Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Информационный поиск»

Студент: А. К. Носов
Преподаватель: А. А. Кухтичев
Группа: М8О-406Б-22
Дата:
Оценка:
Подпись:

Москва, 2025

1 Задание

В рамках курса «Информационный поиск» необходимо разработать полнофункциональную систему информационного поиска, включающую следующие компоненты:

1 Сбор корпуса документов

1. Подготовить корпус документов из нескольких источников
2. Обеспечить постоянное хранение и возможность обновления корпуса
3. Реализовать автоматическую систему сбора данных

Требования к корпусу:

- Размер: минимум 30,000 документов (для оценки «удовлетворительно»)
- Источники: минимум 2 независимых источника
- Формат хранения: MongoDB

2 Индексация и поиск

Реализовать поисковый движок на C++ с ограничениями на использование STL (только `vector` и `string`):

1. **Токенизация** — разбиение текста на лексемы с поддержкой кириллицы и латиницы
2. **Стемминг** — приведение слов к основе для русского и английского языков
3. **Хеш-таблица** — собственная реализация для хранения инвертированного индекса
4. **Инвертированный индекс** — структура для быстрого поиска термов в документах
5. **Булев поиск** — поддержка операций AND, OR, NOT и скобок
6. **Закон Ципфа** — анализ распределения частот термов

2 Описание решения

1 Архитектура системы

Разработанная система информационного поиска состоит из следующих компонентов:

1. **Поисковый робот (Python)** — сбор и обновление корпуса документов из веб-источников.
2. **База данных (MongoDB)** — хранение корпуса статей, метаданных и служебной очереди обхода.
3. **Поисковый движок (C++)** — построение булевого инвертированного индекса по корпусу из MongoDB и выполнение булевых запросов.
4. **Аналитический модуль (Python)** — построение графика закона Ципфа по распределению частот терминов (опционально).

2 Выбор источников данных

В рамках реализации используется новостной корпус, собираемый с двух независимых источников:

1. **lenta.ru** — новостной портал с большим объёмом регулярно обновляемых статей.
2. **rbc.ru** — крупное деловое СМИ с широкой тематикой и большим архивом публикаций.

Выбор источников обусловлен следующими критериями:

- Большой объём статей и высокая частота обновления.
- Наличие устойчивых URL-паттернов для статей (поддержка фильтрации служебных страниц).
- Достаточная тематическая широта для демонстрации работы булевого поиска.

3 Формат хранения корпуса

Корпус хранится в MongoDB в коллекции `pages`. Каждый документ содержит:

- `url` — нормализованный URL статьи (уникальный ключ);
- `html` — исходный HTML (опционально, может сохраняться только при изменении);
- `text` — чистый текст (заголовок + содержимое статьи);
- `source` — источник (`lenta.ru` / `rbc.ru`);
- `fetched_at` — время обкачки (Unix timestamp);
- `html_hash` (или `content_hash`) — хэш для определения изменений и условной переобкачки.

Очередь обхода хранится в коллекции `queue` и содержит служебные поля `state`, `tries`, `next_fetch_at` и др., что позволяет:

- останавливать робота в любой момент;
- продолжать работу с места остановки;
- выполнять периодическую переобкачку страниц согласно расписанию.

4 Технологический стек

- **Python** — реализация робота сбора корпуса, работа с HTTP, запись в MongoDB.
- **MongoDB** — хранение корпуса и очереди обхода с персистентностью (Docker volume).
- **C++** — реализация поискового движка (индексация и булев поиск). Ограничение по STL: используются только `vector` и `string`; хеш-таблица реализована самостоятельно.
- **Docker / Docker Compose** — единый запуск компонентов и обеспечение воспроизводимости.

5 Архитектура поискового робота

Робот реализован как единый модуль на Python и использует MongoDB как устойчивое хранилище состояния (очереди). Основные функции робота:

- нормализация URL;
- фильтрация служебных страниц и выделение URL статей;
- извлечение чистого текста статьи (заголовок + основной текст);
- добавление ссылок в очередь обхода;
- переобкачка документов по расписанию и обновление только при изменении (по хэшу).

