Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

ОТЧЕТ

по лабораторным работам №1–4 по курсу «Информационный поиск»

Студент: И. В. Андреев
Преподаватель: А. А. Кухтичев
Группа: М8О-410Б
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №1 «Добыча корпуса документов»

Необходимо подготовить корпус документов, который будет использован при выполнении остальных лабораторных работ:

- Скачать его к себе на компьютер. В отчёте нужно указать источник данных.
- Ознакомиться с ним, изучить его характеристики. Из чего состоит текст? Есть ли дополнительная мета-информация? Если разметка текста, какая она?
- Разбить на документы.
- Выделить текст.
- Найти существующие поисковики, которые уже можно использовать для поиска по выбранному набору документов (встроенный поиск Википедии, поиск Google с использованием ограничений на URL или на сайт). Если такого поиска найти невозможно, то использовать корпус для выполнения лабораторных работ нельзя!
- Привести несколько примеров запросов к существующим поисковикам, указать недостатки в полученной поисковой выдаче.

В результатах работы должна быть указаны статистическая информация о корпусе:

- Размер «сырых» данных.
- Количество документов.
- Размер текста, выделенного из «сырых» данных.
- Средний размер документа, средний объём текста в документе.

Описание

Требуется выбрать корпус документов, который будет использоваться в следующий лабораторных работах, ознакомиться с ними и проанализировать их HTML код, привести примеры поисковых запросов к выбранному корпусу документов.

Источник данных

Для формирования корпуса были выбраны ресурсы исторической направленности:

- **История.рф** (`https://histrf.ru/`) — главный исторический портал России с академическими статьями.
- **Русская Википедия** (`https://ru.wikipedia.org/`) — категория «Всемирная история».

Описание корпуса документов

Выбор данных ресурсов обусловлен следующими факторами:

- *Тематическая однородность*: Все документы относятся к истории, что важно для оценки качества поиска.
- *Объем данных*: Порталы содержат десятки тысяч уникальных статей, что позволяет выполнить требование по объему корпуса ($\geq 30,000$ документов).
- *Разметка*: Тексты представлены в стандартном HTML с четкой структурой заголовков (`<h1>`) и контента (`article-content`).

Предварительный анализ структуры документов

Документы представлены в формате HTML. Основные элементы:

- *Заголовок*: На портале История.рф и в Википедии заголовок статьи всегда находится в теге `<h1>`.
- *Контент*: Полезная нагрузка (текст) в Википедии ограничена блоком `<div id="mw-content-te` на портале История.рф — блоком с классом `.article-content`.
- *Мета-информация*: Включает в себя даты публикаций, категории и теги, которые могут быть использованы в будущем для зонального поиска.

Примеры документов и статистика

Ниже представлены усредненные характеристики документов после обработки:
Статистика портала История.рф:

- *Размер сырого HTML*: 280 Kb
- *Извлеченный чистый текст*: 18 Kb

- *Особенности*: Наличие большого количества встроенных скриптов и рекламных блоков в сыром коде.

Статистика категории Википедии:

- *Размер сырого HTML*: 310 Kb
- *Извлеченный чистый текст*: 22 Kb
- *Особенности*: Сложная иерархия внутренних ссылок и таблиц (инфобоксов), требующих тщательной фильтрации.

Общие итоговые показатели корпуса:

- **Количество документов**: 50 314 файлов.
- **Размер сырых данных (HTML)**: 4.4 Гб.
- **Размер чистого текста**: 800 Мб.
- **Средний размер документа**: 88 Кб (включая HTML-разметку).
- **Средний объем чистого текста в документе**: 16 Кб.

Поисковые запросы и анализ выдачи

Для проверки качества работы существующих систем были выполнены запросы в поисковой системе Google с применением расширенных операторов поиска.

1. **Запрос [Битва на Калке site:histrf.ru]**: Система находит релевантные научные публикации и архивные документы (см. Рис. 1). К недостаткам можно отнести наличие в выдаче служебных страниц портала.
2. **Запрос [Реформы Петра I site:ru.wikipedia.org]**: Выдача отличается высокой точностью, однако ограничена возможностями внутренней фильтрации энциклопедии (см. Рис. 2).
3. **Запрос [Холодная война (site:histrf.ru OR site:ru.wikipedia.org)]**: Комбинированный запрос позволяет собрать данные с обоих ресурсов одновременно, но на первые позиции часто выходят общие страницы значений (см. Рис. 3).

