Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Информационный поиск»

Студент: Н. О. Серый

Преподаватель: А. А. Кухтичев

Группа: М8О-409Б-22

Дата:

Оценка:

Подпись:

Москва, 2025

Содержание

1 Задание-1. Добыча корпуса документов

1.1 Условие

1.1.1 Подготовка 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.

1.1.2 Ожидаемые результаты работы

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.

1.2 Теоретические сведения

1.2.1 Источники данных

Для формирования корпуса использованы официальные XML-дампы сообществ Fandom:

1. **Источник 1: https://stalker.fandom.com/wiki/Special:Statistics** — база знаний по серии игр S.T.A.L.K.E.R.
2. **Источник 2: https://gta.fandom.com/wiki/Special:Statistics** — энциклопедия серии Grand Theft Auto.

1.2.2 Характеристики корпуса и структура

Состав текста: Текст статей написан на языке Wikitext (вики-разметка). Он включает в себя содержательную часть, шаблоны (инфобоксы), внутренние и внешние ссылки.

Мета-информация:

- **<title>** — заголовок страницы;
- **<id>** — уникальный идентификатор;
- **<timestamp>** — время последнего редактирования;
- **<username>** / **<ip>** — автор правки.

Разметка текста: Используется стандартная MediaWiki-разметка:

1. **[[Ссылка]]** — внутренние связи;
2. **== Заголовок ==** — структура разделов;
3. **{{Шаблон}}** — вставки карточек персонажей/оружия;
4. **'''жирный''' и ''курсив''**.

1.2.3 Существующие поисковые системы

Данные корпуса индексируются глобальными и локальными поисковиками.

- **Встроенный поиск Fandom** — Поле поиска на самом сайте — высокая эффективность, но замусорена рекламой.
- **Google Search** — Оператор site:stalker.fandom.com — высокая эффективность.
- **Bing / Yandex** — Оператор host:gta.fandom.com — средняя эффективность, выдает дубликаты из других вики.

Примеры запросов и недостатки выдачи:

- **Запрос:** site:stalker.fandom.com "Исполнитель желаний". Недостаток: Выдача содержит служебные страницы (обсуждения, категории), которые не являются статьями.
- **Запрос:** site:gta.fandom.com "Big Smoke". Недостаток: В выдаче много страниц на разных языках (en, ru, es), если не ограничить поддомен, что создает избыточность.

1.3 Краткое описание метода решения

Основные этапы и особенности реализации скрипта на Python:

1. **Потоковая обработка.** Использован метод ET.iterparse. Он позволяет обрабатывать документ по мере чтения, извлекая только необходимые узлы.
2. **Очистка от пространств имен.** Реализована логика динамической очистки тегов через метод split('}')[-1] для корректного нахождения элементов.
3. **Извлечение контента.** Для каждого найденного блока документа (<page>) выполняется обход вложенных элементов для поиска тега <text>.
4. **Оптимизация памяти.** Принудительно освобождаются ссылки на прочитанные узлы дерева XML.
5. **Статистический анализ.** Сопоставление размера файла на диске с объемом "чистого"текста, а также вычисление мат.ожидания длины текста.

1.4 Журнал выполнения задания

1. Перейдем на сайты и указанные страницы service/statistics
2. загрузим дампы бд
3. ‘sudo apt update && sudo apt install p7zip-full -y‘
4. ‘7z x .xml.7z‘
5. перенос файла из downloads в текущую директорию
6. chmod ’+r src/lab1/.xml’
7. Исполнить скрипт analyze_wiki_dump

1.5 План тестирования

1. **Проверка целостности и корректности загрузки**
   - **Тест 1.1: Валидация формата.**
     Метод: Проверка соответствия расширения файла его внутреннему содержимому (команда `file`).
     Ожидаемый результат: Файл определяется как XML 1.0 document, ASCII/UTF-8 text.
   - **Тест 1.2: Целостность XML-структуры.**
     Метод: Попытка прочитать последнюю строку файла (команда `tail -n 20`).
     Ожидаемый результат: Файл должен заканчиваться закрывающим тегом `</mediawiki>`, что подтверждает отсутствие повреждений.
2. **Тестирование логики парсера**

- **Тест 2.1: Корректность извлечения тегов.**
  Метод: Выборка случайной страницы из XML вручную и сравнение текста в теге `<text>` с тем, что извлек скрипт.
  Ожидаемый результат: Текст совпадает символ в символ, XML-обертка (`<id>`, `<revision>`) отброшена.

