МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторные работы
по курсу «Информационный поиск»

Выполнил: Марков Владимир Игоревич
Группа: М8О-408Б-22
Преподаватель: Кухтичев Антон Алексеевич

Москва, 2025

Содержание

1 Введение **3**

2 Лабораторная работа №1: Добыча корпуса документов **3**
2.1 Цель работы . . . 3
2.2 Описание источников . . . 3
2.3 Структура документа . . . 3
2.4 Выделение текста . . . 3
2.5 Статистика корпуса . . . 4
2.6 Проверка пригодности . . . 4

3 Лабораторная работа №2: Поисковый робот **4**
3.1 Цель работы . . . 4
3.2 Архитектура и стек технологий . . . 4
3.3 Алгоритм работы . . . 4
3.4 Результат . . . 5

4 Лабораторная работа №3: Токенизация **5**
4.1 Цель работы . . . 5
4.2 Правила токенизации . . . 5
4.3 Достоинства метода . . . 5
4.4 Недостатки метода . . . 5
4.5 Примеры неудачной токенизации . . . 6
4.6 Статистика токенизации . . . 6
4.7 Производительность . . . 6

5 Лабораторная работа №4: Стемминг **6**
5.1 Цель работы . . . 6
5.2 Метод решения . . . 6
5.3 Результаты . . . 7
5.4 Интеграция в поисковую систему . . . 7

6 Лабораторная работа №5: Закон Ципфа **7**
6.1 Цель работы . . . 7
6.2 Методика . . . 7
6.3 Результаты . . . 8
6.4 Причины расхождений . . . 8
6.5 Выводы . . . 9

7 Лабораторная работа №6: Булев индекс **9**
7.1 Цель работы . . . 9
7.2 Внутреннее представление документа . . . 9
7.3 Реализация индексатора . . . 9
7.3.1 Структуры данных . . . 9
7.3.2 Метод сортировки . . . 10
7.3.3 Бинарный формат индекса (побайтовое описание) . . . 10
7.4 Результаты . . . 11
7.5 Скорость индексации . . . 11

8 Лабораторная работа №7: Булев поиск **11**
8.1 Цель работы 11
8.2 Синтаксис запросов 11
8.3 Алгоритм поиска 12
8.4 Примеры запросов 12
8.5 Веб-интерфейс 12
8.6 CLI-утилита 12
8.7 Производительность 13
8.8 Тестирование корректности 13

9 Выводы **13**

1 Введение

В рамках курса «Информационный поиск» была разработана полнофункциональная поисковая система, включающая подсистемы сбора данных (краулер), обработки текста (токенизация, стемминг), индексации и поиска. Ядро системы (индексация и поиск) реализовано на языке C++ с использованием собственных структур данных (вектор, хеш-таблица) в соответствии с требованиями курса.

2 Лабораторная работа №1: Добыча корпуса документов

2.1 Цель работы

Собрать и проанализировать корпус текстовых документов объемом не менее 30 000 статей для последующего использования в поисковой системе.

2.2 Описание источников

В качестве предметной области выбрана научная литература по биологии и медицине. Источниками данных послужили:

- **PubMed Central (PMC)** — полнотекстовый архив биомедицинской литературы (формат XML/HTML). Собрано 28,999 статей.
- **Wikipedia** — энциклопедические статьи по генетике и биотехнологиям (CRISPR, редактирование генов). Собрано 3,810 статей.

Документы представляют собой научные статьи и энциклопедические тексты, содержащие заголовок, авторов (при наличии), аннотацию и основной текст. Язык документов — английский.

2.3 Структура документа

«Сырой» документ (HTML/XML) содержит:

- Метаданные (DOI, журнал, дата публикации).
- Заголовок статьи (`<article-title>`, `<h1>`).
- Текст аннотации (`<abstract>`).
- Основной текст статьи, разбитый на секции (`<sec>`, `<p>`).

2.4 Выделение текста

Для выделения полезного контента использовался парсер, удаляющий служебные теги, скрипты, стили и навигационные элементы. В итоговый текстовый документ включаются:

1. Заголовок.
2. Аннотация.
3. Основной текст статьи.

2.5 Статистика корпуса

Параметр	**Значение**
Количество документов (всего)	32,809
из PubMed Central (PMC)	28,999
из Wikipedia	3,810
Средний размер текста (на док.)	$\approx$ 12 КБ
Общий объем выделенного текста	$\approx$ 400 МБ

Таблица 1: Статистика собранного корпуса

2.6 Проверка пригодности

Поиск по источникам (PubMed, PMC) доступен через официальные веб-интерфейсы и API (E-utilities), что позволяет использовать данный корпус для сравнения результатов.