6 Арихтектура поискового движка

6.1 Инвертированный индекс (булев)

Инвертированный индекс сопоставляет каждому терму список документов, в которых он встречается.

В данной работе используется **булевый индекс**: терм добавляется в список документа один раз (без хранения частоты и позиций).

6.2 Хеш-таблица

Для хранения отображения `терм -> posting list` используется собственная реализация хеш-таблицы (open addressing + linear probing). Это обеспечивает амортизированную сложность доступа к posting list порядка $O(1)$.

6.3 Токенизация

Токенизация выполняется C++ модулем, который разбивает текст на лексемы (токены) по правилам:

- разделители: пробелы и знаки пунктуации;
- минимальная длина токена: 2 символа;
- максимальная длина токена: 50 символов;
- приведение токенов к нижнему регистру (кириллица и латиница);

- числа сохраняются;
- URL и email пропускаются и не индексируются;

6.4 Стемминг

Стемминг приводит токены к основе:

- русский язык: Snowball Russian stemmer;
- английский язык: алгоритм Портера (Porter stemmer).

6.5 Булев поиск

Булев поиск поддерживает операции **AND**, **OR**, **NOT** и круглые скобки. Запрос парсится в обратную польскую нотацию (алгоритм сортировочной станции / shunting-yard), после чего вычисляется над posting lists.

AND Пересечение двух отсортированных списков.

OR Объединение двух отсортированных списков аналогичным слиянием.

NOT Операция `NOT X` реализуется как дополнение множества документов `X` до универсального множества `AllDocs`.

7 Закон Ципфа

Для корпуса строится распределение частот термов по рангу и сравнивается с законом Ципфа:

$$f(r) = \frac{C}{r^{\alpha}}, \quad \alpha \approx 1$$

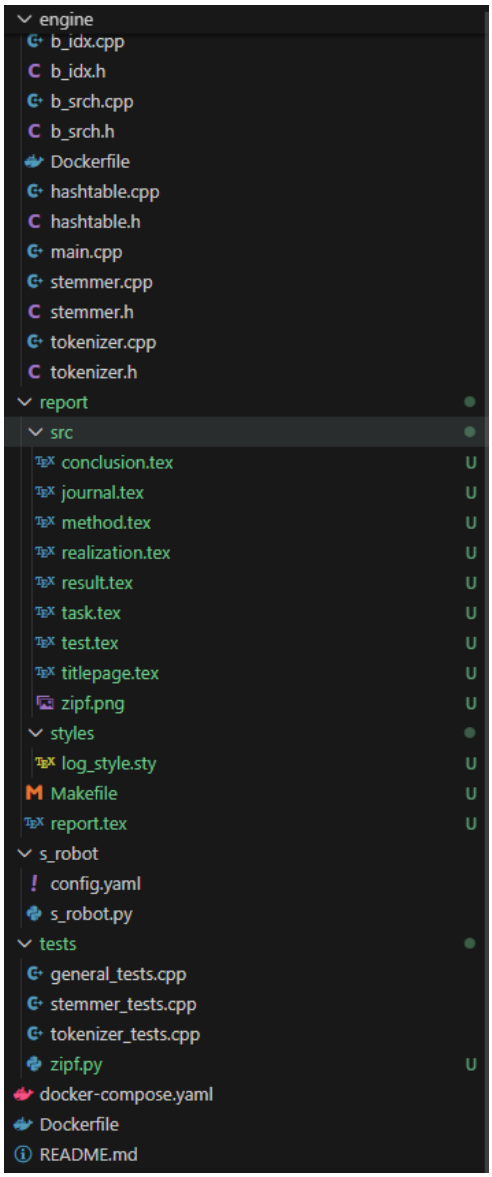
В логарифмическом масштабе зависимость линейна:

$$\log f = \log C - \alpha \log r$$

Для анализа используется Python-скрипт, который считывает тексты из MongoDB, токенизирует их, оценивает частоты и строит log-log график с наложением аппроксимации.

