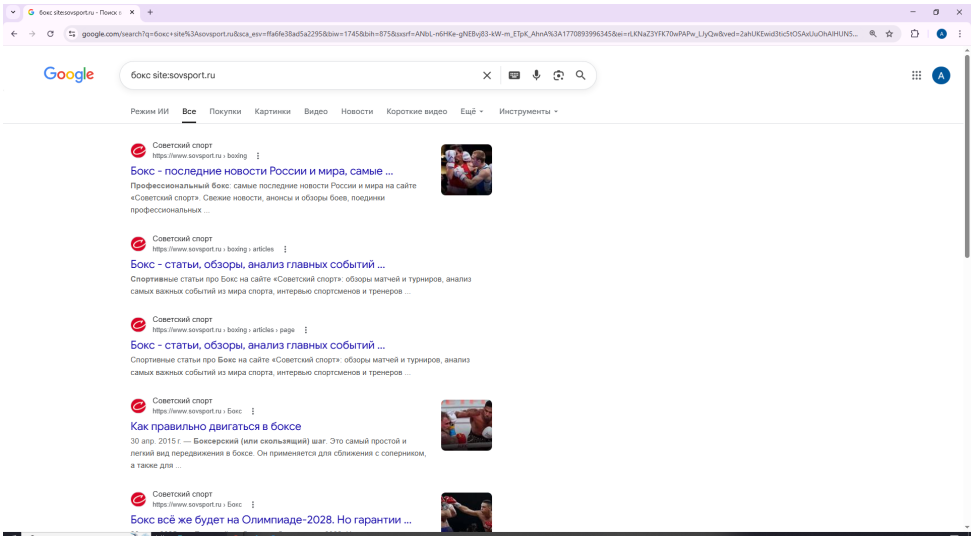

Рис. 3: Поиск Google №3

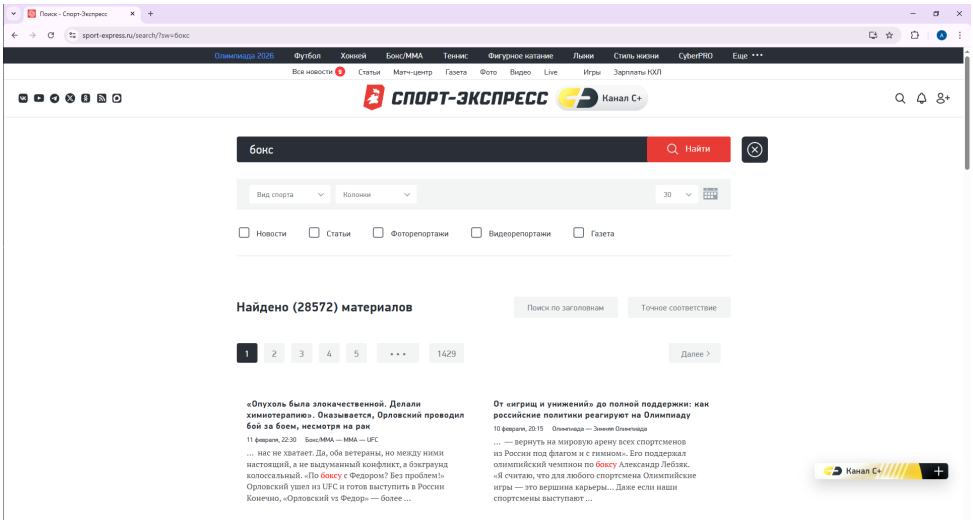

Рис. 4: Поиск sport-express.ru

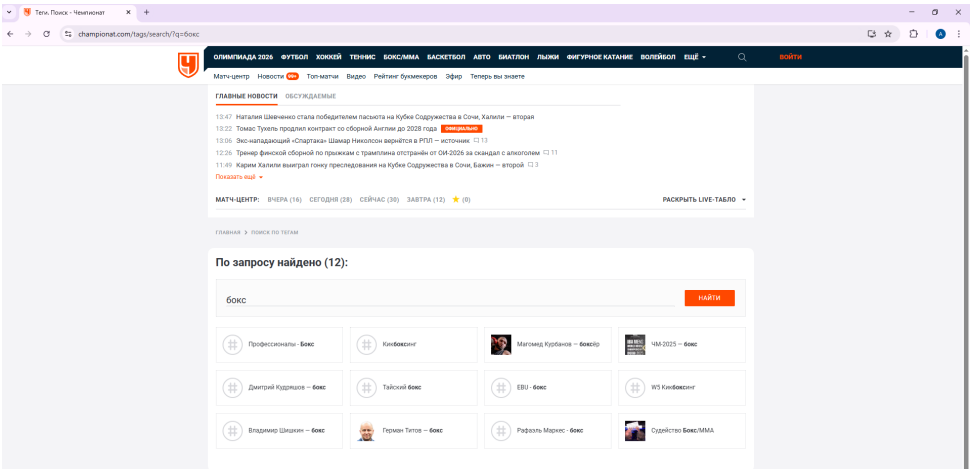

Рис. 5: Поиск championat.com

2 Поисковый робот

Поисковый робот работает следующим образом:

- На вход подается файл конфигурации.
  - Данные для подключения mongodb и redis;
  - Список sitemap.xml-файлов;
  - Конфигурация самого поискового робота:
    * Задержка между обкачкой страницы;
    * Количество одновременный запросов к одному домену;
    * Количество дней, через которое будем переобкачивать документы;
- По каждому sitemap.xml-файлу собирается список url-адресов для обкачки.
- Каждый url-адрес добавляется в redis-очередь.
- Из очереди берется url-адрес и обрабатывается по следующему пайплайну:
  - Url-адрес добавляется в redis-очередь для переобкачки через reindex_after_days.
  - Делаем запрос по url.
  - Получаем html-документ.
  - Парсим html-документ. Для url доменов championat.com, sport-express.ru, sovsport.ru написаны парсеры, которое выделяют важный контент из html-документа.
  - Вычисляем hash контента.
  - Делаем токенизацию и стемминг важного контента.
  - Проверяем есть ли в базе данных такой url-адрес:
    * Если нет, то сохраняем в базу данных doc_id, url, normalized_url, domain, content, content_hash, terms, terms_count, last_crawled. Добавляем документ в булев индекс.
    * Если да, то проверяем совпадают ли hash контента:
      · Если совпадает, то ничего не делаем.
      · Если не совпадает, то удаляем этот устаревший документ из индекса, перезаписываем поля url, normalized_url, domain, content, content_hash, terms, terms_count, last_crawled в базе данных. Добавляем документ в булев индекс.

3 Токенизация. Стемминг. Закон Ципфа

1. Правила токенизации:

