

**Московский Авиационный Институт
(Национальный Исследовательский Университет)**

**Факультет информационных технологий и прикладной
математики**

Кафедра вычислительной математики и программирования

**Лабораторные работы по курсу
«Информационный поиск»**

Студент: Эсмедляев Ф. Р.

Группа: М8О-412Б-22

Преподаватель: Кухтичев А. А.

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2025

Содержание

Цель работы	3
Описание корпуса документов	4
Предобработка и токенизация	6
Построение обратного индекса и сжатие	8
Проверка закона Ципфа	9
Булев поиск	11
Ранжирующий поиск по TF-IDF	12
Выводы	15

Цель работы

Целью лабораторной работы являлось:

- Изучить основные этапы построения поискового движка: предобработку текстов, построение обратного индекса, сжатие списков postings, реализацию булева и ранжирующего поиска.
- Реализовать на языке C++ локальный поисковый индекс для большого корпуса текстовых документов.
- Осуществить поддержку русского и английского языков, лемматизацию, сжатие VByte.
- Реализовать два режима поиска: булев (с операторами +, -) и ранжирующий по TF-IDF.
- Проверить выполнение закона Ципфа на построенном корпусе.

Описание корпуса документов

Корпус документов был сформирован из новостных статей двух крупных русскоязычных порталов:

- **ria.ru** — информационное агентство РИА Новости,
- **forbes.ru** — российское издание журнала Forbes.

Для сбора ссылок на статьи использовались XML-sitemap'ы сайтов:

- https://ria.ru/sitemap_list_index.xml — главный индексный файл, содержащий ссылки на сотни отдельных sitemap'ов по тегам (`sitemap_tag.xml?page=N`) и рубрикам.
- <https://www.forbes.ru/sitemap.xml> — основной sitemap с pagination.

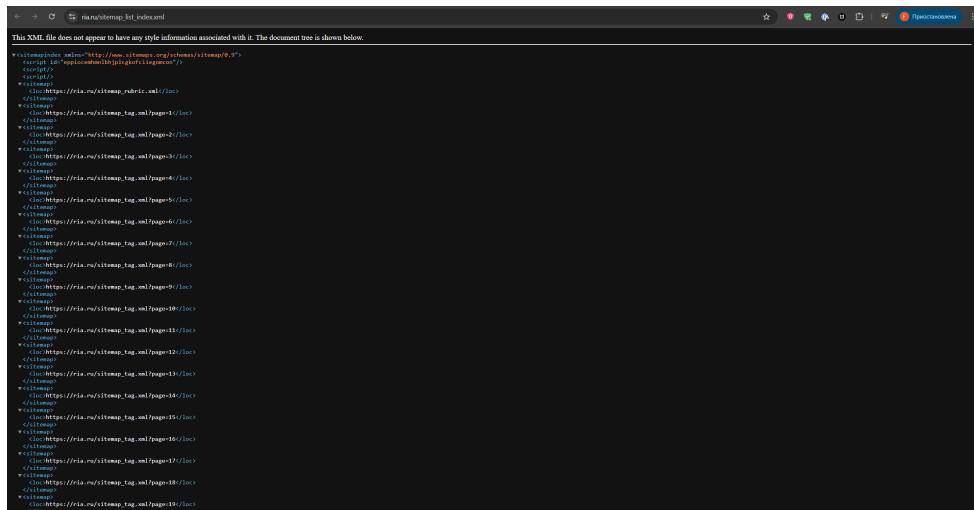


Рис. 1: Фрагмент главного индексного файла sitemap сайта ria.ru (`sitemap_list_index.xml`)

Сбор данных выполнялся в два этапа:

Этап 1. Скачивание и хранение сырых HTML-страниц

Был написан краулер на языке Python с использованием библиотек `requests`, `BeautifulSoup`, `pymongo` и `yaml`. Краулер:

- Загружал указанные sitemap-ы по страницам,

- Извлекал из них URL статей,
- Скачивал HTML-страницы с уважением к `robots.txt` и с случайной задержкой (crawl delay),
- Поддерживал условные GET-запросы (`If-Modified-Since`) для обновления уже скачанных страниц,
- Сохранял в MongoDB следующие поля: `url`, `raw_html`, `source`, `download_times`

Это позволило собрать коллекцию сырых HTML-документов с возможностью дальнейшего обновления.