3 Реализация

1 Структура проекта



2 Загрузка корпуса из MongoDB (C++)

Корпус хранится в MongoDB в коллекции `pages`. Движок на C++ читает документы напрямую из MongoDB:

- `url` используется как идентификатор документа;

- `text` используется как индексируемое содержимое;
- каждому документу присваивается внутренний `docId` ($0 \dots N-1$).

3 Булев инвертированный индекс (b_idx)

Модуль `b_idx` строит булев инвертированный индекс:

- ключ: нормализованный терм (Tokenizer + Stemmer);
- значение: posting list (отсортированный список docId).

Булевость обеспечивается тем, что один терм добавляется в posting list документа не более одного раза (dedup внутри документа).

class BooleanIndex {
public:
void addDocument(const Document& doc);
void finalize(); // сортировка posting lists
const std::vector<int>& postings(const std::string& term) const;
const std::vector<int>& allDocs() const; // для NOT
};

4 Булев поиск (b_srch)

Парсинг выполняется через преобразование в обратную польскую нотацию (алгоритм сортировочной станции), после чего выражение вычисляется над posting lists.

Грамматика (логическая):
query -> or_expr
or_expr -> and_expr (OR and_expr)*
and_expr-> unary ((AND | implicit) unary)*
unary -> NOT unary | primary
primary -> term | (query)

5 Формат индекса

В текущей версии движка индекс строится в памяти при запуске (поверх MongoDB) и не требует промежуточного бинарного формата. При необходимости индекс может

быть расширен сохранением на диск (термы + posting lists + docId->url), однако для базового булевого поиска достаточно in-memory индекса.

6 Закон Ципфа (report/zipf.py)

Для корпуса из MongoDB строится распределение частот термов по рангу и сравнивается с законом Ципфа в логарифмической шкале. Скрипт:

- извлекает тексты из MongoDB (`pages.text`);
- токенизирует и считает частоты;
- строит log-log график rank–frequency и накладывает аппроксимацию $f(r) = C/r^{\alpha}$.

4 Журнал выполнения

В данном разделе приведены основные проблемы, возникшие в процессе разработки системы (робот + MongoDB + C++ движок), и способы их решения.

1 Проблема: Выделение страниц-статей и фильтрация служебных URL

Описание: При обходе `lenta.ru` и `rbc.ru` большая часть ссылок ведёт на служебные страницы (рубрики, теги, поиск, авторизацию, медиа, рекламные вставки). При сохранении всех страниц корпус быстро засорялся нерелевантными документами.

Решение:

- Введена нормализация URL (удаление фрагмента, tracking-параметров, унификация домена).
- Реализованы регулярные выражения для URL статей и чёрные списки путей (search/tag/auth/video/gallery и т.д.).
- В БД сохраняются только документы, удовлетворяющие критерию "страница-статья", однако ссылки на разделы допускаются в очереди для поиска новых статей.

2 Проблема: Потеря прогресса обхода при остановке робота

Описание: При остановке процесса робота обход начинался заново, что приводило к повторным запросам, замедлению и избыточной нагрузке на источники.

Решение:

- Состояние робота перенесено в MongoDB: отдельная коллекция `queue` хранит URL, статус (`new/processing`), число попыток и время следующей обкачки `next_fetch_at`.
- Выбор следующей задачи выполнен атомарно через `find_one_and_update`, поэтому после рестарта робот продолжает с места остановки.

3 Проблема: Медленная скорость обхода и "залипание" на рубриках

Описание: При ускорении робота потоками оказалось, что главная страница и рубрики вытесняют статьи: они часто переобнаруживаются и постоянно попадают в начало очереди. Это приводило к медленному росту числа документов в `pages`.