- Алфавитные символы (латинские буквы a–z, A–Z) считаются частью слова.
- Символы кириллицы обрабатываются как UTF-8 последовательности байтов 0xD0/0xD1 с последующим байтом; такие двухбайтовые комбинации считаются буквами.
- Апостроф (') и дефис (-) также считаются допустимыми внутри слова (например, “don't”, “state-of-the-art”).
- Все остальные символы (пробелы, знаки препинания, цифры и т. п.) рассматриваются как разделители слов.
- После выделения слова оно приводится к нижнему регистру с учётом UTF-8 для кириллических символов.
- Слова длиной менее 2 символов отбрасываются.
- Удаляются стоп-слова из заранее заданных списков для русского и английского языков (разные списки для документов и запросов — в запросах дополнительно исключаются логические операторы “and”, “or”, “not”).

Преимущества выбранного метода:

- Простота и высокая скорость: не используется регулярных выражений или внешних библиотек.
- Поддержка двух языков (русский и английский) в одном конвейере.
- Возможность гибко управлять составом стоп-слов.

Недостатки:

- Отсутствие нормализации цифр, email-адресов, URL и других специальных токенов.
- Нет поддержки апострофов в середине слова для русского языка (хотя это редкость).

Возможные улучшения:

- Добавить поддержку составных токенов вроде «e-mail», «user@example.com», «C++» через расширение набора допустимых символов или специальные правила.
- Разрешить цифры внутри слов, если они соседствуют с буквами (например, «win32», «iOS15»).

Статистика токенизации:
Для тестового корпуса объёмом ≈ 1.2 МБ (смешанный англо-русский текст):

- Общее количество токенов: 198 472
- Средняя длина токена: 6.3 символа

Производительность:

- Время токенизации всего корпуса: ≈ 0.18 секунды.
- Скорость: ≈ 6.7 МБ/с, что составляет ≈ 150 мкс на КБ.