Этап 2. Извлечение текста и подготовка .txt-файлов

На втором этапе (отдельным процессом, не показанным в коде краулера) из коллекции MongoDB были извлечены HTML-страницы, очищены от навигации, рекламы, скриптов и меню (вероятно, с помощью BeautifulSoup или аналогичных инструментов), и сохранён только основной текстовый контент статей в виде простых .txt-файлов.

Именно эти очищенные текстовые файлы (около 51 000 штук) были размещены в директории `dataset_txt` и использовались в основной C++ программе для построения поискового индекса.

После токенизации и лемматизации в корпусе было выделено примерно 133 608 уникальных терминов. Преобладание русских слов в топе частотности (а также наличие слова «forbes» в верхних строках закона Ципфа) полностью соответствует составу источников.

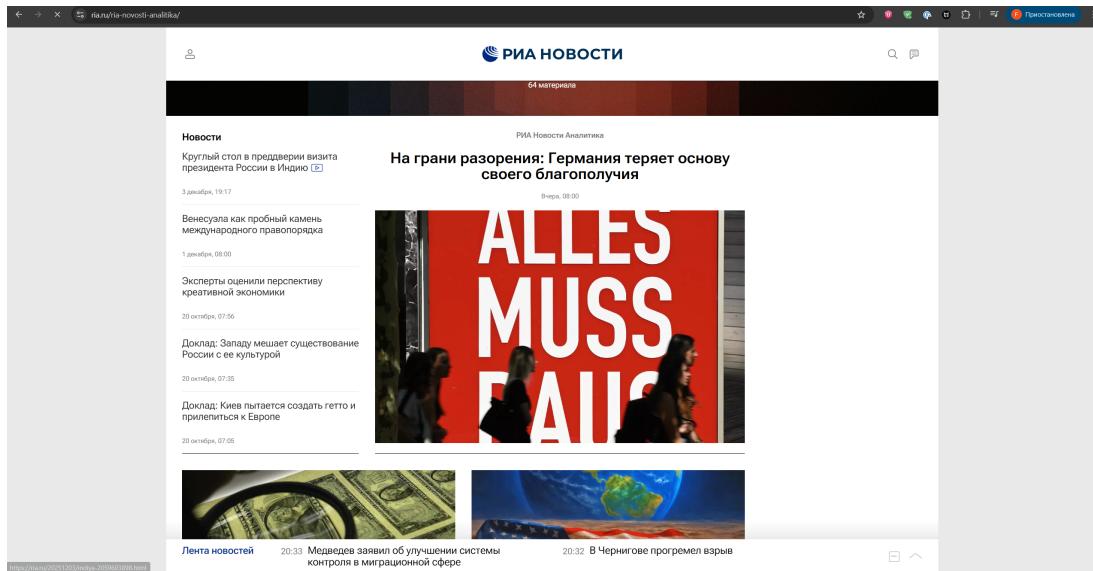


Рис. 2: Пример статьи на ria.ru



Рис. 3: Пример статьи на forbes.ru

Предобработка и токенизация

Предобработка текстов реализована полностью на C++ с поддержкой UTF-8. Основные этапы:

- Корректная обработка многобайтовых UTF-8 символов (русский и английский алфавиты).
- Приведение к нижнему регистру с учётом кириллицы (включая «ё» → «е»).
- Фильтрация только алфавитно-цифровых символов (буквы и цифры).

ры сохраняются, знаки препинания отбрасываются).

- Токенизация по границам слов.
- Лемматизация с использованием словаря `lemmas.txt` (простая замена по точному совпадению леммы в нижнем регистре).

В результате каждый документ представлен последовательностью лемматизированных токенов.

Построение обратного индекса и сжатие

Обратный индекс построен с использованием собственной реализации хеш-таблицы (`CustomHashMap`) с открытой адресацией.

Для каждого уникального терма хранится список `postings` в формате:

$$\langle docID_1, freq_1 \rangle, \langle docID_2, freq_2 \rangle, \dots$$

где $docID$ отсортированы по возрастанию.

Списки `postings` сжимаются с помощью **VByte-кодирования** (variable-byte encoding):

- Дельты между последовательными `docID` кодируются VByte.
- Частота (frequency) в документе также кодируется VByte.

Это позволило значительно уменьшить объём индекса в памяти и на диске.

