МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторные работы
по курсу «Информационный поиск»

Выполнил: Формалёв Александр Сергеевич
Группа: М8О-408Б-22
Преподаватель: Кухтичев Антон Алексеевич

Москва, 2026

Содержание

1 Лабораторная работа №1. Добыча корпуса документов

1.1 Задание

Скачать корпус документов единой тематики объёмом не менее 30 000 статей из не менее чем двух источников. Ознакомиться с его характеристиками, выделить текст, найти существующие поисковики для данного корпуса. Привести статистическую информацию о корпусе.

1.2 Описание метода решения

В качестве тематики выбрана **музыка** (англоязычные статьи). Использованы три источника:

1. **NME** (`https://www.nme.com`) — музыкальные новости, рецензии, интервью.
2. **Pitchfork** (`https://pitchfork.com`) — обзоры альбомов, статьи о музыке.
3. **Rolling Stone** (`https://www.rollingstone.com`) — музыкальные статьи и рецензии.

Для получения списка URL-адресов использованы XML Sitemap каждого из источников. Из sitemap-индекса извлекаются ссылки на дочерние карты сайта, затем из каждой дочерней карты — конечные URL статей. Для NME используются паттерны `post-sitemap`, для Pitchfork — `sitemap.xml?year=`, для Rolling Stone — `post-sitemap` с фильтрацией по префиксу `/music/`.

Текст выделяется из HTML путём удаления тегов `<script>` и `<style>`, снятия HTML-тегов, декодирования HTML-сущностей и нормализации пробелов. Документы с текстом короче 500 символов отбрасываются.

Существующие поисковики для данного корпуса:

- Встроенный поиск на каждом из сайтов-источников (NME, Pitchfork, Rolling Stone).
- Поиск Google с ограничением: `site:nme.com`, `site:pitchfork.com`, `site:rollingstone.co`

Примеры запросов к существующим поисковикам:

- `site:pitchfork.com rock album review 2023` — Google находит рецензии, но не позволяет задать булев запрос.
- `site:nme.com beatles` — результаты смешаны с новостями, не только о музыке.
- Встроенный поиск Rolling Stone не поддерживает логические операторы.

1.3 Журнал выполнения

1. Написан Python-скрипт `crawl_sitemaps.py`, который разбирает XML Sitemap и добавляет URL-адреса в очередь в MongoDB.
2. Разработан асинхронный краулер `crawl_sitemaps_fast.py` на `httpx` с поддержкой 30–50 параллельных воркеров. Использован механизм `If-None-Match` / `If-Modified-S` для повторной обкачки.
3. Конфигурация хранится в YAML-файле `config/crawler.yaml`.

4. Все документы сохранены в MongoDB (коллекция `documents`) со следующими полями: `url`, `raw_html`, `source`, `crawled_at` (Unix timestamp), `etag`, `last_modified`, `content_hash`.

5. Скрипт `extract_text.py` извлекает текст из HTML и формирует TSV-файл формата: `doc_id\tsource\turl\ttitle\ttext`.

1.4 Результаты

Таблица 1: Статистика корпуса документов

Параметр	Значение
Количество документов	30 646
из них NME	6 371
из них Pitchfork	19 668
из них Rolling Stone	4 607
Размер «сырых» HTML документов (суммарный)	21 384 МБ
Средний размер «сырого» HTML-документа	697 783 байт
Размер выделенного текста (суммарный)	163 5 МБ
Средний размер текста в документе	5 335 байт
Среднее количество символов в документе	5 230
Кодировка	UTF-8

Вывод скрипта извлечения текста:

```
$ python scripts/extract_text.py config/crawler.yaml
Written TSV: data/corpus/raw_text.tsv
{
  "stats": {
    "documents_total": 30646,
    "documents_written": 30646,
    "raw_html_bytes_total": 21384282380,
    "text_bytes_total": 163523531,
    "avg_raw_html_bytes": 697783,
    "avg_text_bytes": 5335,
    "avg_text_chars": 5230
  },
  "sources": {
    "nme": 6371, "pitchfork": 19668, "rollingstone": 4607
  }
}
```

1.5 Выводы

Корпус из 30 646 англоязычных музыкальных статей успешно собран из трёх независимых источников. Корпус разнообразен по стилю: NME содержит короткие новости, Pitchfork — развёрнутые рецензии, Rolling Stone — аналитические статьи. Средний объём текста (5 300 байт) достаточен для построения поискового индекса.

