Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Отчет по лабораторным работам по курсу "Информационный поиск"

Студент: Е. А. Медведев
Преподаватель: А. А. Кухтичев
Группа: М8О-401Б-22

Дата: 26.12.2025
Оценка:
Подпись:

Москва, 2025

Содержание

1 Добыча корпуса документов

1.1 Выбор источников и цель корпуса

Корпус документов формируется поисковым роботом из открытых англоязычных источников по тематике *Earth Sciences / Geology*:

- **Wikipedia (en)**: стартовая категория `Category:Earth_sciences` и переходы по страницам категорий и статей;
- **USGS Publications**: поисковая выдача `pubs.usgs.gov` по запросу `geology` с пагинацией;
- **NASA Earth Observatory**: тематический раздел `/topic/geology` и статьи/очерки;
- **ScienceDaily**: раздел новостей `/news/earth_climate/geology/` и страницы релизов.

Цель сбора — получить большой, тематически единый, но стилистически разнообразный корпус (энциклопедические статьи, научные публикации/отчёты, научно-популярные материалы, новости), пригодный для последующих этапов анализа (токенизация, стемминг, закон Ципфа) и построения булевого индекса.

1.2 Конфигурация сбора

Сбор настраивается YAML-конфигурацией `application.yaml`. Основные параметры:

- **MongoDB**: `uri=mongodb://localhost:27017`, база `corpus`, коллекции `documents` и `frontier`;
- **Сетевые ограничения**: `connectTimeoutSec=10`, `readTimeoutSec=20`;
- **Ограничение нагрузки**: `requestDelayMs=5000`;
- **Повторы**: `maxRetries=2`;
- **User-Agent**: строка браузера (Chrome/120) для корректной совместимости;
- **Опции обхода**: `enableRecrawl=false`, `stopWhenTargetsReached=false`;
- **Источники**: массив `sources[]` со списком seed URL, разрешённых хостов и регулярных выражений для допустимых/запрещённых ссылок.

1.3 Формирование списка URL (seed) и правила обхода

Для каждого источника задаются стартовые URL (seed) и правила фильтрации:

Wikipedia (категории и статьи). Разрешённые страницы категорий задаются паттернами вида:

$$\hat{\texttt{h}}\texttt{ttps://en}\dot{\texttt{w}}\texttt{ikipedia}\dot{\texttt{o}}\texttt{rg/wiki/Category:.*\$}$$

Разрешённые статьи отбираются регулярным выражением:

$\hat{\texttt{h}}\texttt{ttps://en}\dot{\texttt{w}}\texttt{ikipedia}\dot{\texttt{o}}\texttt{rg/wiki/[}\dot{:}\texttt{\#?]} + \$\hat{\texttt{h}}\texttt{ttps://en}\dot{\texttt{w}}\texttt{ikipedia}\dot{\texttt{o}}\texttt{rg/wiki/[}\dot{:}\texttt{\#?]} + \$\hat{\texttt{h}}\texttt{ttps://en}\dot{\texttt{w}}\texttt{ikipedia}\dot{\texttt{o}}\texttt{rg/}$

Запрещаются служебные страницы (Help, File, Template, Special, Talk, User, Wikipedia), а также страницы с параметрами (edit/history/diff/oldid) и медиа-файлы.

USGS (поиск + карточки публикаций + DOI). Используется seed `https://pubs.usgs.gov/sea` и пагинация:

`https://pubs.usgs.gov/search?q=geology&page={page}`

для диапазона `1..250`. Разрешаются страницы поиска и карточки публикаций (`/publication/<id>`), а в качестве статей — ссылки на отчёты/серии (fs, sir, ofr, pp, circ, bulletin и т.д.) и DOI (через `doi.org`).

NASA Earth Observatory. Seed `https://earthobservatory.nasa.gov/topic/geology`. Допускаются страницы темы и материалы форматов `/images/<id>/...` и `/features/...`. Запрещаются медиа-файлы (jpg/png/pdf/mp4).

ScienceDaily. Seed `https://www.sciencedaily.com/news/earth_climate/geology/`. В качестве статей допускаются страницы релизов:

$\hat{\text{h}}\text{ttps://wwẇsciencedailẏcom/releases/ṫ/ṫ/.*ḣtm\$}$

1.4 Сохранение в MongoDB

Хранилище состоит из двух основных коллекций:

- `frontier` — очередь задач на обработку URL (планирование обхода);
- `documents` — загруженные документы и метаданные.