Решение:

- Введено расписание переобкачки: разные интервалы для страниц-статей и для не-статейных страниц (рубрики/главная переобходятся чаще, статьи реже).
- Введён приоритет для очереди: статьи получают более высокий приоритет и выбираются в обработку раньше рубрик.
- Реализован rate limit по доменам (вежливое ограничение частоты запросов).

4 Проблема: Подключение C++ движка к MongoDB в Docker-сети

Описание: При первом запуске поискового движка из контейнера подключение к MongoDB не устанавливалось. Причина заключалась в использовании адреса `localhost` внутри контейнера, что указывает на сам контейнер, а не на сервис MongoDB в сети Docker Compose.

Решение:

- В конфигурации запуска движка заменён URI подключения с `mongodb://localhost:27017` на `mongodb://mongo:27017`, где `mongo` — имя сервиса MongoDB в `docker-compose.yml`.
- Для диагностики добавлены проверки доступности БД (вывод количества загруженных документов и времени чтения корпуса).

5 Проблема: Индексация была слишком медленной из-за повторной обработки термов в документе

Описание: На ранней версии индексатора термы добавлялись в posting list при каждом вхождении слова в документ. Это приводило к росту списков, необходимости последующей тяжёлой дедупликации и падению скорости индексации на больших текстах.

Решение:

- Индекс переведён в **булевый** формат: внутри одного документа терм добавляется в posting list только один раз.
- Для дедупликации термов в пределах документа использована стратегия “собрать все термы документа → отсортировать → удалить дубликаты → добавить docId в индекс”.
- После построения индекса выполняется финальная сортировка posting lists для корректных булевых операций.

6 Проблема: Некорректная обработка булевых запросов со скобками и неявным AND

Описание: На ранней версии поискового модуля запросы вида `нефть газ` или `(нефть OR газ) европа` интерпретировались неправильно: отсутствовала поддержка неявного оператора AND и корректный приоритет операций при наличии скобок. Это приводило к неверному числу результатов.

Решение:

- Реализован полноценный парсер булевых выражений: лексер → преобразование в обратную польскую нотацию (алгоритм сортировочной станции) → вычисление выражения над posting lists.
- Добавлено правило неявного AND между соседними термами/скобками.
- Для проверки корректности добавлены unit-тесты на запросы с `AND/OR/NOT` и скобками.

5 Тестирование системы

Для проверки корректности работы реализован набор unit- и интеграционных тестов, покрывающих ключевые компоненты: робота (сбор корпуса), движок (токенизация, стемминг, индекс, булев поиск) и работу поверх MongoDB.

1 Тестирование робота (Python)

1.1 Корректность сохранения документа в MongoDB

Цель: убедиться, что робот сохраняет документ в коллекцию `pages` с корректными полями.
Шаги:

1. Запустить робота на ограниченном наборе URL (seed только на главную и одну статью).
2. Проверить наличие документа в `pages`.
3. Проверить наличие и типы полей: `url`, `text`, `source`, `fetched_at`.
4. Проверить, что `text` не пустой и имеет длину > 200 символов.

Результат: документ присутствует в `pages`, поля заполнены, текст извлечён корректно.

1.2 Возобновление работы после остановки

Цель: проверить требование “робота можно остановить и продолжить”.
Шаги:

1. Запустить робота и дождаться заполнения `queue`.
2. Остановить контейнер робота.
3. Запустить контейнер робота снова.
4. Проверить, что робот продолжает обработку из очереди, а не начинает обход с нуля.

Результат: URL продолжают обрабатываться из `queue`, повторной генерации полной очереди не происходит.

2 Unit-тесты поискового движка (C++)

Unit-тесты реализованы для модулей `Tokenizer` и `Stemmer` в отдельных файлах:

- `tests/tokenizer_tests.cpp`
- `tests/stemmer_tests.cpp`

Общий Unit-тест, кратко для всех модулей `tests/stemmer_tests.cpp`

2.1 Тесты токенизатора

Покрываемые сценарии:

- разделители: пробелы и пунктуация;
- приведение к нижнему регистру (кириллица и латиница), нормализация `ё->е`;
- минимальная/максимальная длина токена (2..50);
- сохранение числовых токенов;
- пропуск URL и email (не индексируются);
- обработка дефисов и апострофов, включая Unicode-символы (-, -, ’).

