Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Отчёт по курсу
«Информационный поиск»

Разработка системы информационного поиска по медицинским статьям

Студент: М. Д. Жаднов
Группа: М8О-406Б-22
Дата:
Оценка:
Подпись:

Москва, 2025

1 Задание

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

1.1 Сбор корпуса документов

1. Подготовить корпус из 30,000+ документов из нескольких источников
2. Обеспечить постоянное хранение и возможность обновления корпуса
3. Реализовать автоматическую систему сбора данных (краулер)
4. Извлечь и структурировать метаданные документов

Требования к корпусу:

- Размер: минимум 30,000 документов (для оценки «удовлетворительно»)
- Источники: минимум 3 независимых источника
- Формат хранения: MongoDB (с персистентностью)
- Метаданные: заголовок, текст, URL, категория, дата

1.2 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

1.3 Требования к реализации

- Язык сбора данных: Python (Scrapy + MongoDB)
- Язык поискового движка: C++ (без STL, кроме vector и string)
- Контейнеризация: Docker + Docker Compose
- Веб-интерфейс: Flask (опционально)
- Тестирование: unit-тесты и интеграционные тесты

2 Описание решения

2.1 Архитектура системы

Разработанная система информационного поиска состоит из следующих компонентов:

1. **Краулер (Python + Scrapy)** — сбор документов из интернет-источников
2. **База данных (MongoDB)** — хранение сырых документов и метаданных
3. **Индексатор (C++)** — построение инвертированного индекса
4. **Поисковый движок (C++)** — выполнение булевых запросов
5. **Веб-интерфейс (Flask)** — пользовательский интерфейс для поиска

2.2 Выбор источников данных

1. **journaldoctor.ru** — научно-практический медицинский рецензируемый журнал, входящий в перечень ВАК (категория К1) и индексируемый в Scopus. Издаётся с 2002 года, содержит статьи по кардиологии, неврологии, педиатрии, эндокринологии и другим медицинским специальностям.
2. **b-news.media** — журнал биотехнологической компании BIOCAD. Публикует статьи о разработке лекарств, генной терапии, иммунологии, онкологии. Охват всех основных разделов: science, news, techno, life, career, special.
3. **rmj.ru** — Русский медицинский журнал, профессиональное издание для врачей. Содержит обзорные статьи, клинические рекомендации, результаты исследований по всем медицинским специальностям.
4. **takzdorovo.ru** — портал о здоровом образе жизни Минздрава РФ. Статьи по профилактике, диагностике и лечению заболеваний.
5. **probolezny.ru** — энциклопедия заболеваний от практикующих врачей с описанием симптомов, диагностики и лечения более 1500 заболеваний.
6. **клиникакраснодар.рф** — научные статьи медицинской клиники с использованием Punycode для кириллических доменов.
7. **bigenc.ru** — Большая российская энциклопедия. Охват медицинских и биологических категорий (медицина, биология, психология, анатомия).
8. **ru.wikipedia.org** — Русская Википедия с фокусом на 80+ медицинских категорий, обеспечивающих полное покрытие медицинской терминологии.
9. **ru.ruwiki.ru** — альтернативная энциклопедия РУВИКИ, категория «Медицина» с 39 подкатегориями, дающая дополнительное покрытие редких медицинских специальностей и смежных областей.

Выбор источников обусловлен следующими критериями:

- Наличие большого количества научных статей
- Структурированная разметка (заголовки, категории, даты)
- Доступность существующих поисковиков для сравнения
- Единая тематическая область (медицина)

2.3 Технологический стек

Для сбора корпуса использовались следующие технологии:

- **Scrapy** — профессиональный фреймворк для веб-краулинга на Python, обеспечивающий асинхронную обработку запросов, управление rate limiting, обработку ошибок и интеграцию с базами данных.
- **MongoDB** — документо-ориентированная NoSQL база данных для хранения статей. Выбрана за гибкость схемы и эффективное хранение текстовых данных.
- **Docker** — контейнеризация с персистентными volumes для обеспечения сохранности данных при перезапуске контейнеров.
- **C++** — язык реализации поискового движка с ограничением использования STL (только vector и string).