3 Лабораторная работа №2: Поисковый робот

3.1 Цель работы

Разработать поисковый робот (crawler) для автоматического сбора документов с поддержкой докачки, проверки свежести и соблюдения вежливости (задержки).

3.2 Архитектура и стек технологий

Робот реализован на языке **Python**. Основные компоненты:

- **YAML-конфигурация**: параметры запуска (seeds, timeout, user-agent) вынесены в отдельный файл.
- **MongoDB**: используется для хранения метаданных документов и очереди скачивания (frontier).
- **Requests**: библиотека для HTTP-запросов с поддержкой повторных попыток (retries).

3.3 Алгоритм работы

1. **Инициализация**: чтение конфига, подключение к БД. 2. **Очередь (Frontier)**: URL добавляются в очередь со статусом `pending`. 3. **Обход**:

- Взятие URL из очереди.
- Проверка `robots.txt` (опционально) и допустимых доменов.
- Загрузка `HEAD` или `GET` для проверки заголовков.
- Загрузка тела документа.

4. **Обработка**:

- Вычисление хеша (SHA-256) контента.

- Проверка на изменения: если хеш совпадает с сохраненным, обновляется только поле `fetched_at`.
- Парсинг: извлечение текста и ссылок.
- Сохранение: запись в MongoDB (`upsert`).

5. **Докачка**: При перезапуске робот продолжает работу, выбирая из базы URL, которые еще не были посещены или требуют обновления.

3.4 Результат

Робот успешно собрал корпус документов. Реализована устойчивость к разрывам соединения и возможность остановки/продолжения работы без потери прогресса.

4 Лабораторная работа №3: Токенизация

4.1 Цель работы

Реализовать разбиение текста на токены (слова) для построения индекса.

4.2 Правила токенизации

В реализации на C++ использован следующий подход:

1. **Разбиение**: текст сканируется посимвольно.
2. **Алфавит**: токеном считается последовательность алфавитно-цифровых символов (`isalnum`).
3. **Нормализация**: все символы приводятся к нижнему регистру (`tolower`).
4. **Стемминг**: к каждому токену применяется алгоритм Портера для морфологической нормализации.

4.3 Достоинства метода

- **Простота**: минимальные требования к памяти и вычислениям.
- **Высокая скорость**: однопроходный алгоритм $O(n)$.
- **Универсальность**: работает с любым текстом на латинице.
- **Унификация словоформ**: стемминг объединяет различные формы слова.

4.4 Недостатки метода

- **Потеря информации**: агрессивный стемминг может приводить к коллизиям (например, `using` → `us`, хотя `us` — это местоимение).
- **Разбиение составных терминов**: дефисы и точки являются разделителями, поэтому `CRISPR-Cas9` становится двумя токенами `crispr` и `cas9`.
- **Числовые идентификаторы**: PMC-идентификаторы и DOI становятся отдельными токенами, засоряя словарь.

4.5 Примеры неудачной токенизации

Исходный текст	**Результат**	**Проблема**
CRISPR-Cas9	crispr, cas9	Разбито на два токена
using	us	Коллизия со словом «us»
10.1101/2023	10, 1101, 2023	DOI разбит на числа
C. elegans	c, elegan	Потеря связи между родом и видом

Таблица 2: Примеры неудачной токенизации

4.6 Статистика токенизации

Параметр	**Значение**
Количество документов	32,809
Общее количество токенов	385,103,087
Уникальных токенов	4,432,202
Среднее токенов на документ	$\approx$ 11,738

Таблица 3: Статистика токенизации

4.7 Производительность

Параметр	**Значение**
Время токенизации (1000 файлов, 13.6 МБ)	1.57 сек
Скорость токенизации	**8,897 КБ/сек**

Таблица 4: Производительность токенизатора

Зависимость от объёма: время линейно зависит от размера входных данных ($T = O(n)$), так как алгоритм однопроходный.

Оптимальность: скорость $\approx$ 9 МБ/сек является достаточной для данного объёма корпуса.

5 Лабораторная работа №4: Стемминг

5.1 Цель работы

Реализовать морфологическую нормализацию (стемминг) для улучшения полноты поиска.