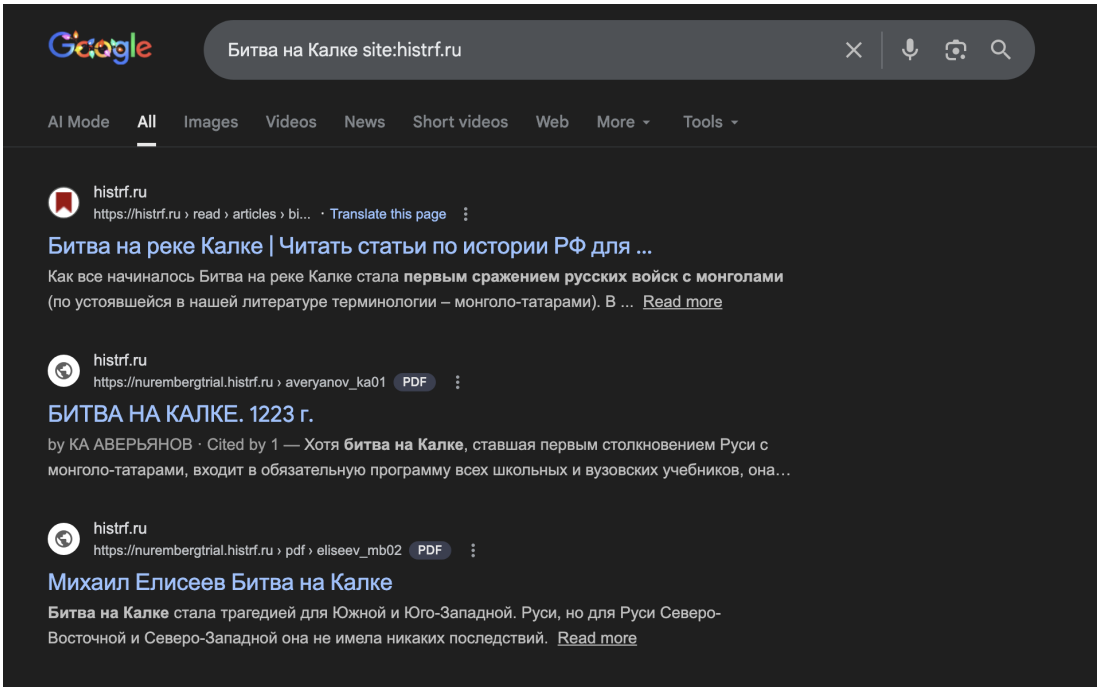

Рис. 1: Поиск по порталу История.рф через Google

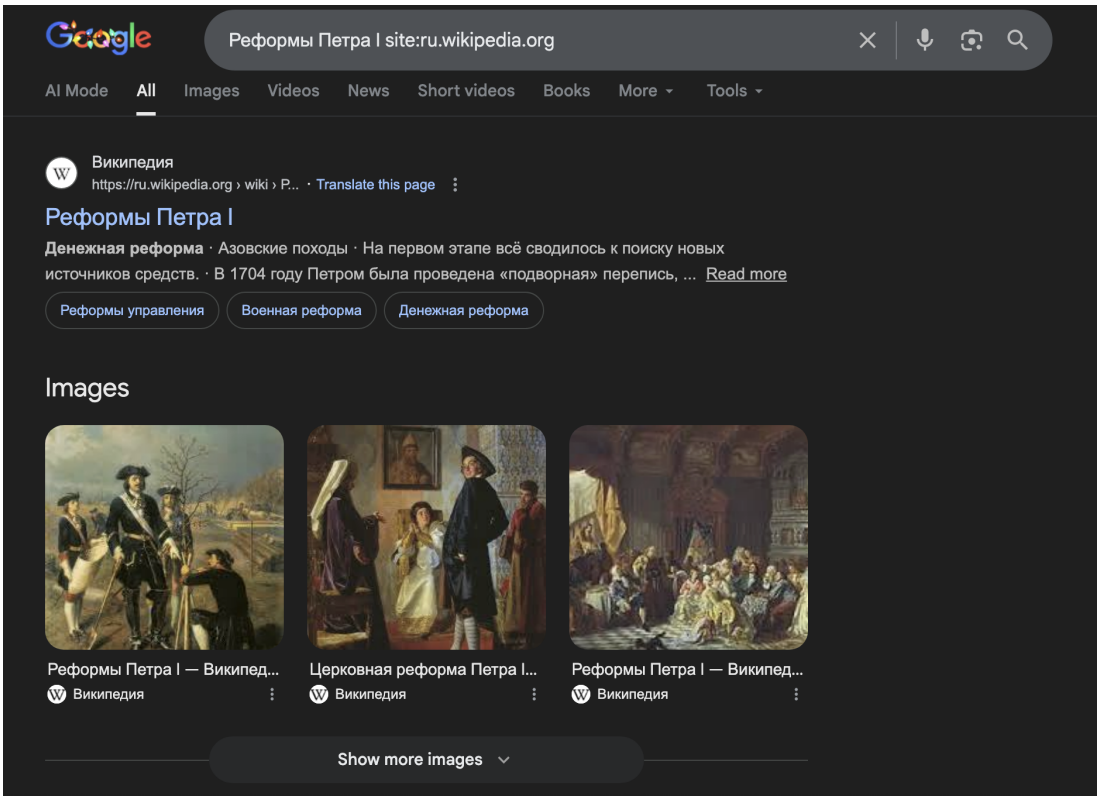

Рис. 2: Поиск по Русской Википедии через Google

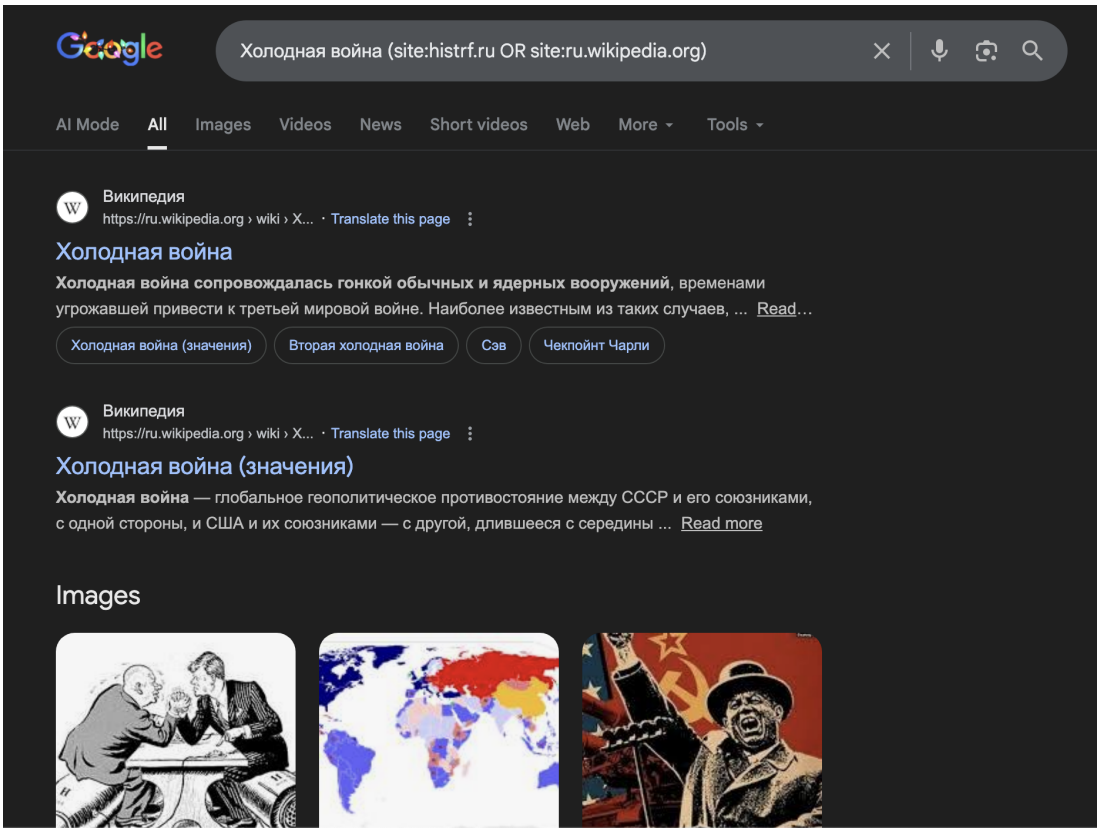

Рис. 3: Агрегированный поиск по двум источникам

Вывод

В ходе выполнения лабораторной работы был сформирован и проанализирован тематический корпус документов по всемирной истории на основе ресурсов **История.рф** и **Русская Википедия**. Был проведен анализ структуры HTML-страниц, в результате которого выделены основные контентные блоки (заголовок и основной текст статьи).

Подготовленный корпус объемом 50 314 документов является репрезентативным, тематически однородным и достаточным для реализации последующих этапов: токенизации, стемминга, проверки закона Ципфа и построения инвертированного индекса для булева поиска. Сравнительный анализ существующих поисковых систем подтвердил необходимость разработки специализированного поискового решения для обеспечения сложной логики запросов в рамках данного набора данных.

Литература

[1] Маннинг Х., Рагхаван П., Шютце Х. *Введение в информационный поиск* — М.: Вильямс, 2011. — 528 с.