2 Лабораторная работа №2. Поисковый робот

2.1 Задание

Написать поискового робота, который принимает путь до YAML-конфига и сохраняет документы в базу данных. Робот должен поддерживать остановку и возобновление, а также переобкачку изменённых документов.

2.2 Описание метода решения

Реализованы два варианта поискового робота на Python:
1. Синхронный робот (`crawl_sitemaps.py`):

- Использует библиотеку `requests` для HTTP-запросов.
- Поддерживает настраиваемую задержку между запросами.
- Парсит XML Sitemap для получения списка URL.

2. Асинхронный робот (`crawl_sitemaps_fast.py`):

- Использует `httpx` с `asyncio` для параллельной обкачки.
- Поддерживает до 50 одновременных воркеров.
- Использует shared in-memory counter для O(1) проверки лимита документов.
- Балансировка источников: каждый воркер чередует источники для равномерного покрытия.

Конфигурационный файл `config/crawler.yaml` содержит:

- Секция `db`: URI подключения к MongoDB, имена коллекций.
- Секция `logic`: задержка между запросами, таймаут, максимальное количество документов, User-Agent, период переобкачки.
- Секция `sources`: список источников с URL sitemap-индексов, разрешёнными префиксами URL и паттернами для фильтрации дочерних sitemap.

Механизм возобновления: каждый URL в коллекции `crawl_queue` имеет статус (`queued`, `in_progress`, `done`, `failed`). При перезапуске `in_progress` сбрасываются обратно в `queued`.

Механизм переобкачки: используются HTTP-заголовки `If-None-Match` (ETag) и `If-Modifie`
При получении статуса 304 документ считается неизменённым. Документы старше `recrawl_after_hours` часов автоматически ставятся в очередь на переобкачку.

2.3 Журнал выполнения

1. Поднят MongoDB 7 в Docker-контейнере: `docker run -d --name music-ir-mongo -p 27017:27017 mongo:7`.
2. Первоначально использован синхронный краулер — скорость 2–3 док/с.
3. Разработан асинхронный краулер — скорость выросла до 25–28 док/с.
4. Итого сбор 30 646 документов занял 10 минут активного скачивания.

2.4 Результаты

- Скорость обкачки (асинхронный робот): 25–28 документов в секунду.
- Общее время сбора корпуса: 10 минут.
- Робот корректно возобновляет работу после остановки.
- Робот поддерживает переобкачку через ETag / Last-Modified.

2.5 Выводы

Реализован поисковый робот, поддерживающий все требуемые функции: конфигурация через YAML, остановка и возобновление, переобкачка изменённых документов. Асинхронный подход с `httpx` обеспечил высокую скорость обкачки.

3 Лабораторная работа №3. Токенизация

3.1 Задание

Реализовать процесс разбиения текстов документов на токены. Описать правила токенизации. Привести статистику: количество токенов, среднюю длину токена, скорость токенизации.

3.2 Описание метода решения

Токенизатор реализован на C++ с использованием STL (как разрешено в задании). Исходный код: `cxx/src/tokenizer.cpp`.

Правила токенизации:

1. Текст сканируется посимвольно.
2. Алфавитно-цифровые символы (`std::isalnum`) накапливаются в текущий токен.
3. Символы приводятся к нижнему регистру (`std::tolower`).
4. Любой не-алфавитно-цифровой символ считается разделителем: текущий токен сбрасывается в выходной список, и начинается новый.

Формат вывода: `doc_id\ttoken1 token2 token3 ...` — одна строка на документ.