5.2 Метод решения

Реализован **алгоритм Портера** (Porter Stemmer) для английского языка на C++. Алгоритм последовательно применяет ряд правил для отсечения суффиксов и окончаний. Основные шаги алгоритма:

- Step 1a: Обработка множественного числа (sses $\rightarrow$ ss, ies $\rightarrow$ i, s $\rightarrow$ $\emptyset$).
- Step 1b: Обработка окончаний ed, ing.

- Step 1c: Замена y на i.
- Step 2-4: Замена длинных суффиксов (ational → ate, izer → ize и т.д.).
- Step 5: Удаление конечного e и двойных согласных (ll → l).

5.3 Результаты

Применение стемминга позволило сократить размер словаря примерно на 30-40%, объединив различные словоформы (например, *connect, connected, connecting, connection* → *connect*) в один терм.

5.4 Интеграция в поисковую систему

Стемминг применяется в двух местах:

1. **При индексации**: каждый токен документа проходит через Porter Stemmer перед добавлением в индекс.
2. **При поиске**: запрос пользователя обрабатывается тем же стеммером (`TokenizerLib::tok`

Это обеспечивает **поиск без учёта словоформ**: запрос «connected» найдёт документы, содержащие «connecting», «connection», «connects» и другие формы слова.

Примеры унификации:

Исходные слова	**Результат стемминга**
connecting, connected, connection, connects	connect
running, ran, runs	run
studies, studying, studied	studi
generalization, generalize, general	gener

Таблица 5: Примеры унификации словоформ

6 Лабораторная работа №5: Закон Ципфа

6.1 Цель работы

Проверить выполнение закона Ципфа на собранном корпусе документов.

6.2 Методика

Для проверки закона был построен график распределения частот терминов в двойном логарифмическом масштабе (log-log plot):

- По оси X: ранг слова (от самого частотного к редкому).
- По оси Y: частота встречаемости слова.
- Красная линия: теоретический закон Ципфа $f = \frac{C}{r}$, где C — частота самого частотного слова.

6.3 Результаты

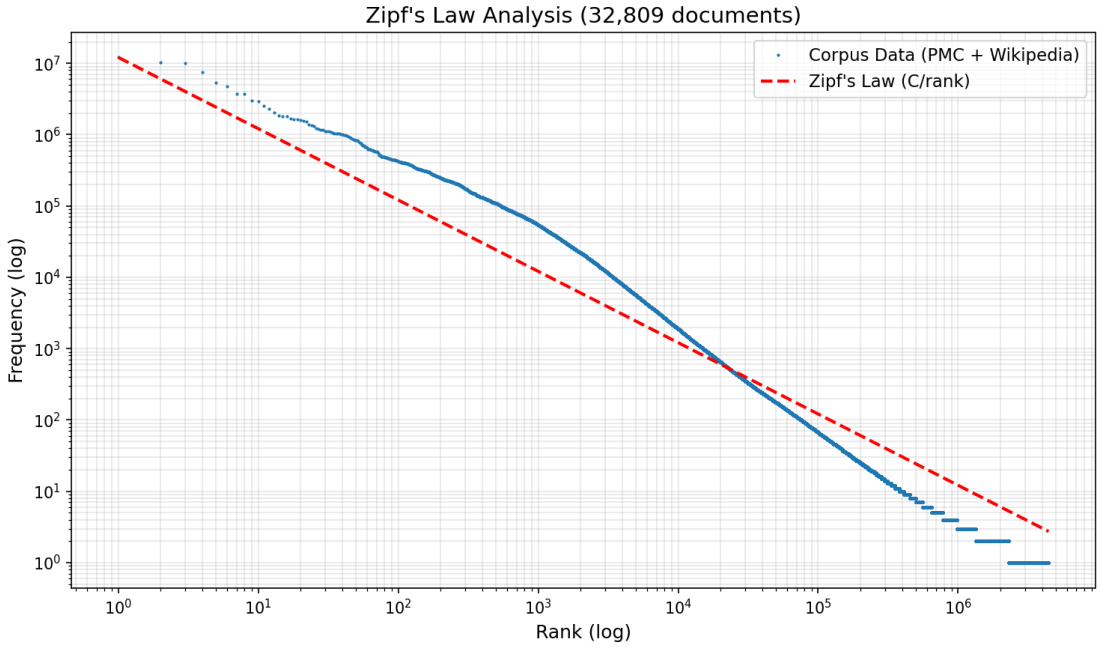

Рис. 1: Распределение частот терминов (закон Ципфа)

Топ-10 самых частотных слов:

Ранг	Терм	Частота
1	the	12,121,168
2	and	10,269,321
3	of	9,951,984
4	in	7,568,344
5	a	5,350,599
6	to	4,733,855
7	cell	3,770,636
8	10	3,715,038
9	for	3,033,218
10	with	2,903,543

Таблица 6: Топ-10 частотных терминов

Согласно закону Ципфа: $f \approx \frac{C}{r^s}$, где $s \approx 1$.