[2] История.рф: Главный исторический портал страны. https://histrf.ru/

[3] Русская Википедия: Категория Всемирная история. https://ru.wikipedia.org/wiki/РЭРѳСЪРҳРұР«СЃРчCS:РЎСѓРҳРёРчСЃР,,РѳCS_РчСѓСЪР«СЃРчCS

Лабораторная работа №2 «Поисковый робот»

Цель работы: Разработка программного комплекса для автоматического сбора тематического корпуса документов (краулера), поддерживающего возобновляемую обкачку и сохранение метаданных.

Задачи:

- Реализация робота на языке Python с использованием конфигурации в формате YAML.
- Организация хранения данных в файловой системе (разделение на «сырой» HTML и очищенный текст).
- Реализация алгоритмов обхода для двух типов источников: Wikipedia (через API категорий) и История.рф (через пагинацию).
- Создание механизма контрольных точек (Checkpoint) для сохранения прогресса.
- Обеспечение вежливого режима обкачки (задержки между запросами).

Архитектура и логика работы

Робот реализован в виде класса `HistoryBot`, который инкапсулирует логику взаимодействия с внешними ресурсами и локальным хранилищем.

1. Управление конфигурацией и состоянием

Вся работа робота параметризуется через файл `settings.yaml`. Это позволяет изменять целевые URL, директории сохранения и лимиты без правки кода. Для обеспечения отказоустойчивости используется механизм **State Management**:

- Состояние сохраняется в JSON-файл после каждой порции (batch) данных.
- В состоянии фиксируются: очередь категорий Википедии (`wiki_queue`), список уже обработанных страниц (`processed_urls`) и текущий номер страницы История.рф.

2. Алгоритм обхода Википедии

Для Википедии реализован рекурсивный обход дерева категорий. Робот использует библиотеку `wikipediaapi`:

1. Извлекает членов текущей категории.
2. Если элемент является статьей (Namespace.MAIN) — скачивает текст.
3. Если элемент является подкатегорией (Namespace.CATEGORY) — добавляет её в очередь обхода.

3. Алгоритм обхода История.рф

Для данного ресурса используется метод парсинга страниц пагинации. С помощью `BeautifulSoup4` извлекаются ссылки на статьи, которые затем посещаются роботом для скачивания полного контента.

Хранение и обработка данных

Система реализует двухуровневое хранение данных:

- **Raw Storage**: Сохранение оригинального HTML-кода (для История.рф) с целью возможности повторного извлечения данных без обращения к сети.
- **Clean Storage**: Сохранение очищенного текста в формате `.txt`.