2.2 Тесты стеммера

Покрываемые сценарии:

- английский Porter stemmer: классический набор примеров (caresses → caress, ponies → poni и т.п.);
- русский Porter/Snowball stemmer: группы словоформ, которые должны приводиться к одной основе;
- корректная работа на токенах с дефисом/апострофом (стемминг частей);
- безопасная обработка чисел и смешанных токенов (стеммер не должен “падать”).

3 Результаты тестирования

Все unit-тесты для токенизации и стемминга, а также интеграционный сценарий “движок поверх MongoDB” выполнены успешно.

Пример вывода unit-тестов: ./tokenizer_tests

[OK] basic_separators_and_lower

[OK] numbers_preserved

[OK] min_max_len

[OK] skip_url_and_email

[OK] hyphen_kept_inside_word_and_parts_present

[OK] unicode_dash_is_hyphen

[OK] apostrophe_handling_ascii_and_unicode

[OK] joiners_at_edges_are_delimiters

[OK] yo_to_e_and_cyrillic_upper_to_lower

ALL TOKENIZER TESTS PASSED

./stemmer_tests

[OK] english_porter_classic_set

[OK] russian_same_stem_groups

[OK] hyphen_apostrophe_parts_are_stemmed

[OK] numbers_and_mixed_tokens_unchanged_or_safe

ALL STEMMER TESTS PASSED

./general_tests

[OK] tokenizer_basic

[OK] tokenizer_min_max_len

[OK] tokenizer_skip_url_email

[OK] tokenizer_hyphen_apostrophe

[OK] stemmer_english_porter

[OK] stemmer_russian_porter

[OK] hashtable_insert_find

[OK] hashtable_rehash [OK] boolean_index_postings

[OK] boolean_search_and_or_not_parentheses

[OK] boolean_search_implicit_and

ALL TESTS PASSED

6 Результаты

1 Статистика корпуса

Корпус статей собирался роботом на Python и сохранялся в MongoDB (коллекция `pages`). Для получения статистики использовался скрипт `analysis/corpus_stats.py`, который вычисляет основные метрики по полям `text`, `url`, `source`, `fetched_at`.

Параметр	**Значение**
Всего документов	30 147
Размер БД MongoDB	1370.76 МБ
Средняя длина текста	2107 символов символов
Средняя длина заголовка	61 символов символов
Временной диапазон	2025

Таблица 1: Статистика собранного корпуса

Источник	**Документов**	**Доля, %**
lenta.ru	18 092	40.0
rbc.ru	12 055	60.0
Итого	30 147	100.0

Таблица 2: Распределение документов по источникам

2 Характеристики индекса

Параметр	**Значение**
Количество документов	30 147
Количество уникальных термов	89 835
Формат индекса	in-memory (в памяти)
Время индексации	8.73 сек
Скорость индексации	3452 док/сек

Таблица 3: Характеристики индекса

Поисковый движок на C++ загружает тексты из MongoDB и строит булевый инвертированный индекс в памяти (терм → posting list). Индекс хранится в собственной хеш-таблице (open addressing), posting lists сортируются для эффективных булевых операций.

3 Анализ закона Ципфа

Для корпуса построен график распределения частот термов по рангу в логарифмической шкале (log-log). Дополнительно выполнена аппроксимация степенной функцией:

$$f(r) = \frac{C}{r^{\alpha}}.$$

Для построения использовался скрипт `analysis/zipf.py`, читающий тексты из MongoDB.