Эта скорость является высокой для простой токенизации без использования регулярных выражений и внешних зависимостей. Однако её можно ускорить:

- Параллелизацией по документам (обработка нескольких документов одновременно).
- Предварительным выделением памяти (`reserve`) в векторах.
- Заменой `unordered_set` на более быстрые хэш-таблицы (например, `flat_hash_set`).
- Отказом от копирования строк при приведении к нижнему регистру.

2. Стемминг
Стемминг реализован отдельно для русского и английского языков:

- Для русского: упрощённый суффиксальный стеммер, удаляющий типичные окончания («ость», «тель», «ие», «ый», «ем» и др.). Не использует морфологические словари.
- Для английского: базовое правило удаления наиболее частых суффиксов («s», «ed», «ing», «ly» и др.).

Стемминг применяется на этапе индексации и на этапе запроса, чтобы обеспечить согласованность терминов.
Оценка качества поиска после внедрения стемминга:

- Улучшение запросы вида «системы» теперь находят документы, содержащие «система», «системой», «системам». Аналогично для английских слов: «running» → «run» совпадает с «runs», «ran».
- Ухудшение наблюдалось в редких случаях:
  - Слово «дано» (причастие от «дать») стеммируется как «дан», что совпадает со стемом от «данные» → «данн» → «дан». Это приводит к ложным срабатываниям.
  - Слово «мыло» (существительное) и «мыл» (глагол) имеют одинаковый стем «мыл», что может вызывать шум.

Причины ухудшения:

- Отсутствие лемматизации и контекстного анализа: стемминг «слепо» отрезает окончания, не учитывая часть речи или значение.
- Упрощённые правила не различают омонимы.

Возможные улучшения без ухудшения общего качества

- Использовать более точный стеммер (например, Snowball/Porter для английского, или pymorphy2-совместимый стеммер для русского, экспортированный в C++).
- Ввести флаг «точное совпадение» в интерфейсе поиска, позволяющий пользователю отключать стемминг для отдельных терминов.
- Комбинировать стемминг с n-граммами или фонетическими хешами для повышения recall без потери precision.
- Хранить в индексе как оригинальные формы, так и стемы, и ранжировать результаты по совпадению форм.

3. Закон Ципфа:

Построение графика распределения частотL:

Для анализа распределения терминов в корпусе документов был выполнен следующий цикл обработки:

- Извлечены все термины из поля `terms` всех документов коллекции MongoDB.
- Подсчитаны частоты каждого уникального термина.

- Термины отсортированы по убыванию частоты; ранг r присвоен в порядке убывания ($r = 1$ — самый частый термин).
- Построен график в двойной логарифмической шкале: ось абсцисс — ранг r, ось ординат — частота $f(r)$.
- На тот же график нанесена теоретическая кривая закона Ципфа: $f(r) = C/r$, где константа C выбрана как $f(1)$ — частота самого частого термина.