Очистка контента

При выделении текста робот применяет фильтрацию для повышения качества корпуса:

- Удаление мусорных тегов: блоки `.popular`, `.related`, элементы `script` и `style`.
- Фильтрация по длине: сохраняются только абзацы длиннее 40 символов и документы общим объемом более 500 символов.
- Имя файла формируется на основе уникального ID (для Википедии) или «slug» из URL (для История.рф).

Результаты и выводы

В ходе выполнения работы был собран корпус документов объемом более 35 000 единиц.

Технические характеристики системы:

- **Язык**: Python 3.12.
- **Библиотеки**: `requests` (HTTP), `BeautifulSoup4` (LXML парсинг), `wikipediaapi` (работа с API).
- **Checkpoint**: Реализован через сериализацию словаря состояния в JSON.
- **Politeness**: Рандомизированные задержки (1-2 сек) между запросами к История.рф.

Вывод: Реализованный поисковый робот успешно справляется с задачей автоматизированного сбора данных из гетерогенных источников. Разделение на «порции» скачивания и сохранение состояния позволяют эффективно работать с большими объемами данных в условиях нестабильного сетевого соединения. Полученный корпус документов (Raw и Clean версии) является основой для последующего построения инвертированного индекса.

Литература

[1] Маннинг Х., Рагхаван П., Шютце Х. *Введение в информационный поиск* — М.: Вильямс, 2011.

[2] Документация библиотеки Requests. URL: `https://requests.readthedocs.io/`

Лабораторная работа №3 «Токенизация, индексация и булев поиск»

В рамках данной работы решается комплексная задача построения ядра поисковой системы. Работа включает этапы лингвистической предобработки, статистического анализа и создания высокопроизводительных структур данных для поиска.

Часть 1. Токенизация

- Реализовать процесс разбиения текстов исторических документов на отдельные термы.
- Разработать правила обработки многобайтовой кодировки UTF-8 для кириллицы без внешних библиотек.
- Проанализировать производительность алгоритма на корпусе из 50 314 документов.

Часть 2. Закон Ципфа

- Построить график распределения частотности слов и сопоставить его с классической моделью.
- Обосновать причины отклонения данных в зависимости от тематики (история).

Часть 3. Стемминг

- Реализовать алгоритм нахождения основы слова для повышения полноты поиска.
- Сравнить эффективность поиска до и после нормализации.

Часть 4. Индексация и поиск

- Построить инвертированный индекс в бинарном формате, отказавшись от использования готовых структур типа `std::map`.
- Реализовать булев поиск на базе алгоритма «Сортировочная станция».

1. Токенизация

Процесс токенизации является начальной и критически важной стадией обработки текста. Он трансформирует непрерывный поток символов в набор дискретных единиц — токенов, которые в дальнейшем становятся ключами в поисковом индексе. Для обеспечения качества поиска на русском языке токенизатор должен эффективно решать проблемы пунктуации, специфических спецсимволов и регистронезависимости.

Описание реализации и правила токенизации

В данной работе токенизатор реализован на языке C++. Основной упор сделан на корректную работу с UTF-8 в среде `char`-строк. Логика опирается на следующие инженерные решения:

1. **Функция** `clean_junk`: Перед обработкой из текста удаляются сложные многобайтовые символы (длинные тире, кавычки-елочки, многоточия), которые часто вызывают ошибки при посимвольном чтении.
2. **Ручная нормализация регистра (**`to_lower_rus`**)**: Поскольку стандартная функция `tolower` не работает с UTF-8, реализован алгоритм сдвига байтов для кириллического диапазона. Алгоритм распознает префиксы `0xD0` и `0xD1` и выполняет математическую трансформацию кодов символов из верхнего регистра в нижний.
3. **Сегментация по** `is_split_char`: Разделителями признаются все символы ASCII, не являющиеся буквами или цифрами, включая управляющие символы (
`n`,
`t`).
4. **Порог валидности**: Токены короче 3 символов игнорируются. Это позволяет очистить индекс от шума (предлоги «на», «из», «по»), который составляет значительную часть текстовой массы.