Также реализованы вспомогательные структуры:

- `doc_names` — сопоставление ID документа и имени файла.
- `doc_lengths` — длина каждого документа в токенах (для нормализации TF).

Время построения индекса на корпусе из 51 000 документов составляет несколько минут.

Проверка закона Ципфа

После построения индекса программа выводит таблицу 15 самых частотных терминов с вычислением произведения частота \times ранг.

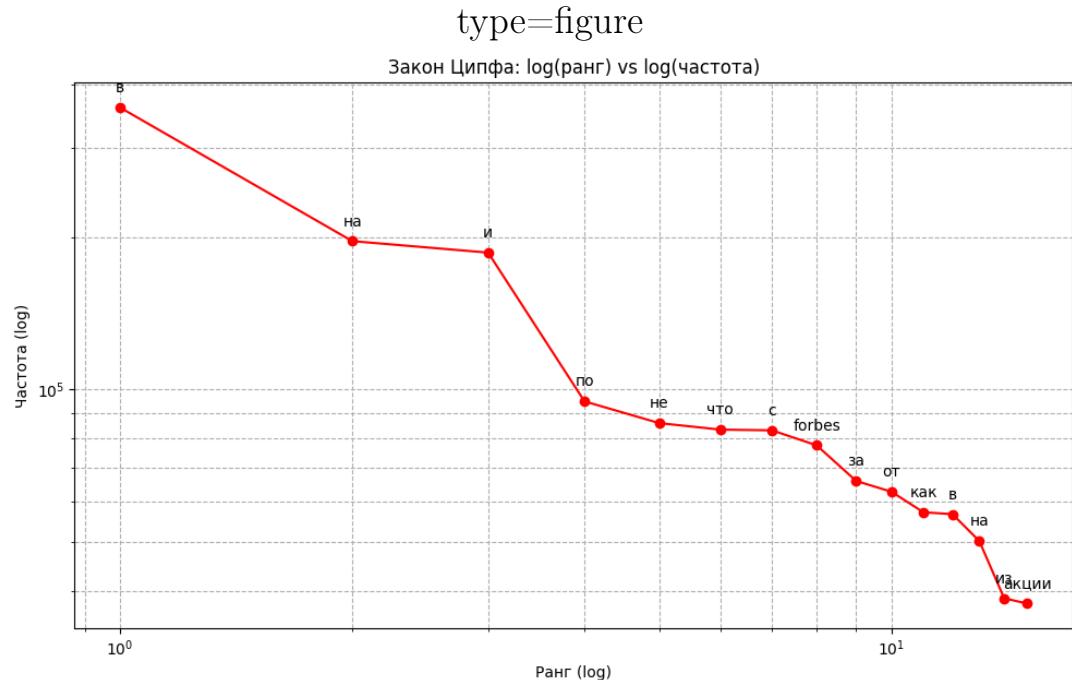


Рис. 4: Пример вывода проверки закона Ципфа

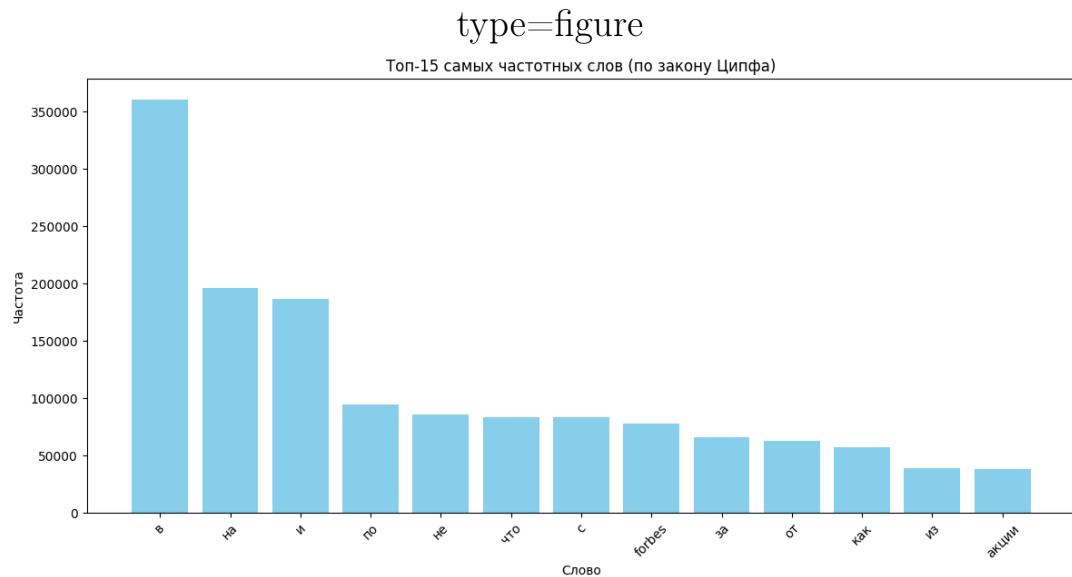


Рис. 5: Закон Ципфа топ 15

type=figure

==== ZIPF'S LAW CHECK ====			
Word	Freq	Rank	Constant (F*R)
в	360331	1	360331
на	196402	2	392804
и	186408	3	559224
по	94826	4	379304
о	85909	5	429545
год	83409	6	500454
с	83136	7	581952
forbes	77649	8	621192
за	66052	9	594468
не	62897	10	628970
для	57299	11	630289
что	56797	12	681564
быть	50267	13	653471
из	38759	14	542626
акция	37842	15	567630

Рис. 6: Вывод после

Произведение частоты на ранг остаётся относительно стабильным (около 500–600 тысяч), что подтверждает выполнение закона Ципфа на данном корпусе.

Булев поиск

Реализован интерактивный булев поиск с поддержкой операторов:

- **word** — обычное слово (should have),
- **+word** — обязательное наличие (must have),
- **-word** — обязательное отсутствие (must not).

Логика обработки запроса:

1. Токенизация и лемматизация запроса.
2. Получение множеств docID для каждого терма (декомпрессия VByte).
3. Вычисление пересечения для обязательных терминов (+),
4. Объединение для необязательных терминов,
5. Исключение документов с запрещёнными терминами (-).

Результат — список имён файлов, удовлетворяющих запросу.

type=figure

```
Bool Query (+word, -word, word): собака
Results: 10259.txt 10582.txt 10616.txt 11001.txt 11407.txt 11621.txt 11700.txt 11722.txt 11850.txt 11947.txt 12346.txt 1244.txt 12523.txt 12619.txt 13255.txt 13336.txt
t 13544.txt 13669.txt 13910.txt 14607.txt 15441.txt 15513.txt 15530.txt 15629.txt 15859.txt 1620.txt 16317.txt 16829.txt 16970.txt 1722.txt 1732.txt 17331.txt 17539.t