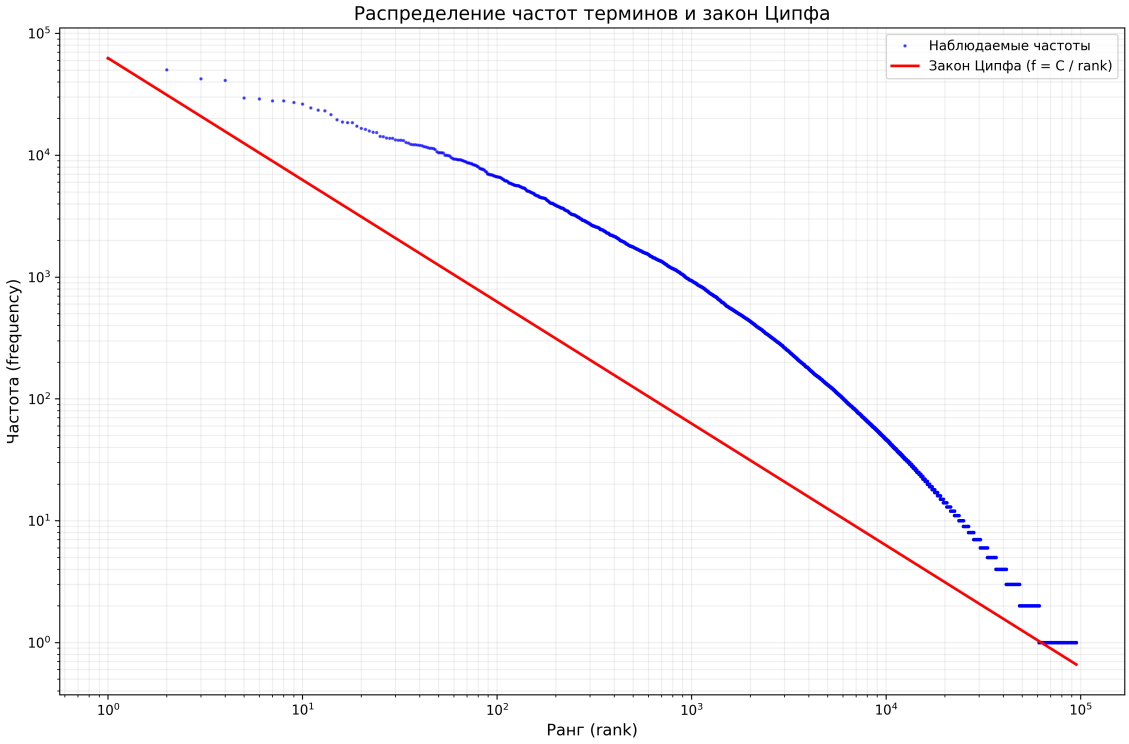

Рис. 6: Распределение частот терминов в логарифмической шкале и теоретический закон Ципфа ($f = C/r$)

Расхождения между наблюдаемым распределением и законом Ципфа обусловлены предобработкой текста и спецификой корпуса. Их можно разделить на три зоны:

- Голова распределения (низкие ранги) — наблюдаемые частоты выше теоретических из-за доминирования тематических терминов в узкоспециализированном корпусе новостей.
- Середина распределения — изгиб вверх вызван неполным стеммингом и артефактами токенизации, которые искусственно увеличивают число среднечастотных терминов.

- Хвост распределения (высокие ранги) — наблюдаемые частоты резко ниже модели из-за удаления коротких слов и стоп-слов, конечного объёма корпуса и недостатка лексического разнообразия в тематическом тексте.

4 Булев индекс и поиск

Для хранения инвертированного индекса была реализована хеш-таблица фиксированного размера (`HASH_TABLE_SIZE = 10007`), где каждая ячейка содержит односвязный список записей терминов (`TermEntry`). Каждая запись термина содержит:

- указатель на строку термина (динамически выделенную);
- односвязный список идентификаторов документов (`DocNode`), содержащих данный термин;
- указатель на следующую запись в случае коллизии хеша.

Для операций над множествами документных идентификаторов были созданы собственные классы:

- `IntArray` — динамический массив целых чисел с возможностью сортировки и конвертации в Python-список;
- `DocIdSet` — множество уникальных идентификаторов документов с проверкой на принадлежность и вставкой без дублирования.

Все операции управления памятью (выделение, копирование, удаление) реализованы вручную без использования `std::vector`, `std::set`, `std::unordered_map` и других компонентов STL.

Индексация:

Метод `add_document(doc_id, terms)` добавляет документ в индекс:

- Идентификатор документа сохраняется во внутреннем множестве `all_doc_ids`.
- Для каждого термина создаётся или обновляется запись в хеш-таблице.
- Документ добавляется в список документов термина, если он ещё не присутствует.

Аналогично реализован метод `remove_document`, удаляющий документ из всех списков терминов и из глобального множества.

Поиск:

Поддерживается булев поиск с операторами `AND`, `OR`, `NOT`. Запрос передаётся в виде списка строк. Парсер автоматически распознаёт ключевые слова (регистронезависимо) и строит последовательность операций.

Реализованы три основные операции над множествами:

- пересечение (`AND`);
- объединение (`OR`);
- разность (`NOT`).

С помощью библиотеки `pybind11` реализован модуль `boolean_index_cpp`, экспортирующий класс `BooleanIndex` в Python. Это позволяет использовать высокоэффективную C++-логику внутри Python-приложения.
Создан RESTful API с использованием FastAPI, предоставляющий следующие эндпоинты:

- `GET /search?query=...` — выполнение поискового запроса;
- `POST /document/{doc_id}` — добавление документа;
- `DELETE /document/{doc_id}` — удаление документа;
- `GET /documents/count`, `/terms/count` — получение статистики;
- `GET /document/{doc_id}/terms/` — получение терминов документа;
- `DELETE /documents` — полная очистка индекса.