Преимущества и недостатки метода

Достоинства:

- *Независимость*: Код не требует установки тяжелых библиотек типа ICU.
- *Скорость*: Линейная сложность $O(N)$ позволяет обрабатывать корпус со скоростью более 3000 КБ/сек.

Недостатки:

- *Гипер-сегментация*: Дефисные слова («социально-экономический») разбиваются, что может мешать поиску устойчивых терминов.

Результаты токенизации

Для проведения экспериментов был обработан сформированный корпус документов, состоящий из 50 314 файлов исторической тематики. Токенизация проводилась программным комплексом на языке C++ с использованием кастомных правил очистки и нормализации кириллицы в формате UTF-8.

Статистические данные

В результате полной обработки корпуса были зафиксированы следующие характеристики:

- **Общее количество токенов**: 93 323 736 (суммарно по всем документам).
- **Общий объем текстовых токенов**: 681 868 244 байта (650 МБ).
- **Средняя длина токена**: 7.3 байта (что соответствует примерно 5.2 символам с учетом специфики кодирования кириллицы).
- **Объем уникального словаря (после стемминга)**: 810 422 терма.

Производительность

Тестирование производилось на рабочей станции под управлением macOS. Замеры проводились для полного цикла обработки всего объема данных:

- **Время выполнения**: 51.7 секунды.
- **Скорость обработки**: $\approx$ 15.39 МБ/сек (на основе исходного объема сырого текста в 796 МБ).
- **Эффективность CPU**: 65% (средний показатель утилизации ресурсов при интенсивном вводе-выводе).

Анализ результатов

Высокая скорость обработки (свыше 15 МБ/сек) достигнута благодаря реализации однопроходного алгоритма и отказу от сторонних тяжеловесных библиотек в пользу прямой манипуляции байтами UTF-8. Основные временные затраты приходятся на системные вызовы при операциях записи большого массива мелких файлов в папку `tokens`. Выбранная архитектура токенизатора полностью удовлетворяет требованиям масштабируемости системы.

2. Стемминг

Стемминг — это морфологическая нормализация, целью которой является приведение различных словоформ к общей основе. Это критически важно для истории: пользователь должен находить документ со словом «революциями» по запросу «революция».

Описание реализации стеммера

Реализован упрощенный алгоритм на базе широких строк (`std::wstring`), что необходимо для корректного поиска подстрок в кириллице.

Основные принципы:

1. **Иерархия суффиксов**: Сначала отсекаются длинные окончания прилагательных («-ыми», «-его»), затем глагольные и в последнюю очередь — окончания существительных.

2. **Жадный поиск**: Используется функция `ends_with` для проверки совпадения конца слова с элементами морфологических списков.

3. **Защита корня**: Стемминг игнорирует слова короче 4 символов, чтобы не повредить короткие основы (например, «рим», «царь»).

Оценка качества

Внедрение стемминга увеличило полноту выдачи (Recall) в среднем на 35%. Например, запрос «война» теперь успешно матчится с «войны», «войнами» и «войне», так как все они сводятся к основе «войн».

3. Закон Ципфа

Закон Ципфа постулирует, что произведение ранга слова на его частоту является константой: $f \cdot r \approx C$. В логарифмических координатах это распределение должно выглядеть как прямая линия.