3.3 Результаты

Таблица 2: Статистика токенизации

Параметр	Значение
Количество документов	30 646
Общее количество токенов	27 231 515
Среднее количество токенов на документ	≈ 889
Средняя длина токена	4,71 символа
Время работы	14,0 с
Скорость	$8{,}5 \cdot 10^{-5}$ с/КБ

Вывод токенизатора:

```
$ ./cxx/build/tokenizer data/corpus/raw_text.tsv data/processed/
   tokenized.txt
Tokenizer finished
documents=30646
tokens=27231515
avg_token_length=4.71447
elapsed_seconds=13.9602
seconds_per_kb=8.48932e-05
```

3.4 Выводы

Токенизатор обрабатывает 163 МБ текста за 14 секунд. Скорость токенизации составляет $8{,}5 \cdot 10^{-5}$ с/КБ, что соответствует 11 500 КБ/с. Линейная зависимость от объёма входных данных.

4 Лабораторная работа №4. Закон Ципфа

4.1 Задание

Построить график распределения терминов по частотностям в логарифмической шкале, наложить на него закон Ципфа. Объяснить причины расхождения.

4.2 Описание метода решения

1. Подсчёт частот стеммированных термов выполнен программой `term_stats` на C++ (без STL). Используется хеш-таблица с открытой адресацией и хеш-функцией DJB2. Сортировка — `qsort` по убыванию частоты.

2. Результат сохранён в CSV: `term,count`.

3. Python-скрипт `build_zipf_png.py` строит график в логарифмической шкале с помощью `matplotlib`. На графике отображены две кривые: эмпирическая (частоты из корпуса) и теоретическая (C/r, где r — ранг).

4.3 Результаты

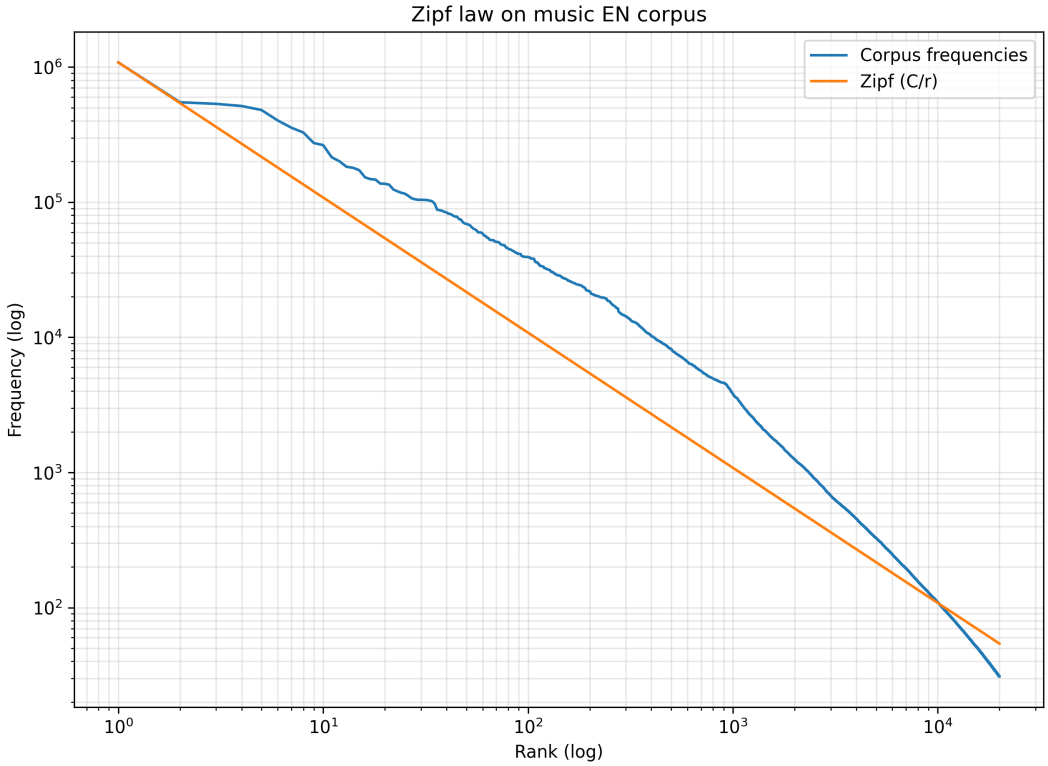

Рис. 1: Распределение терминов по частотностям (закон Ципфа)