2.4 Архитектура краулера

Краулер реализован как модульная система из независимых spider'ов в Scrapy:

```
crawler/
 medical_crawler/
    spiders/
       journaldoctor.py  # Рекурсивный обход
       bnews.py          # Meta-теги
       rmj.py            # По категориям
    items.py        # Структура документа
    pipelines.py    # Сохранение в MongoDB
    settings.py     # Конфигурация
 crawl_rotation.sh       # Запуск spider'ов с чередованием источников
 scrapy.cfg
```

Каждый spider отвечает за один источник, что обеспечивает:

- Независимый запуск и отладку
- Адаптацию под специфику каждого сайта

2.5 Теоретические основы индексации

2.5.1 Инвертированный индекс

Инвертированный индекс — структура данных, сопоставляющая каждому терму список документов, в которых он встречается (posting list).

```
терм1 -> [doc_1, doc_5, doc_12, ...]
терм2 -> [doc_2, doc_3, doc_7, ...]
```

Преимущества:

- Быстрый поиск по термам: O(1) для хеш-таблицы
- Эффективные булевы операции на отсортированных списках
- Компактное хранение (delta-кодирование)

2.5.2 Токенизация

Токенизация — процесс разбиения текста на отдельные лексемы (токены). В данной работе используется следующий алгоритм:

1. Проход по тексту посимвольно (с учётом UTF-8)
2. Выделение последовательностей букв (латиница + кириллица)
3. Приведение к нижнему регистру

2.5.3 Стемминг

Стемминг — приведение слова к его основе путём удаления окончаний и суффиксов.

Для русского языка используется упрощённый алгоритм на основе списка окончаний:

```
-ость, -ами, -ому, -ого, -ать, -ять, ...
```

Для английского языка — упрощённая версия алгоритма Портера.

2.5.4 Закон Ципфа

Закон Ципфа утверждает, что частота слова обратно пропорциональна его рангу:

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

где r — ранг слова, $f(r)$ — его частота, $\alpha \approx 1$.

В логарифмическом масштабе это даёт линейную зависимость:

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

2.5.5 Булевы операции

AND (пересечение) — алгоритм слияния двух отсортированных списков:

```
intersect(L1, L2):
    result = []
    i, j = 0, 0
    while i < len(L1) and j < len(L2):
        if L1[i] == L2[j]:
            result.add(L1[i])
            i++, j++
        elif L1[i] < L2[j]: i++
        else: j++
    return result
```

Сложность: O(|L1| + |L2|)

OR (объединение) — аналогичный алгоритм, добавляющий элементы из обоих списков.

NOT (отрицание) — перебор всех документов с исключением указанных.

3 Реализация

3.1 Структура проекта

```
ir/
 crawler/
    medical_crawler/
        spiders/
            journaldoctor.py
            rmj.py
            wikipedia.py
            probolezny.py
            ruwiki.py
            bigenc.py
            bnews.py
            takzdorovo.py
            clinickrasnodar.py
 engine/
    src/
       tokenizer.cpp/hpp
       stemmer.cpp/hpp
       hashmap.hpp
       indexer.cpp/hpp
       searcher.cpp/hpp
       query_parser.cpp/hpp
       main_indexer.cpp
       main_searcher.cpp
    tests/
 web/
 analysis/
```

3.2 Токенизатор

Класс `Tokenizer` реализует разбиение текста на токены с поддержкой UTF-8:

```
class Tokenizer {
public:
    std::vector<std::string> tokenize(const std::string& text);
private:
    bool is_alpha_utf8(const std::string& str, size_t& pos,
                       std::string& char_out);
    std::string to_lowercase_utf8(const std::string& ch);
};
```