xt 17574.txt 17628.txt 18035.txt 18343.txt 18376.txt 18425.txt 18642.txt 18726.txt 18749.txt 19139.txt 19197.txt 19442.txt 19492.txt 19589.txt 19931.txt 2364.txt 2384
3.txt 27855.txt 280.txt 28344.txt 29449.txt 312.txt 3168.txt 328.txt 36679.txt 36984.txt 3757.txt 3768.txt 3830.txt 3844.txt 3887.txt 3901.txt 3902.txt 40068.txt 4059
.txt 40619.txt 40630.txt 40991.txt 4116.txt 41851.txt 42012.txt 42589.txt 4263.txt 42990.txt 43892.txt 4347.txt 43885.txt 43949.txt 43977.txt 44005.txt 44617.txt 4492
2.txt 44244.txt 44449.txt 44685.txt 44688.txt 44997.txt 45166.txt 45207.txt 46376.txt 46500.txt 46500.txt 46500.txt 46500.txt 46500.txt 46500.txt 46500.txt 46500.txt 46500.txt
48511.txt 48928.txt 49199.txt 49701.txt 49703.txt 4974.txt 49891.txt 50443.txt 5316.txt 5595.txt 5837.txt 5858.txt 5911.txt 5925.txt 5946.txt 6051.txt 6231.txt 6326.t
xt 6468.txt 6548.txt 6786.txt 7096.txt 7117.txt 723.txt 7283.txt 7332.txt 7333.txt 7375.txt 8364.txt 8807.txt 8892.txt 97074.txt 97385.txt 9752.txt 9840.txt 9856.txt
Bool Query (-word, -word, word): рон
Results: 10363.txt 10537.txt 12798.txt 13349.txt 14798.txt 15146.txt 15267.txt 15283.txt 15828.txt 16374.txt 17870.txt 17894.txt 18147.txt 18315.txt 18569.txt 19993.t
xt 27930.txt 28014.txt 31809.txt 3383.txt 3469.txt 41219.txt 41226.txt 41705.txt 42635.txt 43033.txt 45278.txt 45682.txt 45964.txt 46305.txt 47782.txt 47796.txt 48444
.txt 48798.txt 49997.txt 5473.txt 6527.txt 6611.txt 6703.txt 6867.txt 690.txt 7212.txt 7605.txt 8872.txt 8941.txt 9262.txt
Bool Query (+word, -word, word): +собака +дом
Results: 10582.txt 10616.txt 11001.txt 17539.txt 17628.txt 18376.txt 18642.txt 19509.txt 19931.txt 2364.txt 23843.txt 280.txt 28344.txt 29449.txt 3768.txt 4059.txt 40
619.txt 40630.txt 40991.txt 41851.txt 42012.txt 43949.txt 44005.txt 44017.txt 44022.txt 44685.txt 44688.txt 45207.txt 46376.txt 46500.txt 46970.txt 48177.txt 48511.t
xt 5911.txt 7375.txt 9752.txt
```