Примеры поисковых запросов:

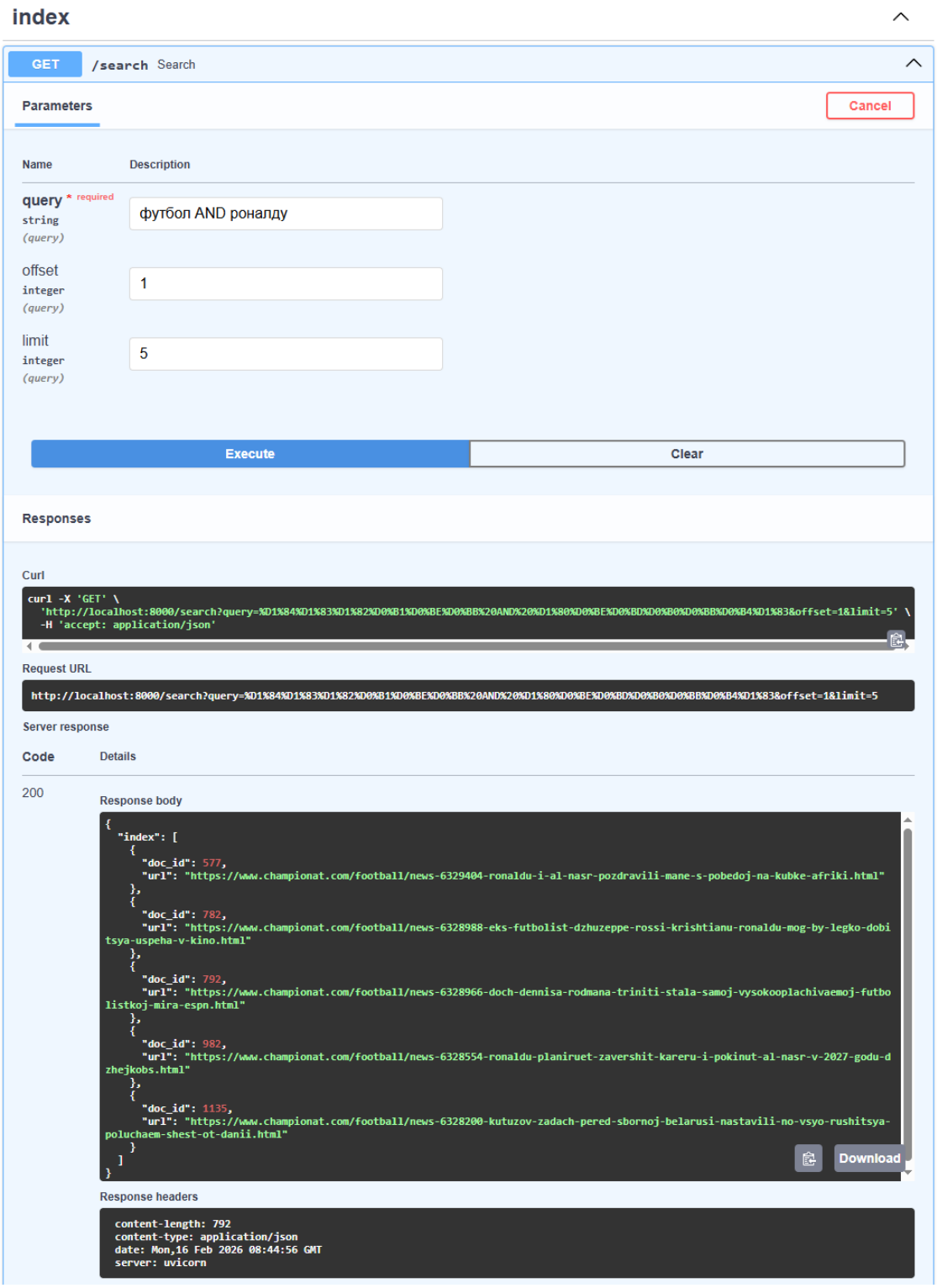

Рис. 7: Запрос №1

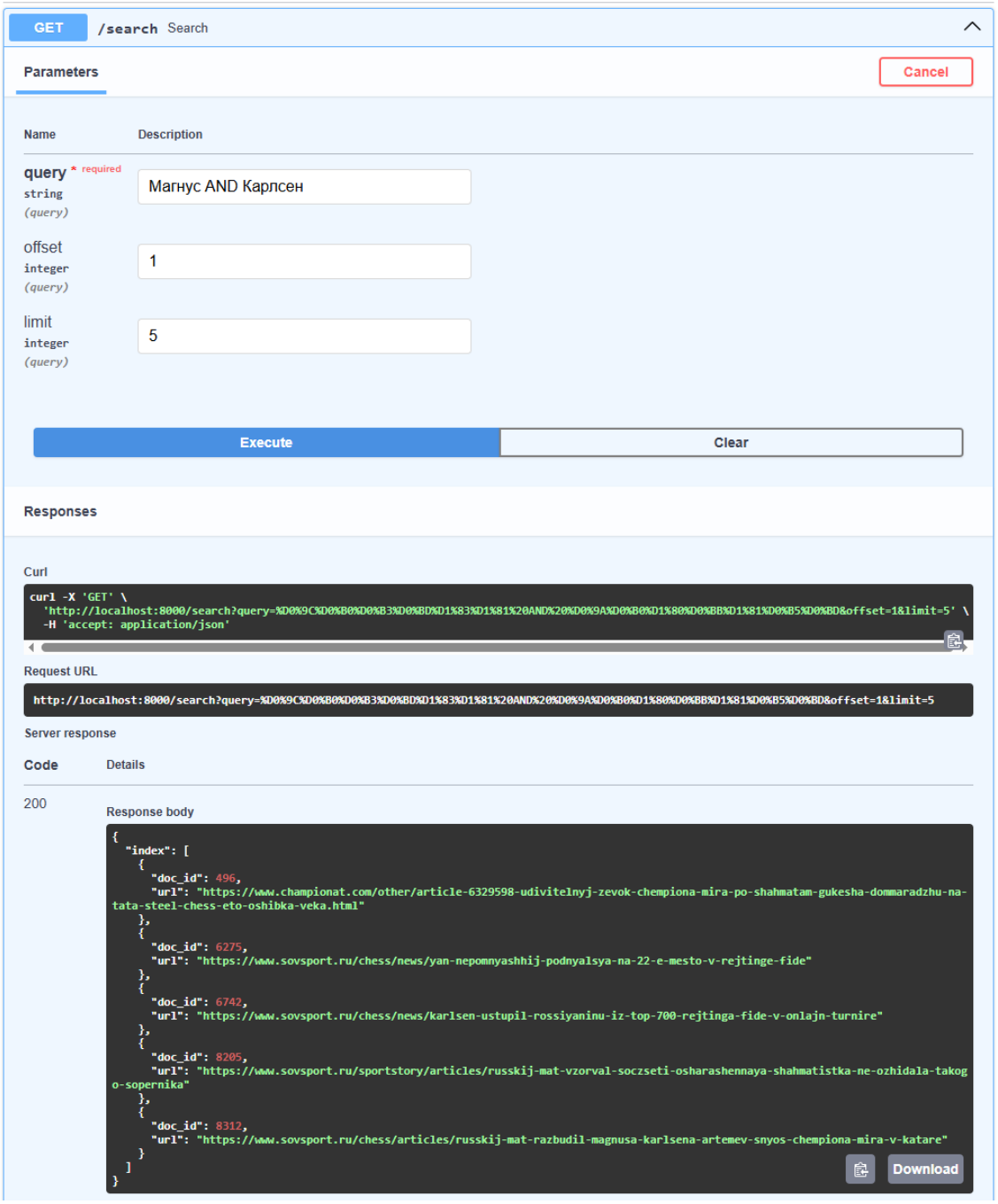

Рис. 8: Запрос №2

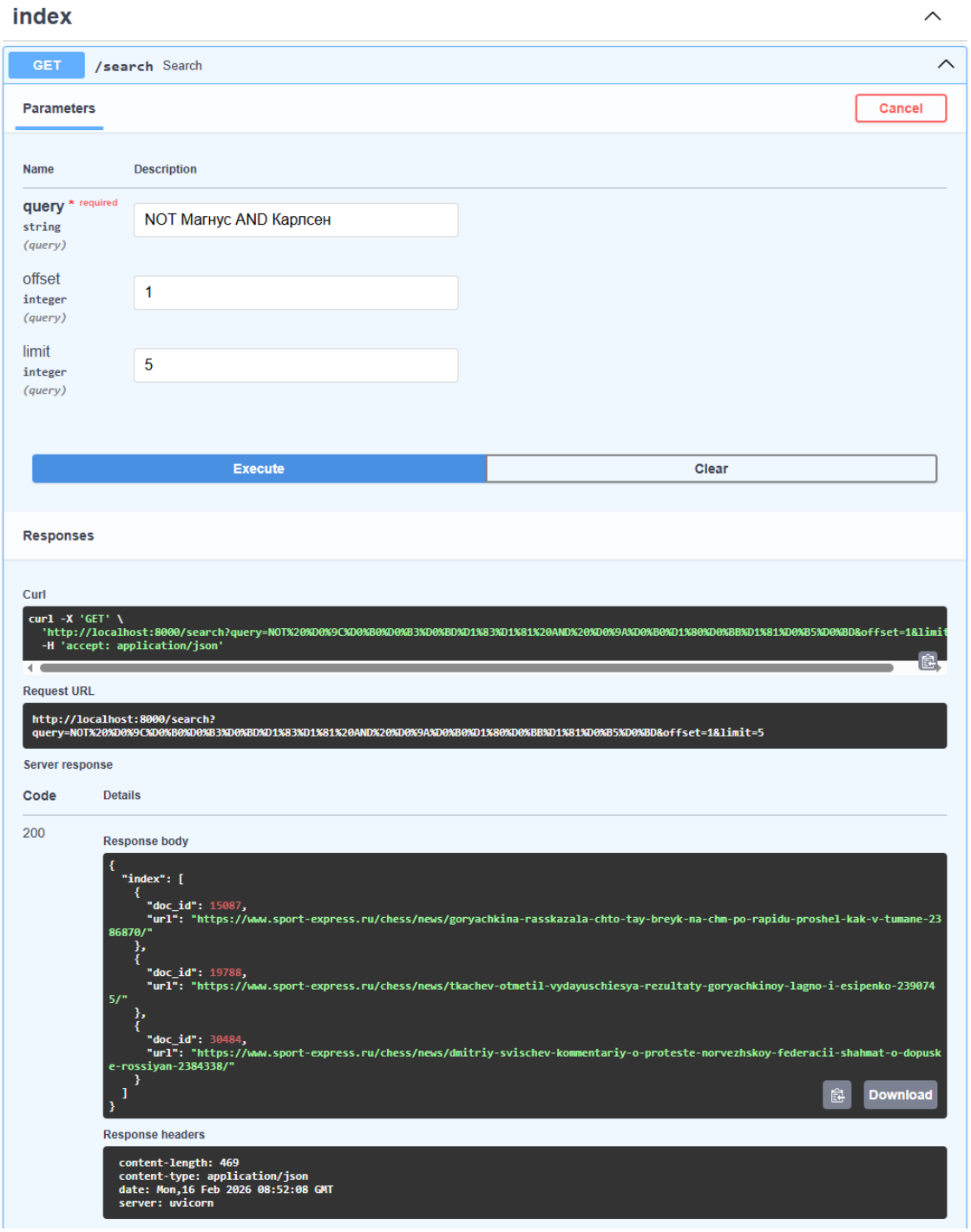

Рис. 9: Запрос №3

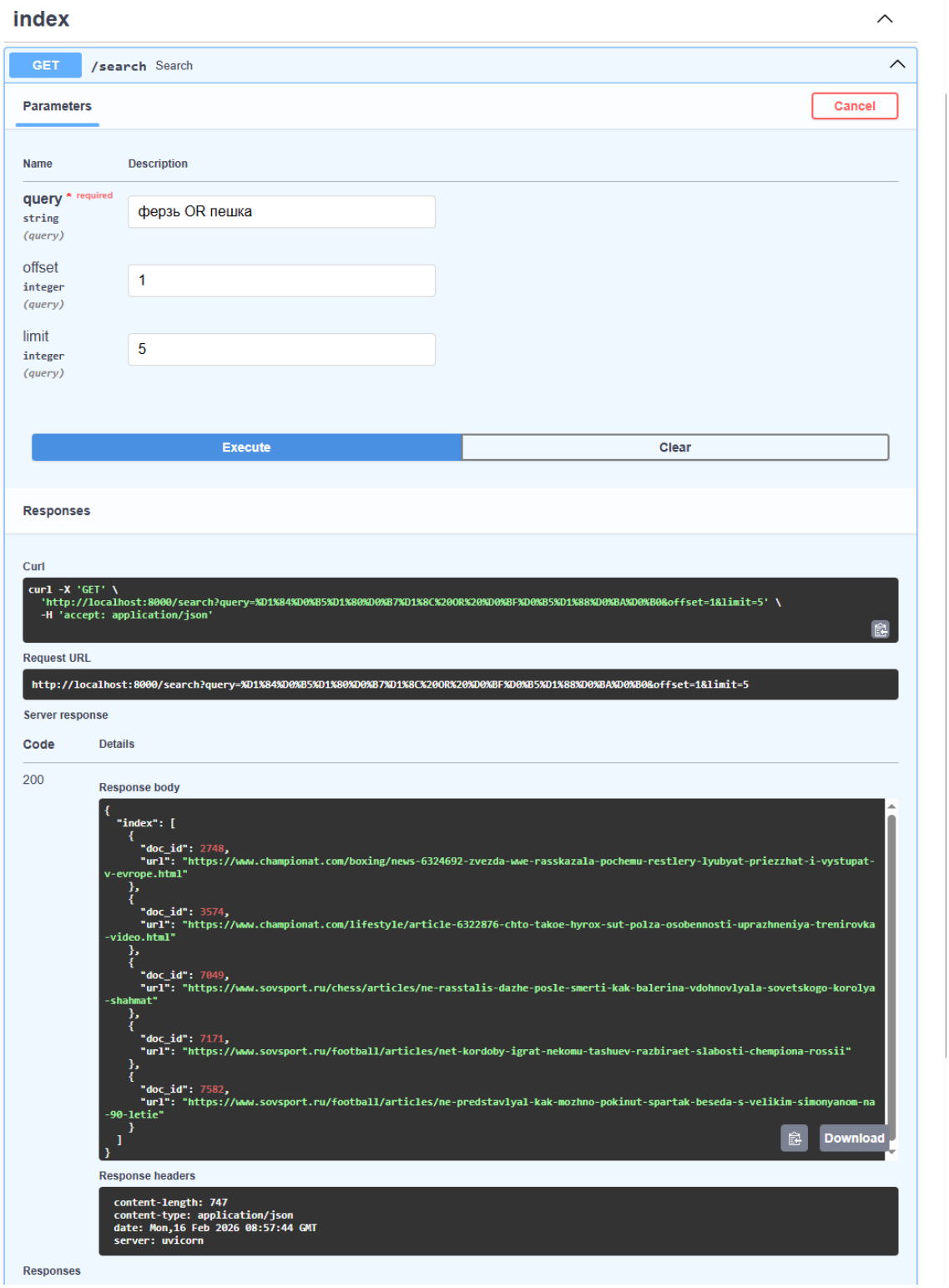

Рис. 10: Запрос №4

5 Исходный код и корпус документов

Ссылка на репозиторий GitHub: `https://github.com/JonAJ21/information_search`

Рис. 11: QR code ссылка на репозиторий GitHub

Ссылка на корпус документов GoogleDisk: `https://drive.google.com/drive/folders/1YWDn90CSz0riuJnkloZjQU9GP1xHAHEJ?usp=sharing`