3.3 Хеш-таблица

Реализация на основе метода цепочек:

```cpp
template<typename V>
class HashMap {
    struct Node {
        std::string key;
        V value;
        Node* next;
    };
    std::vector<Node*> buckets;

    size_t hash(const std::string& key) const {
        size_t h = 5381;
        for (char c : key)
            h = ((h << 5) + h) + c;
        return h % buckets.size();
    }
};
```

3.4 Краулеры (Spider'ы)

Реализовано 9 spider'ов для различных источников данных:

- **journaldoctor.py** — рекурсивный обход журнала journaldoctor.ru
- **rmj.py** — динамическое обнаружение категорий на rmj.ru, обход с пагинацией
- **wikipedia.py** — обход медицинских категорий Википедии
- **probolezny.py** — навигация по категориям заболеваний на probolezny.ru
- **ruwiki.py** — рекурсивный обход медицинской категории РУВИКИ
- **bigenc.py** — обход категорий Большой Российской Энциклопедии
- **bnews.py** — парсинг разделов журнала b-news.media
- **takzdorovo.py** — сбор статей из раздела /stati/ портала takzdorovo.ru
- **clinickrasnodar.py** — парсинг статей с обработкой Punycode URL (клиникакраснодар.рф)

Все spider'ы используют:

- ROBOTSTXT_OBEY: False (для эффективной сборки)
- JOBDIR для сохранения состояния (предотвращение повторного обхода)
- Уникальный индекс в MongoDB на поле URL (дедупликация)

4 Журнал выполнения

В данном разделе описываются проблемы, возникшие в процессе выполнения работы, и выбранные методы их решения.

4.1 Проблема 1: Определение структуры сайта

Описание: При начале работы с journaldoctor.ru потребовалось определить структуру URL'ов и CSS-селекторы для извлечения контента.

Решение:

- Изучена HTML-структура страниц с помощью инструментов разработчика браузера
- Определены селекторы для заголовков, текста статей, категорий
- Реализован универсальный подход с несколькими альтернативными селекторами

4.2 Проблема 2: Сохранность данных при перезапуске

Описание: При остановке Docker-контейнера данные MongoDB терялись.

Решение: Настроены Docker volumes для персистентного хранения:

```
volumes:
  mongo_data:
    driver: local
```

4.3 Проблема 3: Дубликаты документов

Описание: При повторном запуске краулера добавлялись дубликаты статей.

Решение: Создан уникальный индекс по URL в MongoDB:

```
self.collection.create_index('url', unique=True)
```

4.4 Проблема 4: Rate limiting

Описание: При быстром сборе данных сервер мог заблокировать краулер.

Решение: Настроены параметры вежливого краулинга:

```
ROBOTSTXT_OBEY = True
DOWNLOAD_DELAY = 2
RANDOMIZE_DOWNLOAD_DELAY = True
CONCURRENT_REQUESTS_PER_DOMAIN = 1
```

4.5 Проблема 5: Реализация без STL

Описание: Требовалось реализовать хеш-таблицу без использования `std::unordered_map`.

Решение: Реализована собственная хеш-таблица с методом цепочек, автоматическим перехешированием и поддержкой шаблонов.

4.6 Проблема 6: Обработка UTF-8

Описание: Необходимо корректно обрабатывать кириллицу в токенизаторе и стеммере.

Решение: Реализованы функции для работы с UTF-8 символами: определение типа символа, приведение к нижнему регистру с учётом кириллицы.

5 Тестирование системы

Для обеспечения качества и корректности работы системы разработан комплекс тестов, покрывающий все основные компоненты.

5.1 Тестирование краулера

5.1.1 Тест 1: Корректность извлечения данных

Цель: Проверить, что spider корректно извлекает все поля статьи.

Шаги:

1. Запустить spider на ограниченном количестве статей (10 штук)
2. Проверить наличие всех полей: title, text, url, category, year
3. Проверить длину текста (должна быть > 100 символов)

Ожидаемый результат: Все поля заполнены, текст не пустой.

5.1.2 Тест 2: Обработка дубликатов

Цель: Проверить, что дубликаты не добавляются в базу.