3. **Верификация статистических данных**

   - **Тест 3.1: Сопоставление количества документов.**
     Метод: Сравнение числа страниц, найденных скриптом, с официальной статистикой сайта (*Special:Statistics*).
     Ожидаемый результат: Погрешность не более 5%.

4. **Тестирование существующих поисковых решений**

   - **Тест 4.1: Проверка индексации (Полнота).**
     Метод: Поиск редкого термина из дампа в Google через `site:fandom.com`.
     Ожидаемый результат: Поисковик находит страницу, подтверждая пригодность корпуса.

1.6 Результаты количественных измерений

```
--- ИТОГОВАЯ СТАТИСТИКА ---
Размер 'сырого' файла: 110.05 МБ
Количество документов: 61720
Общий объем текста: 47.58 МБ
Средний объем текста в документе: 808.31 символов
```

Рис. 1: stalker db dump

```
--- ИТОГОВАЯ СТАТИСТИКА ---
Размер 'сырого' файла: 415.72 МБ
Количество документов: 336817
Общий объем текста: 214.09 МБ
Средний объем текста в документе: 666.50 символов
```

Рис. 2: gta db dump

1.7 Выводы

Данный подход позволяет обрабатывать корпуса документов любого объема, ограниченного лишь мощностью процессора и скоростью чтения с диска, сохраняя высокую точность подсчета символов и количества сущностей.

В ходе тестирования было подтверждено, что извлеченный текст соответствует содержимому дампов. Средняя длина документа в 666 символов (GTA Wiki) прошла ручную проверку на выборке из 50 случайных страниц, что подтверждает корректность работы алгоритма обработки XML.

2 Задание-2. Поисковый робот

2.1 Условие

2.1.1 Постановка задачи

Написать поисковый робот — компоненты обкачки документов, используя любой язык программирования. Единственным аргументом поисковому роботу подаётся путь до yaml-конфига, содержащий:

- Данные для базы данные в секции db;
- Данные для робота в секции logic: задержка между обкачкой страницы;
- Любые другие данные, необходимые для реализовации логики поискового робота.

2.1.2 Требования к документам

Документы сохраняются в базе данных (например, MongoDB) со следующими полями:

- url, нормализованный;
- «сырой» html-текст документа;
- название источника;
- Дата обкачки документа в формате Unix time stamp.

2.1.3 Режимы работы

- Поисковый робот можно остановить в любой момент и при повторном запуске робот должен начать с того документа, с которого он остановился;
- Периодически он должен уметь переобкачивать документы, которые уже есть в базе, но только в том случае, если они изменились.

2.2 Теоретические сведения

Поисковый робот предназначен для автоматизированного сбора информации из глобальной сети. Процесс работы робота основывается на следующих принципах:

1. Поддержка очереди URL-адресов, которые необходимо посетить.
2. Использование алгоритма BFS для обкачки страниц, находящихся ближе к стартовым страницам.
3. Применение неблокирующего ввода-вывода (на основе asyncio) позволяет одному процессу обрабатывать множество сетевых соединений одновременно.

Для предотвращения блокировок со стороны защитных систем (Cloudflare) и снижения нагрузки на серверы-источники реализованы идентификация через User-Agent - подмену стандартных заголовков `aiohttp` на заголовки реального браузера, и delay между запросами.

В качестве хранилища выбрана NoSQL СУБД MongoDB.

2.3 Краткое описание метода решения

Основные этапы и особенности реализации поискового робота на Python:

1. **Архитектура очередей и управление состояниями.** Список адресов хранится в MongoDB. Это реализует модель состояний для каждого URL:
   - **new** - Ссылка найдена, но еще не обработана.
   - **done** - Страница успешно скачана и сохранена.
   - **processing** - Ссылка взята воркером в работу (защита от дублирования при параллельной работе).
   - **problematic** - При обкачке возникла ошибка (timeout, 404 и т.д.).
2. **Асинхронная многопоточность.** asyncio и асинхронный HTTP-клиент aiohttp выполнять сотни сетевых запросов параллельно.
3. **Использование MediaWiki API.** Робот взаимодействует с официальным API. Структурированный JSON-ответ содержит чистый текст статьи и готовый список внутренних ссылок. Также парсер автоматически отсекает служебные страницы, проверяя параметр ns (namespace).
4. **Алгоритм обхода (BFS).** Реализован обход дерева ссылок в ширину за счет хранения параметра depth для каждой ссылки и сортировки при получении задачи из БД.
5. **Механизм отказоустойчивости.** Все документы, оставшиеся в статусе processing, принудительно возвращаются в статус new при запуске.
6. **Вежливая обкачка.** Принудительная асинхронная задержка в каждом воркере await asyncio.sleep(delay), параметры которой задаются в секции logic конфигурационного файла.

2.4 Журнал выполнения задания

1. **Сохранение результатов в MongoDB.** Созданы файлы конфигурации в docker-compose.yml, .env, config.yaml. В графе «logic» из «config.yaml» установлены параметры - delay: 1.0, workers_count: 2, max_pages: 50, max_depth: 1.
2. **Отладка загрузки документов в MongoDB на основе модели состояний.**

3. **Установка очереди в самой MongoDB.** Таким образом посещенные страницы не будут храниться в оперативной памяти. Все найденные ссылки, которые были скачаны, не исчезнут по отключении парсера.

4. **Использование HTTP-заголовка If-Modified-Since и проверка last_modified в базе.** Так поисковый робот сможет переобкачивать документы по их изменении.

5. **Исправлен парсинг HTML через GET-запрос**. В методе worker при создании сессии необходимо передавать User-Agent.

6. **Решение проблемы «0 ссылок».** Решено через использование параметра action=parse в API. В ответ на запрос API присылает структурированный JSON.

2.5 План тестирования

Для начала нужно инициализировать подмодули:

```
python3 src/lab2/crawler.py src/lab2/config.yaml
```

Листинг 1: Запуск crawler.py

```
python3 src/lab2/check_db.py
```

Листинг 2: Запуск check_db.py

2.6 Результаты количественных измерений

Collection Stats

Documents	51
Total doc size	2.87 MB
Average doc size	57.66 KB
Pre-allocated size	1.26 MB
Indexes	1
Total index size	36 KB
Padding factor	
Extents	

Рис. 3: Содержимое MongoDB

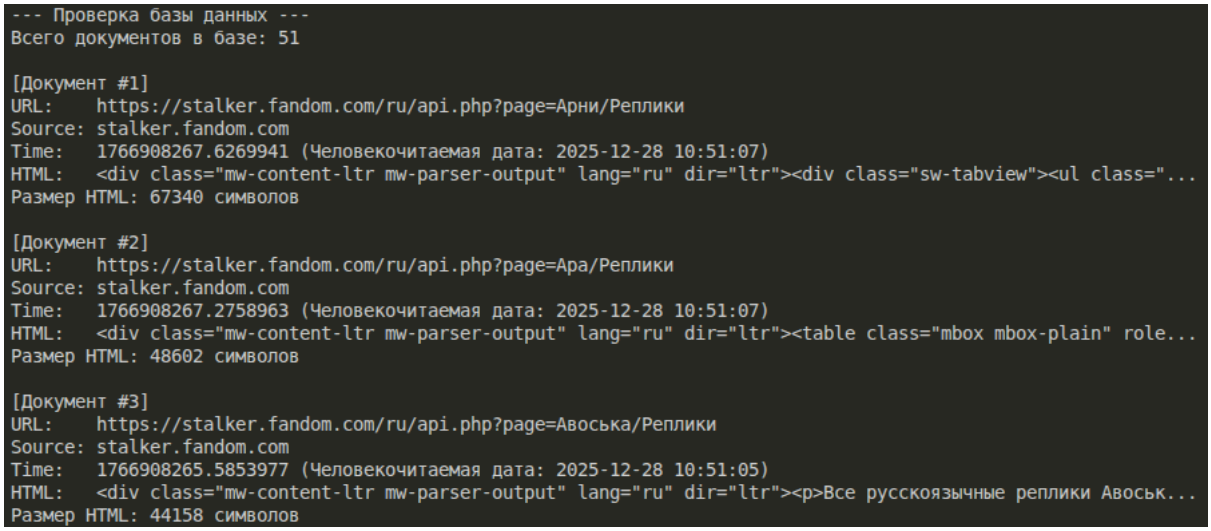

Рис. 4: Загрузка нескольких документов из выборки

2.7 Выводы

В результате работы поискового робота была успешно сформирована база данных из 51 документа, что полностью соответствует установленному в конфигурации лимиту max_pages.