Концептуально (на уровне требований к данным) документ в `documents` содержит:

- `url`, `source`, `fetchedAt`;
- `statusCode` и данные об ошибке (если была);
- `etag`, `lastModified` (если предоставляет сервер);
- `contentHash` (контроль изменений);
- `rawContent` (HTML/текст, в зависимости от обработчика).

Запись frontier-задачи в `frontier` содержит:

- `url`, `source`;
- `status` (*pending/in_progress/done/failed*);
- `attempts` (число попыток);
- `nextFetchAt` (время, когда задачу можно брать снова);
- служебные timestamps.

1.5 Связь краулера и офлайнового пайплайна

После накопления корпуса в MongoDB выполняется выгрузка в файловый формат, используемый последующими утилитами (токенизация, индексирование). Офлайновая часть работает с текстовым файлом `data/corpus.txt` и далее строит статистики и индекс:

`corpus.txt` → `tokenizer` → `zipf_analyzer.py` → `index_builder` → `search`

Разделение на онлайн-сбор и офлайн-анализ упрощает повторяемость экспериментов: можно фиксировать срез корпуса и многократно прогонять анализ с разными параметрами токенизации/поиска.

2 Поисковый робот

2.1 Общая схема

Поисковый робот реализован на Kotlin с использованием Spring Boot и MongoDB. Внутренняя логика построена как конвейер (pipeline) из двух потоков работ:

1. **Discoverer** (обнаружение ссылок): извлекает новые URL со страниц, фильтрует их по правилам источника и добавляет в `frontier`.
2. **Fetcher** (загрузка контента): получает задачи из `frontier`, выполняет HTTP-загрузку и сохраняет документ в `documents`, обновляя статус задачи.

Такое разделение позволяет независимо масштабировать стадии поиска ссылок и скачивания контента и обеспечивает устойчивость при сетевых сбоях.

2.2 Компоненты и структура проекта

Ключевые компоненты проекта:

- `SearchRobotApplication.kt` — точка входа Spring Boot;
- `robot/CrawlRobot.kt` — основной orchestrator: управляет стадиями discover/fetch, retries, backoff, статистикой;
- `robot/FrontierTaskClaimer.kt` — атомарная выдача задач из очереди (`frontier`);
- `repository/DocumentRepo.kt, FrontierRepo.kt` — слой доступа к MongoDB;
- `net/HttpFetcher.kt` — HTTP-клиент (таймауты, заголовки, повторные попытки);
- `config/RobotConfig.kt` — типизированная конфигурация источников и лимитов;
- `util/` — нормализация URL, хеширование, парсинг HTML (Jsoup) и вспомогательные функции;
- `scheduler/DumpScheduler.kt` — периодические дампы/бэкапы коллекций.

2.3 Нормализация URL и дедупликация

Для предотвращения дублей и корректного фронтира применяется нормализация URL:

- приведение схемы/хоста к нижнему регистру;
- удаление фрагмента (`#...`);
- нормализация относительных ссылок через `urljoin`-аналог (в Kotlin);
- фильтрация по `allowedHosts` и регулярным выражениям `allowPatterns/denyPatterns`.

Дедупликация обеспечивается за счёт уникальности URL (или нормализованного URL) на уровне `frontier` и `documents`.

2.4 Frontier: выдача задач и статусы

Очередь задач хранится в MongoDB (`frontier`). Выдача следующей задачи реализуется компонентом `FrontierTaskClaimer` атомарной операцией "claim": задача выбирается по статусу *pending* и условиям доступности, после чего одним запросом переводится в *in_progress*. Это исключает гонки между потоками и обеспечивает, что один URL не будет скачан двумя воркерами одновременно.

2.5 HTTP-загрузка и уважение к источникам

HTTP-загрузка выполняется компонентом `HttpFetcher` с заданными таймаутами и задержкой `requestDelayMs=5000` между запросами. Это снижает нагрузку на внешние сайты и повышает устойчивость к временным ограничениям со стороны источников.

2.6 Resilience: повторы и backoff

Для задач, завершившихся ошибкой (сетевые таймауты, временные коды HTTP и т.п.), применяется политика повторов:

- число повторов ограничено `maxRetries=2`;
- между попытками используется экспоненциальный backoff (увеличение задержки);
- при исчерпании попыток задача помечается как *failed* и исключается из активного потока обработки.