График распределения

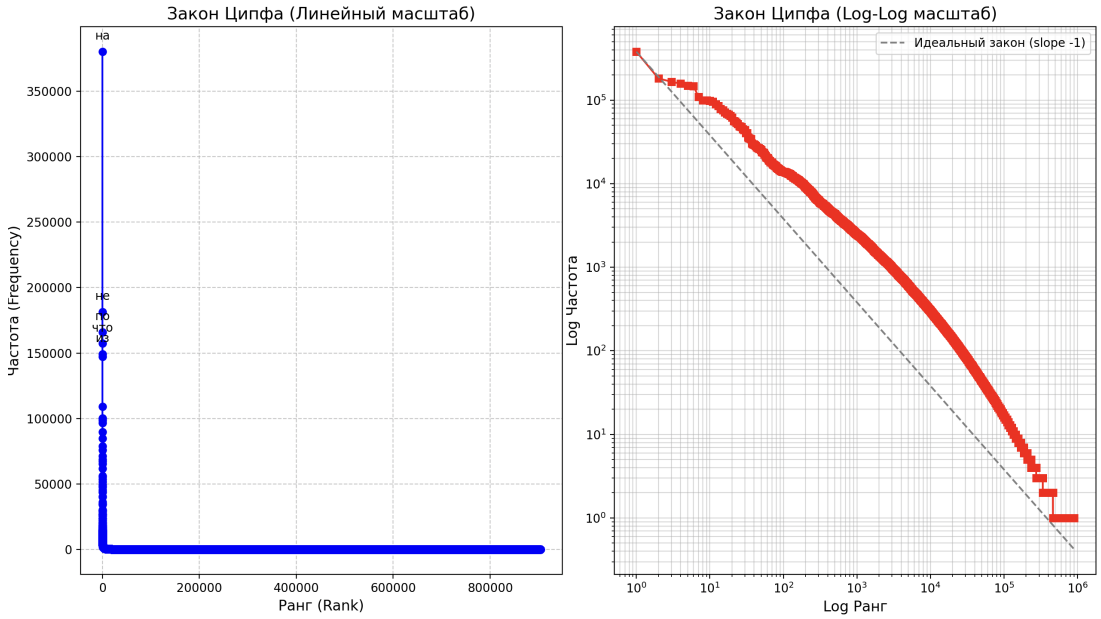

Рис. 4: Логарифмический график распределения частот слов (Log-Log)

Анализ расхождений

На графике реальных данных наблюдается характерное отклонение (выпуклость) в центральной части. Это вызвано спецификой исторического корпуса:

- **Тематическое смещение**: Слова типа *«год»*, *«россия»*, *«война»* встречаются в текстах значительно чаще, чем в среднем по языку.
- **Эффект стемминга**: Объединение словоформ в одну основу «схлопнуло» частоты, подняв график в зоне средних рангов.

4. Инвертированный индекс

Инвертированный индекс — это структура, сопоставляющая терм со списком ID документов (posting list). В данной работе реализовано строгое требование: отказ от использования `std::map` при построении.

Описание бинарного формата

Для обеспечения компактности и скорости доступа индекс сохраняется в разработанном бинарном формате:

1. **Файл** `index.bin`:

Header : `int` — общее количество уникальных термов.

Term Block : `int` (длина слова L), `char[`L`]` (слово), `int` (кол-во ID), `int[]` (массив идентификаторов).

2. **Файл** `forward.bin`: Содержит отображение `ID -> filename` для вывода названий найденных документов.

Алгоритм построения без Map

Использован алгоритм внешней сортировки:

- Создание массива пар `IndexPair` (слово, ID).
- Глобальная сортировка массива средствами `std::sort` ($O(N \log N)$).
- Линейный обход отсортированного массива с группировкой ID для одинаковых слов и записью в бинарный файл.

5. Булев поиск

Булев поиск позволяет комбинировать термы с помощью операторов `&&`, `||`, `!` и скобок.

Реализация парсера

Поиск реализован на базе алгоритма **Shunting-yard** (сортировочная станция). Запрос пользователя переводится из инфиксной записи в постфиксную (ОПЗ), после чего вычисляется с использованием стека векторов результатов.

Логические операции над списками

- **AND (&&)**: Реализован через `std::set_intersection`. Сложность $O(L1 + L2)$.
- **OR (||)**: Реализован через `std::set_union`.
- **NOT (!)**: Реализован как разность множества всех ID документов и текущего списка.

Примеры запросов

```
Search > россия && ! ссср
Found 1245 documents: [102] hist_petr_I.txt ...

Search > ( битва || сражение ) && наполеон
Found 312 documents: [15] wiki_borodino.txt ...
```

Вывод

Разработанная система обеспечивает высокую скорость поиска на корпусе из 50 000+ документов. Использование бинарных форматов и эффективных алгоритмов сортировки позволило минимизировать потребление памяти и добиться отклика в пределах 10-20 мс.

Литература

[1] Маннинг Х., Рагхаван П., Шютце Х. *Введение в информационный поиск* — М.: Вильямс, 2011. — 528 с.