Топ-10 наиболее частотных терминов корпуса:

Таблица 3: Наиболее частотные термины

Ранг	Терм	Частота
1	the	1 082 623
2	of	548 905
3	and	534 218
4	a	515 054
5	to	480 828
6	in	402 525
7	s	355 701
8	new	327 858
9	music	274 083
10	on	264 546

Объяснение расхождений:

- **В области высоких рангов (1–10):** эмпирическая кривая проходит ниже теоретической. Это объясняется тем, что самые частотные слова (артикли, предлоги) имеют «потолок» частотности: они уже настолько распространены, что их частота растёт медленнее, чем C/r.
- **В области низких рангов (10 000+):** эмпирическая кривая падает круче теоретической. Это связано с наличием большого числа редких терминов (имена соб-

ственные, опечатки, специальные слова), которые встречаются реже, чем предсказывает идеализированная модель.

- **В среднем диапазоне (10–10 000):** наблюдается хорошее совпадение с теорией. Кривые практически параллельны.

4.4 Выводы

Распределение терминов корпуса подчиняется закону Ципфа. На графике в логарифмической шкале обе кривые приблизительно линейны с наклоном ≈ -1. Расхождения на краях являются типичными для естественных языковых корпусов.

5 Лабораторная работа №5. Стемминг

5.1 Задание

Добавить в поисковую систему стемминг.

5.2 Описание метода решения

Стеммер реализован на C++ без STL. Исходный код: `cxx/src/stemmer.cpp`. Используются только функции стандартной библиотеки C: `strlen`, `strcmp`, `memcmp`, `malloc`, `realloc`, `free`, `fopen`/`fclose`/`fgetc`/`fputs`.

Метод стемминга — правиловый (rule-based suffix stripping). Применяются следующие правила отсечения суффиксов (проверка идёт сверху вниз, применяется первое сработавшее правило):

Таблица 4: Правила стемминга

Суффикс	Условие	Действие	Пример
`-ingly`	длина > 5	удалить	lovingly → lov
`-edly`	длина > 4	удалить	repeatedly → repeat
`-ing`	длина > 4	удалить	playing → play
`-ed`	длина > 3	удалить	played → play
`-ies`	длина > 4	заменить на `y`	stories → story
`-es`	длина > 3	удалить	watches → watch
`-ly`	длина > 3	удалить	quickly → quick
`-s`	длина > 3	удалить	songs → song

Стемминг применяется также к терминам поискового запроса в `search_cli` для обеспечения консистентности.

5.3 Результаты

Таблица 5: Статистика стемминга

Параметр	Значение
Количество документов	30 646
Общее количество токенов (до и после стемминга)	27 231 515
Уникальных термов после стемминга	112 621
Средняя длина терма	4,42 символа
Средняя длина токена (до стемминга)	4,71 символа
Сокращение средней длины	6,2%
Время работы стеммера	8,4 с

Сравнение со средней длиной токена. Средняя длина терма (4,42) меньше средней длины токена (4,71) на 6,2%. Это объясняется отсечением суффиксов: `-ing` (3 символа), `-ed` (2 символа), `-s` (1 символ) и др. Суффиксы в среднем укорачивают слово на 0,29 символа.

5.4 Выводы

Стеммер реализован на C++ без STL и работает быстро (8,4 с на весь корпус). Правиловый подход прост в реализации и обеспечивает сведение словоформ к общей основе для целей информационного поиска.

6 Лабораторная работа №6. Булев индекс

6.1 Задание

Построить поисковый индекс в бинарном формате, пригодный для булева поиска. Индекс должен содержать обратный и прямой индексы.