2.7 Ограничение области обхода (фильтры)

Для каждого источника конфигурация задаёт:

- список допустимых хостов `allowedHosts`;
- регулярные выражения допустимых страниц `pageAllowPatterns`;
- регулярные выражения допустимых статей `articleAllowPatterns`;
- запрещающие паттерны `denyPatterns` (медиа, служебные страницы, редактирование и т.п.).

Фильтрация решает две задачи: (1) удержание обхода в пределах предметной области, (2) защита от "мусорных" URL (медиа, сервисные страницы, параметры редактирования).

2.8 Результат работы робота

Результатом работы робота является накопленная коллекция `documents` в MongoDB с сохранёнными материалами и метаданными, а также заполненная и обработанная очередь `frontier`. Далее данные корпуса выгружаются в `data/corpus.txt` для офлайнового анализа и построения индекса.

3 Стемминг

3.1 Назначение стемминга

Стемминг используется для приведения словоформ к общей основе (стему), что уменьшает размер словаря и повышает полноту поиска. В текущей реализации корпус преимущественно англоязычный, поэтому применяется простой английский rule-based стеммер.

3.2 Реализация

Стеммер реализован в утилите `src/simple_stemmer.cpp` и основан на эвристическом удалении распространённых суффиксов и некоторых дополнительных правилах:

- базовые суффиксы: `ing`, `ed`, `ly`, `es`, `s`, `'s`;
- преобразования `ies/ied` → `y`;
- частные случаи для суффиксов `iness`, `ization`, `ational`, `tional`, `biliti`, `fulness`, `ousness`;
- устранение двойных согласных в конце слова (при условии, что это не гласная).

Подход является эвристическим: стеммер не гарантирует получение леммы, но даёт нормализованную основу, удобную для индексирования.

3.3 Достоинства и ограничения

Достоинства:

- высокая скорость и отсутствие внешних зависимостей;
- заметное сокращение числа уникальных термов в словаре.

Ограничения:

- не учитывает морфологию и часть речи;
- возможны переусечения (overstemming) и недоусечения (understemming);
- правила ориентированы на английский и требуют адаптации при смене языка корпуса.

4 Закон Ципфа

4.1 Суть закона Ципфа для текстов

Для текстов естественного языка характерно распределение частот слов, близкое к закону Ципфа:

$$f(r) \approx \frac{C}{r^s},$$

где r — ранг слова (1 для самого частотного), $f(r)$ — его частота, C — константа, $s \approx 1$.

Проверка закона Ципфа на собранном корпусе позволяет:

- оценить "естественность" распределения частот;
- увидеть долю высокочастотных служебных слов;
- оценить объём длинного хвоста редких термов (разреженность словаря).

4.2 Формирование распределения частот

Распределение частот формируется C++-утилитой `tokenizer` (файл `src/tokenizer.cpp`). Утилита:

- читает корпус из `data/corpus.txt`;
- выделяет токены как последовательности `isalnum` символов;
- приводит токены к нижнему регистру;
- отбрасывает токены длиной менее 2 символов;
- собирает частотную таблицу на собственной хэш-таблице фиксированного размера.

Результат записывается в CSV `results/frequencies.csv` формата:

`Rank,Frequency,Word`.

Дополнительно утилита формирует файл статистики `results/stats.txt` (объём входа, число токенов, число уникальных слов, средняя длина токена, время работы).

4.3 Аппроксимация параметров закона Ципфа

Анализ распределения и оценка параметров выполняются Python-скриптом `src/zipf_analyzer.py`. Для оценки s используется логарифмирование:

$$\ln f = \ln C - s \ln r,$$

что сводится к линейной регрессии в координатах $(\ln r, \ln f)$.