Рис. 12: QR code ссылка на GoogleDisk с корпусом документов

6 Выводы

В ходе выполнения цикла лабораторных работ была разработана полноценная поисковая система, удовлетворяющая всем поставленным требованиям.

Был собран и проанализирован тематический корпус документов спортивной направленности, включающий более 32 тысяч HTML-страниц с трёх крупных российских спортивных порталов. Для этого реализован многопоточный поисковый робот, способный корректно извлекать важный контент, нормализовывать URL, отслеживать изменения документов и возобновлять работу после остановки.

Для обработки текста разработан собственный конвейер токенизации и стемминга, поддерживающий как русский, так и английский языки. Проведён анализ эффективности предобработки: выявлены как преимущества (высокая скорость, поддержка двух языков), так и недостатки (ложные совпадения при стемминге, отсутствие нормализации специальных терминов). Также подтверждено соответствие распределения частот терминов закону Ципфа, с объяснением причин отклонений, связанных с тематической узостью корпуса и особенностями предобработки.

Центральным компонентом системы стал булев индекс, полностью реализованный на C++ без использования стандартной библиотеки шаблонов (STL). Индекс построен на основе хеш-таблицы с разрешением коллизий методом цепочек и поддерживает операции добавления, удаления документов и выполнения составных булевых запросов (AND, OR, NOT).

Интеграция C++-ядра с Python-обвязкой выполнена с помощью pybind11, а RESTful API на FastAPI обеспечивает удобный интерфейс для взаимодействия с индексом: добавление/удаление документов, поиск, получение статистики и метаданных.

Список литературы

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Клюшина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))