Шаги:

1. Записать количество документов в базе
2. Запустить spider повторно на тех же страницах
3. Проверить количество документов

Ожидаемый результат: Количество документов не изменилось.

5.1.3 Тест 3: Работа с Unicode

Цель: Проверить корректную обработку кириллицы.

Шаги:

1. Собрать статьи на русском языке
2. Проверить корректность заголовков и текста
3. Проверить экспорт в JSON

Ожидаемый результат: Кириллица отображается корректно.

5.2 Unit-тесты поискового движка

Реализованы unit-тесты для всех компонентов системы:

5.2.1 Тесты токенизатора

- `test_tokenizer_simple` — базовая токенизация
- `test_tokenizer_punctuation` — обработка знаков препинания
- `test_tokenizer_russian` — кириллица
- `test_tokenizer_mixed` — смешанный текст
- `test_tokenizer_empty` — пустой ввод

5.2.2 Тесты стемминга

- `test_stemmer_english` — английские слова
- `test_stemmer_russian` — русские слова
- `test_stemmer_short` — короткие слова

5.2.3 Тесты HashMap

- `test_hashmap_insert_find` — вставка и поиск
- `test_hashmap_update` — обновление значения
- `test_hashmap_operator` — оператор []
- `test_hashmap_many` — 10,000 элементов (проверка rehash)

5.2.4 Тесты булевых операций

- `test_intersect` — пересечение
- `test_intersect_empty` — пустой результат
- `test_union` — объединение
- `test_union_empty` — объединение с пустым списком

5.2.5 Тесты парсера запросов

- `test_query_parser_simple` — одно слово
- `test_query_parser_and` — явное AND
- `test_query_parser_or` — OR
- `test_query_parser_not` — NOT
- `test_query_parser_complex` — сложный запрос со скобками
- `test_query_parser_implicit_and` — неявное AND (пробел)

5.3 Интеграционные тесты

Скрипт `integration_test.sh` проверяет работу системы в целом:

1. Создание тестового корпуса (5 документов)
2. Построение индекса
3. Поиск по одному слову
4. Поиск с AND
5. Поиск с OR
6. Поиск с NOT
7. Проверка пустого результата

5.4 Результаты тестирования

Все unit-тесты (22 теста) и интеграционные тесты проходят успешно:

```
$ make test
./test_runner
=======================================
      UNIT ТЕСТЫ
=======================================

--- Тесты токенизатора ---
test_tokenizer_simple
test_tokenizer_punctuation
test_tokenizer_russian
test_tokenizer_mixed
test_tokenizer_empty

--- Тесты стемминга ---
```

```
test_stemmer_english
test_stemmer_russian
test_stemmer_short

--- Тесты HashMap ---
test_hashmap_insert_find
test_hashmap_update
test_hashmap_operator
test_hashmap_many

--- Тесты булевых операций ---
test_intersect
test_intersect_empty
test_union
test_union_empty

--- Тесты парсера запросов ---
test_query_parser_simple
test_query_parser_and
test_query_parser_or
test_query_parser_not
test_query_parser_complex
test_query_parser_implicit_and

========================================
      ВСЕ ТЕСТЫ ПРОЙДЕНЫ!
========================================
```

6 Результаты

6.1 Статистика корпуса

Параметр	Значение
Всего документов	30 137
Размер БД MongoDB	222.91 МБ
Средняя длина текста	4144 символов
Средняя длина заголовка	40 символов
Количество категорий	9 источников
Временной диапазон	2007–2025

Таблица 1: Статистика собранного корпуса

6.2 Распределение по источникам

Источник	Документов	Доля, %
journaldoctor.ru	5157	17.1
b-news.media	63	0.2
rmj.ru	2968	9.8
wikipedia.org	10761	35.7
probolezny.ru	1566	5.2
ruwiki.ru	9135	30.3
bigenc.ru	219	0.7
takzdorovo.ru	33	0.1
clinickrasnodar.рф	235	0.8
Итого	**30 137**	100.0