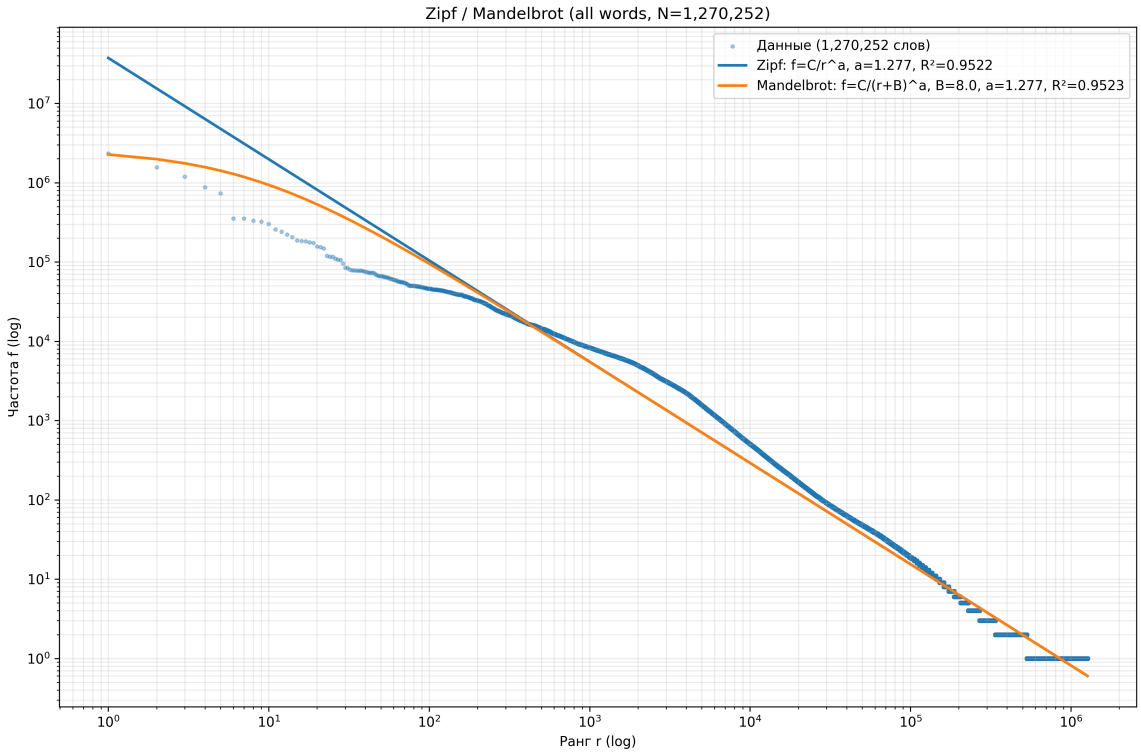

Рис. 1: Распределение частот терминов корпуса и аппроксимация законом Ципфа

4.4 Визуализация

На рисунке показано распределение частот слов корпуса в логарифмических координатах (по оси абсцисс отложен ранг слова r, по оси ординат — частота его встречаемости $f(r)$). Экспериментальные точки образуют характерную для естественного языка картину: небольшое число высокочастотных слов при малых рангах и длинный “хвост” редких термов при больших значениях r, где частоты убывают на несколько порядков.

Поверх эмпирических данных приведены аппроксимации законом Ципфа

$$f(r) = Cr^{-a}$$

и законом Мандельброта

$$f(r) = \frac{C}{(r+B)^a}.$$

Оценённый показатель степени $a \approx 1.277$ близок к типичным значениям для крупных текстовых корпусов. Качество аппроксимации является высоким: коэффициент детерминации составляет $R^2 \approx 0.952$ для закона Ципфа и немного больше для модели Мандельброта.

На среднем диапазоне рангов аппроксимирующие кривые хорошо согласуются с экспериментальными данными, что подтверждает выполнение закона Ципфа для данного корпуса. Отклонения в области малых рангов объясняются высокой долей наиболее частотных служебных слов и особенностями тематики корпуса, тогда как расхождения в правой части распределения связаны с наличием большого числа редких слов и дискретным характером частот (ступенчатость “хвоста” распределения).

5 Булев индекс и поиск

5.1 Нулевая модель булевого поиска

Реализован булев поиск в нулевой модели информационного поиска: документ либо удовлетворяет запросу, либо нет; веса термов и ранжирование не вычисляются. Основной акцент сделан на эффективной реализации постинг-листов и операций над ними.

5.2 Построение индекса (инвертированный файл)

Построение индекса выполняется утилитой `index_builder` (файл `src/boolean_index.cpp`). Индекс строится по дампу корпуса в текстовом формате со структурой:

```
==DOC_START==
<externalId>
==CONTENT_START==
<multiline content>
==DOC_END==
```

Для каждого документа:

- выделяются токены (алфавитно-цифровые последовательности), приводятся к нижнему регистру;
- внутри документа используется **уникализация токенов** (сортировка + `unique`);
- для каждого токена в хэш-таблице добавляется `doc_id` в постинг-лист.