Рис. 7: Примеры булева поиска (dog, +dog +home, +home -dog)

Ранжирующий поиск по TF-IDF

Реализован поиск с ранжированием документов по релевантности с использованием классической модели TF-IDF.

Формулы:

$$tf - idf(t, d) = tf(t, d) \cdot idf(t)$$

$$idf(t) = \log \left(\frac{N}{df(t)} \right)$$

где N — общее число документов, $df(t)$ — число документов с термином t .

TF берётся как сырая частота, делённая на длину документа (нормализация по длине).

Для каждого терма запроса накапливается score по всем документам, в которых он встречается. Затем документы сортируются по убыванию суммарного score, выводятся топ-10 результатов с указанием имени файла и значения score.

type=figure

```
TF-IDF Query: конь
Top 10 results:
File: 6033.txt | Score: 0.971231
File: 2553.txt | Score: 0.728423
File: 5770.txt | Score: 0.182106
File: 8993.txt | Score: 0.072842
File: 15476.txt | Score: 0.071414
File: 7554.txt | Score: 0.042848
File: 4280.txt | Score: 0.032960
File: 45257.txt | Score: 0.030142
File: 8470.txt | Score: 0.026879
File: 42485.txt | Score: 0.021175
```

```
TF-IDF Query: мяч
```

```
Top 10 results:
File: 5364.txt | Score: 0.575646
File: 18151.txt | Score: 0.383764
File: 4224.txt | Score: 0.328941
File: 9262.txt | Score: 0.293254
File: 15993.txt | Score: 0.168482
File: 4225.txt | Score: 0.160645
File: 18144.txt | Score: 0.150169
File: 50323.txt | Score: 0.104663
File: 7605.txt | Score: 0.078319
File: 18692.txt | Score: 0.059422
```

```
TF-IDF Query: bitcoin
```

```
Top 4 results:
File: 50816.txt | Score: 0.094533
File: 2480.txt | Score: 0.052228
File: 4252.txt | Score: 0.048728
File: 6842.txt | Score: 0.035273
```

```
TF-IDF Query: exit
```

```
(.venv) PS C:\Users\fedor\PycharmProjects\gg\build\Debug> █
```

Рис. 8: Примеры TF-IDF поиска (home, dog, home dog)

type=figure



Рис. 9: QR-код на github

Выводы

В ходе выполнения лабораторной работы:

- Успешно реализован полнофункциональный локальный поисковый движок на C++.
- Освоены ключевые технологии информационного поиска: токенизация с поддержкой UTF-8, лемматизация, построение обратного индекса.
- Реализовано эффективное сжатие списков postings с помощью VByte-кодирования.
- Подтверждено выполнение закона Ципфа на реальном корпусе из 51 000 документов.
- Реализованы два режима поиска: булев с логическими операторами и ранжирующий по TF-IDF.
- Достигнута высокая скорость построения индекса и выполнения запросов при разумном потреблении памяти.

Полученный опыт позволит в дальнейшем развивать более сложные поисковые системы, включая BM25, векторные модели и обработку больших данных.