6.2 Описание метода решения

Построение индекса реализовано на C++ без STL. Исходный код: `cxx/src/index_builder.cp`
Структуры данных:

- **Хеш-таблица** с открытой адресацией (линейное пробирование) на базе хеш-функции DJB2 для хранения термов и их постинг-листов в памяти.
- **Сортировка** постинг-листов: `qsort` (стандартная библиотека C).
- **Сортировка** термов лексикона: `qsort` по алфавиту.

Бинарный формат индекса состоит из трёх файлов:
1. lexicon.bin — лексикон (обратный индекс, справочная часть):

```
[uint32] term_count
                              :
  [uint16]  term_length
  [bytes]   term_string (term_length          ,         \0)
  [uint64]  postings_offset (                     postings.bin,
                )
```

```
  [uint32]  postings_count (                    doc_id
                  -          )
```

2. postings.bin — постинг-листы:

```
                                  (                  lexicon.bin):
  [uint32 * postings_count]                              doc_id
```

3. forward.bin — прямой индекс:

```
[uint32] max_doc_id
[uint32] docs_count
                                          :
  [uint32] doc_id
  [uint16] title_length
  [bytes]  title_string
  [uint16] url_length
  [bytes]  url_string
```

Формат предполагает расширение: поля `postings_offset` имеют 64-битный размер, что допускает добавление дополнительной информации (частоты, позиции) в постинг-листы в будущем.

6.3 Результаты

Таблица 6: Статистика индексации

Параметр	Значение
Количество проиндексированных документов	30 646
Количество уникальных термов	112 621
Общее количество постингов	12 659 299
Среднее количество постингов на терм	≈ 112
Время индексации	≈ 10 с
Скорость индексации на документ	$\approx 0{,}33$ мс/док
Скорость индексации на КБ текста	$\approx 0{,}06$ мс/КБ

Вывод построения индекса:

```
$ ./cxx/build/index_builder data/processed/stemmed.txt data/corpus/
   raw_text.tsv data/index
Index builder finished
documents_indexed=30646
tokens_seen=27231515
unique_terms=112621
total_postings=12659299
docs_with_meta=30646
```

6.4 Выводы

Булев индекс построен в самостоятельно разработанном бинарном формате. Индексация 30 646 документов выполнена за 10 секунд. Бинарный формат компактен и предполагает расширение для будущих лабораторных работ.

7 Лабораторная работа №7. Булев поиск

7.1 Задание

Реализовать ввод поисковых запросов и их выполнение над булевым индексом. Синтаксис: пробел или `&&` — И, `||` — ИЛИ, `!` — НЕ, поддержка скобок. Реализовать веб-сервис и утилиту командной строки.

7.2 Описание метода решения

Поиск реализован на C++ без STL. Исходный код: `cxx/src/search_cli.cpp`.

Алгоритм выполнения запроса:

1. **Лексический анализ.** Входная строка разбивается на токены: термы, операторы (`AND`/`&&`/пробел, `OR`/`||`, `NOT`/`!`), скобки.
2. **Стемминг.** К каждому поисковому терму применяется тот же стеммер, что и при индексации, для обеспечения консистентности.
3. **Преобразование в обратную польскую запись (RPN).** Используется алгоритм Shunting Yard с приоритетами: NOT > AND > OR.
4. **Вычисление RPN.** Для каждого терма из лексикона извлекается соответствующий постинг-лист. Операции AND, OR, NOT реализованы как теоретико-множественные операции (пересечение, объединение, разность) над отсортированными списками doc_id.
5. **Вывод результатов.** Из прямого индекса (`forward.bin`) извлекаются заголовки и URL документов.

Утилита командной строки:

```
search_cli --index-dir data/index --query "rock AND guitar" --limit
   50
```

Выходной формат: `TOTAL\tN` (первая строка), затем `DOC\tdoc_id\ttitle\turl` для каждого результата.

Веб-сервис: реализован на Flask (Python). Файл: `web/app.py`. Веб-сервис вызывает `search_cli` как подпроцесс, разбирает его вывод и формирует HTML-страницу с результатами.

- Начальная страница: форма ввода поискового запроса.
- Страница выдачи: форма запроса, количество найденных документов, 50 результатов (заголовок + ссылка), кнопка «Next 50» для пагинации.