Внутренняя структура отображения `term` → `postings` реализована как собственная хэш-таблица `SimpleHashMap` (цепочки в бакетах). Постинг-лист хранится как `vector<int>` и остаётся отсортированным благодаря монотонно возрастающему `doc_id`.

5.3 Формат индексного файла

Утилита сохраняет индекс в файл `data/boolean_index.idx` в текстовом формате:

```
DOCS
<N>
doc_id|title|preview
...
TERMS
<M>
term|docid,docid,docid
...
```

Где:

- `title` соответствует `externalId` из дампа;
- `preview` — первые 200 символов контента (с заменой переводов строк на пробел).

5.4 Операции над постинг-листами

Поисковый модуль `search` (файл `src/boolean_search.cpp`) поддерживает операции:

- **AND** — пересечение постинг-листов алгоритмом двух указателей (two-pointer);
- **OR** — объединение двух отсортированных списков с устранением дублей;
- **NOT** — дополнение списка относительно множества всех `doc_id`.

Все операции выполняются за линейное время от суммарной длины входных списков.

5.5 Обработка запросов

Запрос токенизируется по пробелам. Поддерживаются формы:

- `word1 word2` — неявный AND по всем словам;
- `word1 AND word2`;
- `word1 OR word2`;
- `NOT word`.

В случае более сложной строки без корректного парсинга операторов применяется режим “fallback”: игнорируются служебные токены `and/or/not`, а остальные термы пересекаются как AND.

6 Как работает поиск в приложении

6.1 Скрипт полного запуска

В корне офлайновой части расположен скрипт `run_all.sh`, который выполняет полный запуск:

1. компиляция C++-утилит (`compile.sh`);
2. токенизация корпуса и формирование `results/frequencies.csv`;
3. анализ закона Ципфа Python-скриптом;
4. построение булевого индекса в `data/boolean_index.idx`;
5. запуск интерактивного поиска.

6.2 Сборка утилит

Скрипт `compile.sh` компилирует:

- `src/tokenizer.cpp` → `bin/tokenizer`;
- `src/simple_stemmer.cpp` → `bin/stemmer`;
- `src/boolean_index.cpp` → `bin/index_builder`;
- `src/boolean_search.cpp` → `bin/search`.

Компиляция выполняется с оптимизацией `-O2`, стандартом `-std=c++11` и проверкой кода возврата каждого шага.

6.3 Интерактивный режим

Утилита `search` загружает индекс `data/boolean_index.idx` и запускает цикл чтения запросов. На каждый запрос:

- выполняется токенизация и интерпретация операторов;
- вычисляется результат (список `doc_id`);
- выводятся метаданные (external id) и preview первых документов;
- дополнительно измеряется время выполнения запроса (мс).

6.4 Демонстрация работы

Ниже предоставлены результаты работы программы).