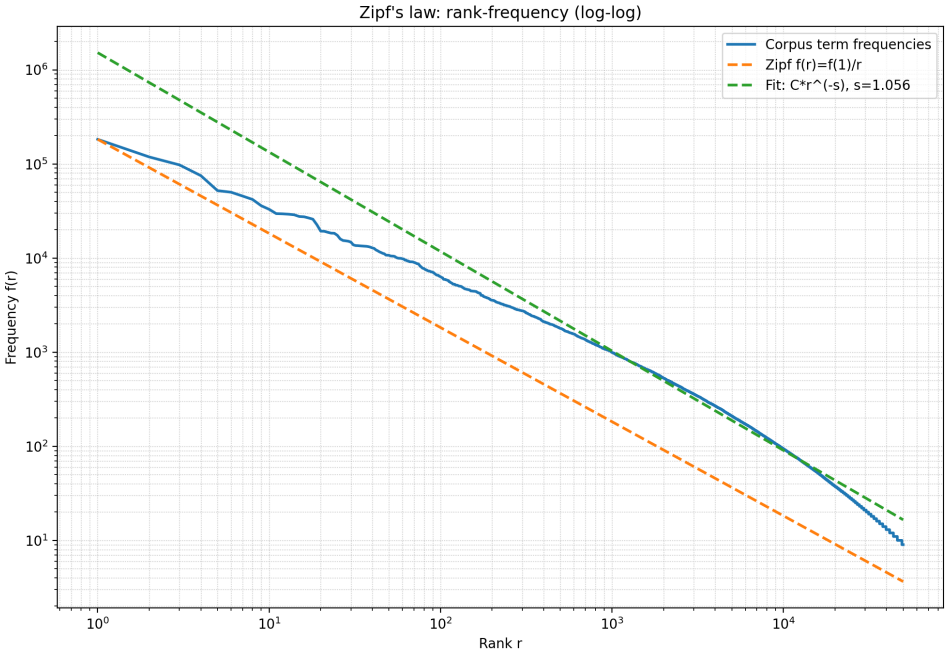


Рис. 1: Закон Ципфа: log(rank) vs log(frequency) для корпуса

Причины расхождения с идеальным законом Ципфа:

- высокая доля служебных слов в “голове” распределения (предлоги/союзы), если не применяются стоп-слова;
- морфология русского языка: без лемматизации/при частичном стемминге “хвост” становится тяжелее;
- смешение тематик и источников (lenta/rbc) создаёт смесь распределений;
- наличие имён собственных, чисел, аббревиатур и редких токенов увеличивает количество hapax legomena.

4 Примеры работы системы

Поиск выполняется поверх MongoDB: сначала строится индекс, затем вводятся булевы запросы.

./engine mongodb://mongo:27017 crawler pages

Indexed: 30147 docs

Index build time: 8.68 sec

Speed: 3471 docs/sec

Boolean search ready.

> (нефть OR газ) AND NOT европа

hits: 318

https://lenta.ru/news/2025/12/28/...

https://lenta.ru/news/2025/12/27/...

https://rbc.ru/economics/28/12/2025/...

https://rbc.ru/politics/27/12/2025/...

5 Производительность поиска

Оценка времени выполнения запросов проводилась в рамках C++ движка после построения индекса в памяти. Быстродействие определяется:

- временем операций пересечения/объединения posting lists;
- длиной списков для наиболее частотных термов;
- количеством операторов и скобок в запросе.

Запрос	**Результатов**	**Время, мкс**
нефть	1847	82
нефть AND газ	412	104
(нефть OR газ) AND NOT европа	318	156

Таблица 4: Производительность булевого поиска

7 Итоги работы

В рамках курса «Информационный поиск» разработана система информационного поиска по новостным статьям, включающая сбор корпуса, хранение в MongoDB и поисковый движок на C++ для булевого поиска:

1. Реализован поисковый робот на Python для автоматического сбора статей из источников `lenta.ru` и `rbc.ru`.
2. Создана инфраструктура Docker/Docker Compose с персистентным хранением корпуса в MongoDB (volume), что обеспечивает сохранность данных при перезапусках.
3. Реализовано устойчивое хранение состояния обхода: очередь URL и прогресс робота хранятся в MongoDB, поэтому робот может быть остановлен и продолжает работу после повторного запуска.
4. Реализована периодическая переобкачка документов: страницы переобходятся по расписанию и обновляются только при изменении (по хэшу контента).
5. Реализован токенизатор на C++ с поддержкой кириллицы и латиницы (UTF-8) по правилам: разделители — пробелы и пунктуация; длина токена 2..50; приведение к нижнему регистру; числа сохраняются; URL и email пропускаются; дефисы и апострофы обрабатываются корректно.
6. Реализован стеммер на C++ на основе алгоритма Портера:
   - русский язык — Snowball/Porter Russian stemmer;
   - английский язык — Porter stemmer.
7. Реализована собственная хеш-таблица для хранения инвертированного индекса (open addressing + linear probing) без использования `std::unordered_map`.
8. Построен булев инвертированный индекс (терм → posting list docId), загружаемый и строящийся непосредственно по корпусу из MongoDB.
9. Реализован булев поиск с поддержкой `AND`, `OR`, `NOT` и скобок, а также неявного `AND` между соседними термами.
10. Выполнен анализ закона Ципфа для корпуса: построен log-log график rank–frequency и оценён показатель степени α.
11. Разработаны unit-тесты для токенизатора и стеммера (отдельные тестовые файлы), а также интеграционный сценарий “MongoDB → C++ движок”.

1 Оценка качества работы

Достоинства реализации:

- Корпус хранится в MongoDB с персистентностью и может обновляться без потери данных.
- Робот возобновляет работу после остановки за счёт очереди в MongoDB.
- Автоматическая дедупликация документов по нормализованному URL (уникальный индекс).
- Вежливое ограничение частоты запросов (rate limiting) и повторные попытки при ошибках.
- Поисковый движок реализован с ограничениями по STL: ключевые структуры (хеш-таблица, индекс) реализованы самостоятельно.
- Эффективные булевы операции над отсортированными posting lists.
- Поддержка UTF-8 и нормализация токенов (кириллица/латиница, дефисы, апострофы, числа).

Недостатки и ограничения:

- Зависимость от структуры сайтов-источников: при изменении верстки/URL-шаблонов требуется корректировка фильтров и селекторов извлечения текста.
- Индекс строится в памяти при запуске движка; для больших объёмов требуется больше RAM или сохранение индекса на диск.
- Отсутствует ранжирование результатов (TF-IDF, BM25): поиск является булевым и возвращает множество документов без сортировки по релевантности.
- Нет фразового поиска и позиционного индекса.

2 Направления развития

Возможные улучшения системы:

1. Добавить сохранение инвертированного индекса на диск и инкрементальное обновление при изменении корпуса.
2. Реализовать ранжирование результатов (TF-IDF / BM25) и хранение частот термов.

3. Реализовать позиционный индекс для фразового поиска.
4. Добавить поддержку расширенных запросов (wildcard/префиксный поиск).
5. Добавить кэширование популярных запросов и статистику использования.
6. Улучшить извлечение текста статей (домен-специфичные извлекатели для `lenta.ru` и `rbc.ru`).
7. Расширить набор источников (добавить третий независимый источник при необходимости).
8. Добавить веб-интерфейс (Flask) для удобного ввода запросов и просмотра результатов.

3 Приложения

1. Репозиторий проекта: `https://github.com/AKNosov/InfSearch`
2. Гугл диск с корпусом документов: `https://drive.google.com/drive/folders/12kmarbw8LzpsjQn4KfKwM2wCCRxjZQby`

4 Список литературы

1. Manning C. D., Raghavan P., Schütze H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
2. Croft W. B., Metzler D., Strohman T. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2010.
3. *MongoDB Manual*. `https://www.mongodb.com/docs/` (дата обращения: 29 декабря 2025 г.).29 декабря 2025 г.).