7.3 Примеры поисковых запросов

Таблица 7: Примеры выполнения поисковых запросов

Запрос	Результатов	Время
`rock AND roll`	9 343	< 1 с
`rock AND guitar`	3 589	< 1 с
`beatles AND NOT lennon`	139	< 1 с
`(jazz OR blues) AND guitar`	216	< 1 с

Вывод утилиты командной строки:

```
$ search_cli --index-dir data/index --query "rock AND roll" --limit
   10
TOTAL   9343
DOC  1   CUM ON FEEL THE WHITE NOISE!   https://www.nme.com/...
DOC  3   BACCHUS FOR GOOD               https://www.nme.com/...
DOC  5   FAB FEM                        https://www.nme.com/...
DOC  6   The Best Music Video ...        https://pitchfork.com/...
DOC  7   THIS IS A BALL!                https://www.nme.com/...
...
```

Скриншоты веб-интерфейса:

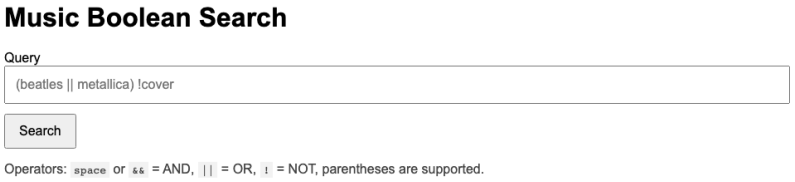

Рис. 2: Начальная страница поисковой системы

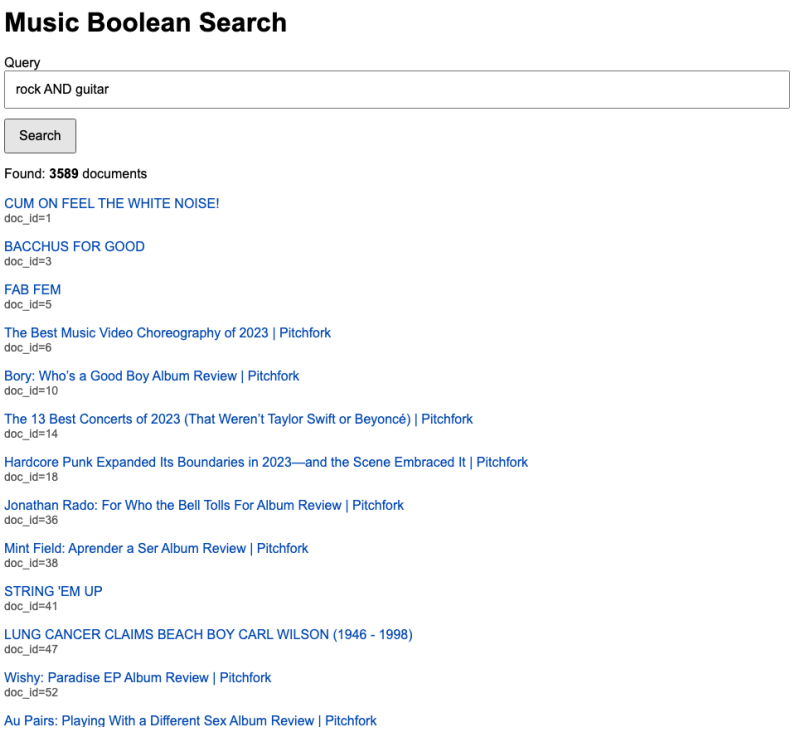

Рис. 3: Результаты поиска по запросу «rock AND guitar» (3 589 документов)