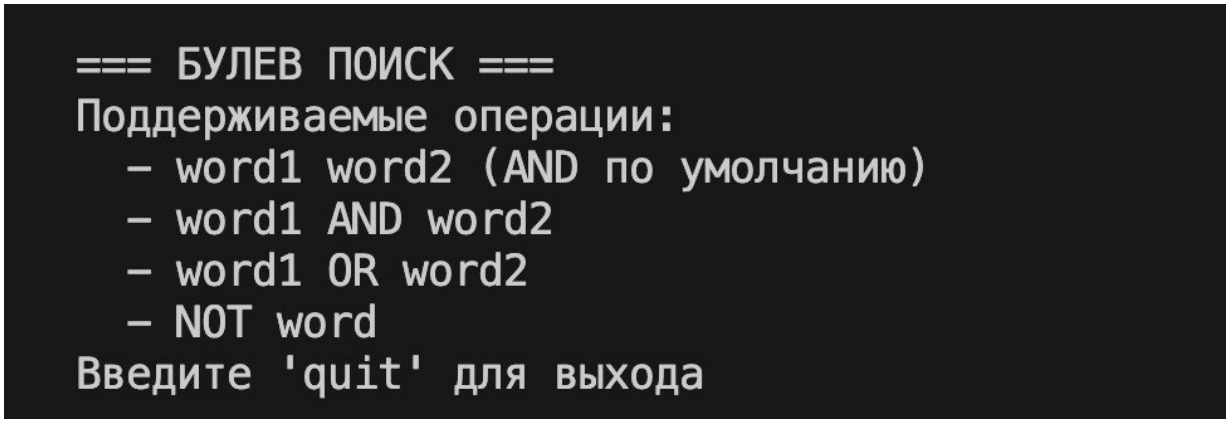

Рис. 2: Панель при запуске интерактивного поиска

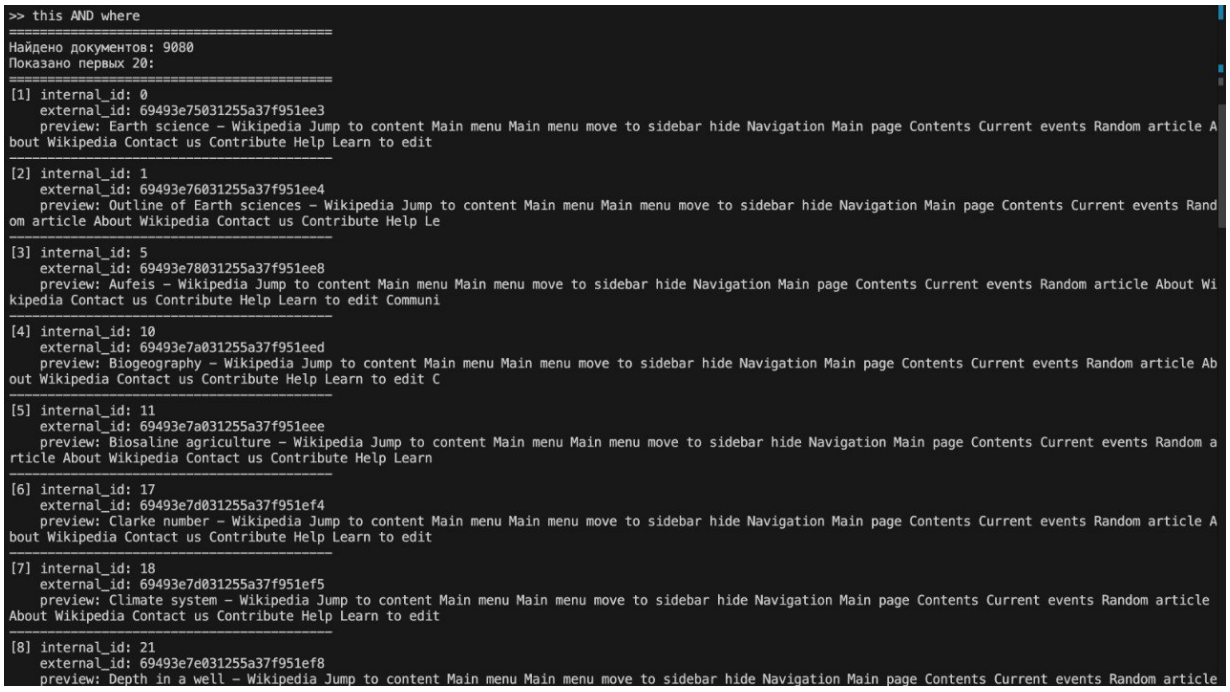

Рис. 3: Пример выполнения запроса и вывод результатов(выводится список подходящих документов + превью текста с совпадениями)