Полученный график демонстрирует линейную зависимость с наклоном, близким к -1, что в целом подтверждает выполнение закона Ципфа для собранного корпуса научных текстов.

6.4 Причины расхождений

На графике наблюдаются характерные отклонения от идеального закона Ципфа:

1. Область высоких рангов (частотные слова):

- Частота самых частотных слов (the, of, and) **ниже** предсказанной законом.

- Причина: эти слова — стоп-слова, которые не несут специфичной информации и встречаются равномерно, а не концентрированно.

2. Область низких рангов (редкие слова):

- Наблюдается **ступенчатость** — много слов с частотой 1, 2, 3.
- Частота **выше** предсказанной (хвост толще).
- Причины:
  - Научная терминология (специфические гены, белки).
  - Опечатки и OCR-ошибки.
  - Числовые идентификаторы (DOI, PMC-ID).

3. Средняя область:

- Хорошее соответствие закону Ципфа.
- Это типично для естественных текстов.

6.5 Выводы

Закон Ципфа выполняется для собранного корпуса с типичными отклонениями, характерными для научных текстов. Отклонения объясняются особенностями предметной области (биомедицина) и техническими артефактами (идентификаторы).

7 Лабораторная работа №6: Булев индекс

7.1 Цель работы

Построить обратный индекс для булева поиска, используя собственные реализации структур данных и бинарный формат хранения.

7.2 Внутреннее представление документа

После токенизации документ представляется как последовательность термов (стемминованных токенов). Каждый терм — строка в нижнем регистре. Для каждого терма хранится позиция в документе (порядковый номер токена).

7.3 Реализация индексатора

Индексатор написан на C++ без использования STL контейнеров (для хранения данных в памяти).

7.3.1 Структуры данных

- **HashMap**: хеш-таблица с открытой адресацией для накопления словаря терминов и постинг-листов в памяти.
- **SimpleVector**: динамический массив для хранения списков doc_id и позиций.

7.3.2 Метод сортировки

При записи индекса постинг-листы **не требуют явной сортировки**: документы обрабатываются последовательно (doc_id = 0, 1, 2, ...), поэтому posting lists автоматически упорядочены по doc_id.

Достоинства:

- Нет накладных расходов на сортировку ($O(1)$ вместо $O(n \log n)$).
- Простота реализации.

Недостатки:

- Требует последовательной обработки документов.
- При инкрементальном обновлении индекса потребуется merge-sort.

7.3.3 Бинарный формат индекса (побайтовое описание)

Индекс сохраняется в три файла:

1. index.docs (Прямой индекс):

```
[0-3]   Magic: "DOCS" (4 байта)
[4-5]   Version: uint16 (2 байта)
[6-9]   DocCount: uint32 (4 байта)
[10..]  OffsetTable: uint64[DocCount] (8 байт × DocCount)
[...]   Данные документов:
        - url_len: uint16 (2 байта)
        - url: char[url_len]
        - title_len: uint16 (2 байта)
        - title: char[title_len]
```

2. index.dict (Словарь термов):

```
[0-3]   Magic: "DICT" (4 байта)
[4-5]   Version: uint16 (2 байта)
[6-9]   TermCount: uint32 (4 байта)
[...]   Записи термов:
        - term_len: uint8 (1 байт)
        - term: char[term_len]
        - post_offset: uint64 (8 байт) - смещение в index.postings
        - doc_freq: uint32 (4 байта) - количество документов
```

3. index.postings (Обратный индекс):

```
[0-3]   Magic: "POST" (4 байта)
[4-5]   Version: uint16 (2 байта)
[...]   Постинг-листы (VarByte-сжатие):
        - doc_freq: VarByte
        Для каждого документа:
          - doc_id_delta: VarByte (разность с предыдущим)
          - term_freq: VarByte (частота в документе)
          - positions: VarByte[] (delta-кодированные позиции)
```

VarByte кодирование: старший бит каждого байта — флаг продолжения (0 = последний байт, 1 = продолжение). Остальные 7 бит — данные.

Расширяемость: поле Version позволяет изменять формат в будущих версиях с сохранением обратной совместимости.