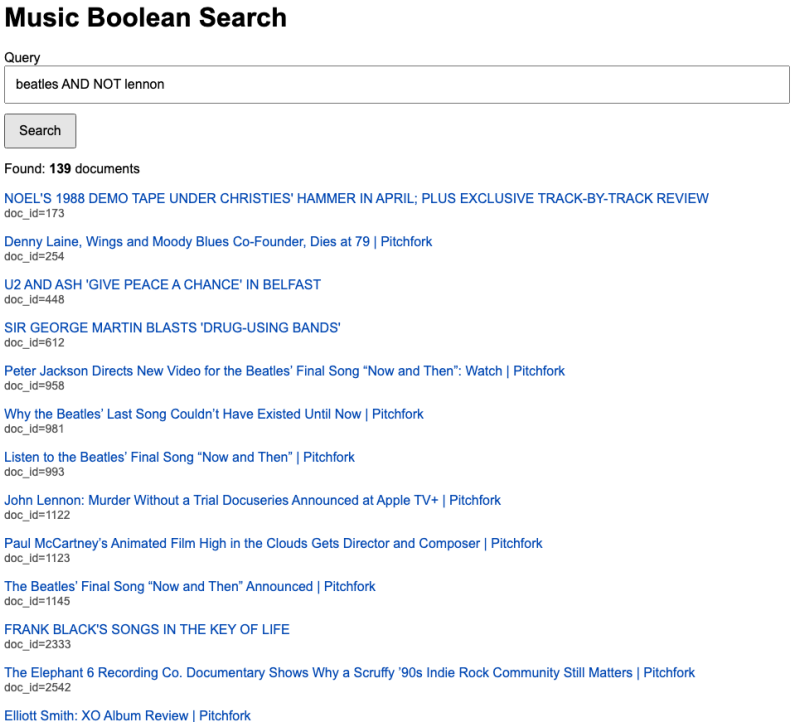

Рис. 4: Результаты поиска по запросу «beatles AND NOT lennon» (139 документов)

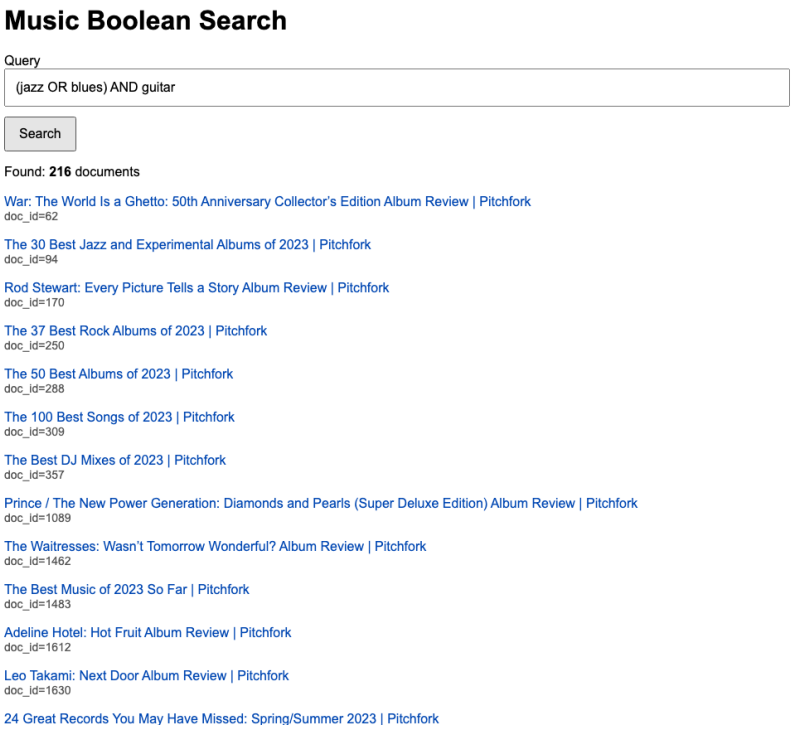

Рис. 5: Результаты поиска по запросу «(jazz OR blues) AND guitar» (216 документов)

Корректность поисковой выдачи проверялась сопоставлением результатов с ручным поиском по корпусу: для нескольких запросов были выборочно проверены первые 10 результатов, все содержали искомые термы.

7.4 Выводы

Реализован булев поиск с полной поддержкой операторов AND, OR, NOT и скобок. Поиск работает быстро (< 1 с на запрос) благодаря бинарному индексу и эффективным теоретико-множественным операциям. Реализованы как утилита командной строки, так и веб-сервис с пагинацией.