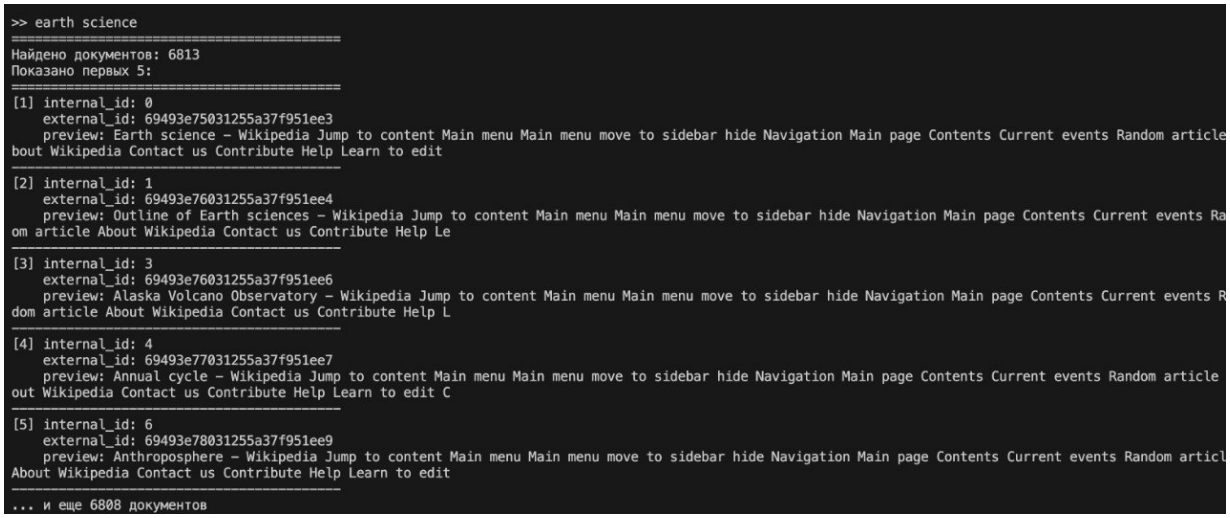

Рис. 4: Пример выполнения запроса и вывод результатов(выводится список подходящих документов + превью текста с совпадениями)

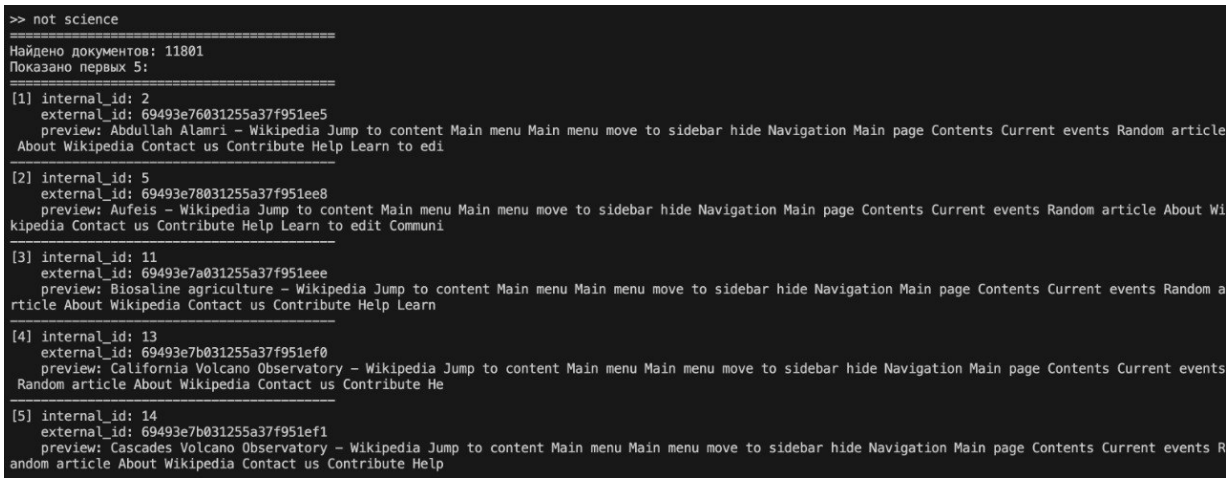

Рис. 5: Пример выполнения запроса и вывод результатов(выводится список подходящих документов + превью текста с совпадениями)

7 Выводы

В ходе выполнения лабораторных работ был реализован базовый конвейер информационного поиска: сбор корпуса документов, формирование частотных статистик и проверка закона Ципфа, построение булевого индекса и выполнение поисковых запросов.

Разработан поисковый робот на Kotlin/Spring Boot с хранением данных в MongoDB. Реализация поддерживает управление очередью URL (frontier), многопоточную обработку стадий discover/fetch, конфигурирование источников и resilient-поведение (повторы, backoff, фиксация ошибок).

Офлайновая часть реализована в виде набора утилит на C++ и Python-скрипта анализа. Построен инвертированный индекс с хранением постинг-листов в собственных структурах данных; реализованы операции булевого поиска AND/OR/NOT и интерактивный режим работы.

Система может служить основой для дальнейшего расширения: усложнение парсинга булевых выражений (скобки, приоритеты), введение ранжирования (TF–IDF/BM25), улучшение нормализации токенов и более строгая обработка языка корпуса.

8 Список литературы

1. Солтон Дж., Макгилл М. Введение в современный информационный поиск. — М.: Мир, 1983. — 416 с.

2. Кузнецов С. Д. Основы информационного поиска. — М.: Физматлит, 2009. — 320 с.

3. Мэннинг К., Рагхаван П., Шютце Х. Введение в информационный поиск. — М.: Диалектика, 2011. — 528 с.