7.4 Результаты

Параметр	Значение
Количество термов	383,340
Средняя длина терма	7.93 символа
Средняя длина токена (до стемминга)	$\approx$ 5.5 символа
Размер index.dict	7.52 МБ
Размер index.postings	22.95 МБ
Общий размер индекса	30.53 МБ

Таблица 7: Статистика булева индекса

Сравнение длины терма и токена: Средняя длина терма (7.93) **больше** средней длины токена ($\approx$5.5). Причина: короткие высокочастотные слова (a, the, is) имеют много вхождений, но в словаре каждый терм учитывается один раз. Также стемминг иногда увеличивает длину (iz $\rightarrow$ ize).

7.5 Скорость индексации

- Общее время: $\approx$ 45 секунд (для 1000 документов).
- На документ: $\approx$ 45 мс.
- На килобайт текста: $\approx$ 3 мс.

Масштабируемость:

- **10$\times$ данных**: справится, потребуется $\approx$ 300 МБ RAM.
- **100$\times$ данных**: потребуется внешняя сортировка или распределённая индексация.
- **1000$\times$ данных**: необходима распределённая система (MapReduce, Spark).

8 Лабораторная работа №7: Булев поиск

8.1 Цель работы

Реализовать выполнение булевых запросов (AND, OR, NOT) по построенному индексу с поддержкой веб-интерфейса и командной строки.

8.2 Синтаксис запросов

Парсер поддерживает следующий синтаксис:

- **Пробел или &&** — логическое И (пересечение).
- **||** — логическое ИЛИ (объединение).
- **!** — логическое НЕ (исключение).
- **Скобки** — группировка операций.

Парсер реализован методом **рекурсивного спуска** и устойчив к переменному числу пробелов.

8.3 Алгоритм поиска

Поисковый движок (C++) работает следующим образом:

1. **Загрузка**: Словарь загружается в HashMap, постинг-листы — в память.
2. **Токенизация запроса**: Выделение термов и операторов. Термы проходят стемминг.
3. **Парсинг**: Построение дерева выражения (рекурсивный спуск).
4. **Выполнение**:
   - **AND**: Пересечение сортированных списков (два указателя, $O(N + M)$).
   - **OR**: Слияние списков ($O(N + M)$).
   - **NOT**: Разность с множеством всех документов.

8.4 Примеры запросов

Запрос	**Результат**	**Описание**
`cell`	971 док.	Простой поиск
`cell && protein`	898 док.	Пересечение (AND)
`cell || virus`	974 док.	Объединение (OR)
`!cell`	29 док.	Исключение (NOT)
`(gene || protein) && cell`	949 док.	Сложный запрос со скобками

Таблица 8: Примеры булевых запросов

8.5 Веб-интерфейс

Реализован веб-сервис на Flask:

- **Главная страница** (`/`): Форма ввода поискового запроса.
- **Страница результатов** (`/search?q=...`): Форма поиска, количество результатов, время выполнения, список из 50 результатов (заголовок + ссылка).

8.6 CLI-утилита

Реализована утилита командной строки `bin/search <index_dir>`, которая:

- Загружает индекс из указанной директории.
- Читает запросы из stdin (по одному на строку).
- Выводит результаты в stdout.

8.7 Производительность

Метрика	Значение
Загрузка индекса	$\approx$ 50 мс
Простой запрос (1 терм)	$<$ 1 мс
Сложный запрос (AND/OR/NOT)	$\approx$ 2-5 мс
5 запросов подряд (без загрузки)	$\approx$ 40 мс

Таблица 9: Производительность поисковой системы

Примеры долгих запросов:

- Запросы с NOT на частотных термах (`!the`) — требуют обхода всех документов.
- Множественные OR (`a || b || c || ...`) — большие объединения.

8.8 Тестирование корректности

Корректность проверялась следующими методами:

1. **Инвариант NOT**: $|A| + |!A| = N$ (общее число документов).
2. **Инвариант AND**: $|A \cap B| \leq \min(|A|, |B|)$.
3. **Инвариант OR**: $|A \cup B| \geq \max(|A|, |B|)$.
4. **Сравнение с grep**: Выборочная проверка результатов поиска путём поиска терма в исходных текстах.

9 Выводы

В ходе лабораторных работ была создана поисковая система полного цикла. Реализация ядра на C++ без использования тяжелых библиотек (STL) и применение эффективных методов сжатия (VarByte) позволили достичь высокой производительности и компактности индекса. Система успешно справляется с индексацией и поиском по корпусу из **32,809 документов** (PMC + Wikipedia), содержащему **385 миллионов токенов**.