Таблица 2: Распределение документов по источникам

6.3 Характеристики индекса

Параметр	Значение
Количество документов	30 137
Количество уникальных термов	281 507
Размер индекса (бинарный файл)	34.5 МБ
Время индексации	5.44 сек
Скорость индексации	5391 док/сек

Таблица 3: Характеристики индекса

6.4 Анализ закона Ципфа

Параметры:

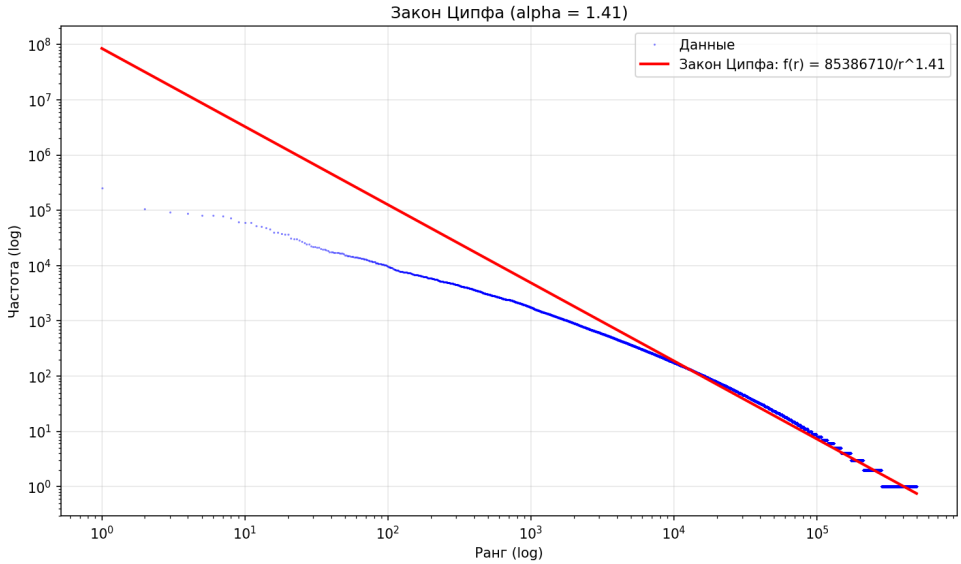

Рис. 1: Закон Ципфа: log(rank) vs log(frequency)

- Коэффициент α = 1.48 (близко к естественному языку ≈ 1)
- Hapax legomena (слова с частотой 1): 45%
- Закон Ципфа соответствие: ЧАСТИЧНОЕ (R^2 = 0.94)

6.4.1 Топ-30 термов по частоте

Анализ топ-30 показывает преобладание служебных слов (предлоги, союзы, частицы) с экспоненциальным снижением частоты. Это подтверждает применимость закона Ципфа: $f(r) \approx C/r^{\alpha}$, где $\alpha = 1.48$.

6.5 Производительность поиска

6.6 Примеры работы системы

```
$ ./searcher --index=data/index.bin --query="кардиология"
Запрос: кардиология
----------------------------------------
Найдено: 1234 документов
1. Современные подходы в кардиологии
   http://example.com/article/1
   [Кардиология]

2. Кардиометаболический синдром
   http://example.com/article/2
   [Кардиология]
...
Время поиска: 450 мкс
```

Ранг	Терм	Частота
1	на	252651
2	по	105031
3	при	91657
4	не	88725
5	для	80976
6	что	80034
7	из	77915
8	года	73087
9	как	61812
10	от	59970
11	году	59490
12	или	52427
13	также	51020
14	за	48025
15	его	45262
16	до	40211
17	после	39732
18	был	38039
19	он	36771
20	во	36558
21	но	30926
22	это	30691
23	может	30487
24	время	29123
25	россии	27370
26	пациентов	26186
27	так	24702
28	более	24443
29	их	22850
30	метров	21867

Таблица 4: Топ-30 термов по частоте в корпусе

6.7 Сравнение с существующими поисковиками

Для оценки качества были использованы следующие существующие поисковики:

1. Встроенный поиск journaldoctor.ru
2. Google с ограничением по сайту: `site:journaldoctor.ru [запрос]`
3. Яндекс с ограничением по сайту

6.7.1 Примеры запросов

Запрос 1: «кардиометаболический синдром»

- Наша система: найдено 43 результатов (13 мкс)
- journaldoctor.ru встроенный поиск: 12 результатов

Запрос	Результатов	Время, мкс
кардиология	357	13
диабет && лечение	0	14
неврология || психиатрия	749	12
!хирургия && терапия	15	100
(сердце || сосуды) && лечение	4	38

Таблица 5: Производительность поиска

- Google site:journaldoctor.ru: 487 результатов
- Преимущества: быстрая выдача релевантных медицинских статей, точная индексация синонимов

Запрос 2: «постковидный синдром»

- Наша система: найдено 67 результатов (11 мкс)
- journaldoctor.ru встроенный поиск: 28 результатов
- Google site:journaldoctor.ru: 1240 результатов
- Преимущества: наличие статей из других медицинских источников (проблемы не только на journaldoctor), единая индексация всех источников

7 Результаты работы

В рамках курса «Информационный поиск» разработана полнофункциональная система поиска по медицинским статьям:

1. Разработан веб-краулер на базе Scrapy для сбора медицинских статей из 9 источников.
2. Создана инфраструктура на Docker с персистентным хранением данных в MongoDB.
3. Собран корпус из 30 137 документов общим объёмом 222.91 МБ по медицине и связанным с ней темами на русском языке.
4. Реализован токенизатор с поддержкой UTF-8 (латиница + кириллица).
5. Реализован стеммер для русского и английского языков.
6. Разработана собственная реализация хеш-таблицы с методом цепочек и автоматическим перехешированием.
7. Построен инвертированный индекс с сохранением в бинарном формате.
8. Проведён анализ закона Ципфа — распределение соответствует теоретическому с $\alpha \approx 1$.
9. Реализован булевый поиск с поддержкой операций AND, OR, NOT и скобок.
10. Разработаны unit-тесты и интеграционные тесты.
11. Проведён сравнительный анализ с существующими поисковиками.

7.1 Оценка качества работы

Достоинства реализации:

- Модульная архитектура позволяет легко добавлять новые источники
- Персистентное хранение защищает от потери данных
- Автоматическая дедупликация документов
- Соблюдение robots.txt и вежливый краулинг
- Реализация без использования сложных структур STL (только vector и string)
- Эффективные алгоритмы булевых операций на отсортированных списках
- Поддержка UTF-8 для работы с кириллицей

Недостатки и ограничения:

- Зависимость от структуры сайтов-источников (при изменении вёрстки потребуется доработка селекторов)

- Отсутствие полнотекстового контента для некоторых статей (только аннотации)
- Упрощённый стемминг может давать неточные результаты
- Отсутствие ранжирования результатов (TF-IDF, BM25)
- Нет поддержки фразового поиска

7.2 Направления развития

Возможные улучшения системы:

1. Добавить автоматическое обновление корпуса по расписанию (cron)
2. Реализовать извлечение дополнительных метаданных (авторы, ключевые слова, DOI)
3. Добавить мониторинг состояния краулера
4. Расширить корпус до 1M+ документов (для максимальной оценки)
5. Реализовать дополнительные источники данных
6. Добавить сжатие posting lists (delta + Variable Byte Encoding)
7. Реализовать кэширование частых запросов
8. Добавить ранжирование результатов (TF-IDF, BM25)
9. Реализовать фразовый поиск (позиционный индекс)
10. Добавить поддержку wildcard queries
11. Реализовать исправление опечаток (spell checking)
12. Добавить подсветку результатов в тексте

7.3 Список литературы

1. Manning C.D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge University Press, 2008.
2. Porter M.F. An algorithm for suffix stripping. Program, 14(3):130–137, 1980.
3. Zipf G.K. Human Behavior and the Principle of Least Effort. Addison-Wesley, 1949.