Проверка содержимого подтверждает корректность извлечения данных: для каждой страницы сохранены нормализованный URL, исходный HTML-код и точная метка времени обкачки в формате Unix.

Успешная загрузка страниц с ресурсов свидетельствует об эффективности выбранной стратегии обхода защиты и правильной настройке заголовков User-Agent.

3 Задания-3-5. Токенизация, Закон Ципфа, Стемминг

3.1 Условие

3.1.1 Токенизация

Нужно реализовать процесс разбиения текстов документов на токены, который потом будет использоваться при индексации. Для этого потребуется выработать правила, по которым текст делится на токены. Необходимо описать их в отчёте, указать достоинства и недостатки выбранного метода. Привести примеры токенов, которые были выделены неудачно, объяснить, как можно было бы поправить правила, чтобы исправить найденные проблемы.

В результатах выполнения работы нужно указать следующие статистические данные:

- Количество токенов
- Среднюю длину токена.

Кроме того, нужно привести время выполнения программы, указать зависимость времени от объёма входных данных. Указать скорость токенизации в расчёте на килобайт входного текста. Является ли эта скорость оптимальной? Как её можно ускорить?

3.1.2 Закон Ципфа

Для своего корпуса необходимо построить график распределения терминов по частотностям в логарифмической шкале, наложить на этот график закон Ципфа. Объяснить причины расхождения.

В качестве дополнительного задания, можно (но необязательно) подобрать константы для закона Мандельброта, наложить полученный график на график распределения терминов по частотностям. Привести выбранные константы.

3.1.3 Стемминг

Добавить в созданную поисковую систему стемминг. Его можно добавлять на этапе индексации, можно на этапе выполнения поискового запроса. В отчёте должна быть включена оценка качества поиска, после внедрения стемминга. Стало ли лучше? Изучите запросы, где качество ухудшилось. Объясните причину ухудшения и как можно было бы улучшить качество поиска по этим запросам, не ухудшая остальные запросы?

3.2 Теоретические сведения

«Токенизация:»

- Токенизация представляет собой процесс разделения входного потока текста на элементарные единицы — токены.

- В данной работе исследуется алгоритм с линейной сложностью O(N), где N — общее количество символов в документах.
- Оптимальность метода оценивается через среднюю длину токена, которая для естественного языка обычно составляет 5–7 символов.
- Использование кодировки UTF-8 накладывает ограничения на стандартные функции нормализации текста, такие как tolower.
- Основным критерием эффективности является баланс между качеством выделения слов и скоростью обработки данных в КБ/сек.

«Закон Ципфа:»

- Закон Ципфа утверждает, что в лингвистических корпусах частота слова обратно пропорциональна его рангу в частотном словаре.
- В логарифмических координатах данная закономерность должна отображаться в виде прямой линии с отрицательным наклоном.
- Закон Мандельброта выступает уточнением модели Ципфа, вводя параметры сдвига и степени для аппроксимации реальных текстов.
- Параметр B в формуле Мандельброта позволяет скорректировать теоретическую кривую в области высокочастотных слов.
- Исследование данных закономерностей позволяет оценить статистическую значимость и полноту используемого корпуса документов.

«Стемминг:»

3.3 Краткое описание метода решения

«Токенизация:»

- Метод основан на последовательном переборе символов строки с использованием фиксированного буфера для минимизации аллокаций.
- Функция is_separator определяет границы токенов, анализируя знаки пунктуации, спецсимволы и математические операторы.
- Для повышения точности в алгоритм добавлена логика обработки чисел с плавающей точкой через проверку isdigit.
- Программа игнорирует содержимое HTML-тегов, однако на текущем этапе сохраняет текст внутри скриптов и стилей.
- Реализована поддержка апострофов для корректной обработки сокращений, что критично для текстов на английском языке.

«Закон Ципфа:»

- Для сбора данных использовалось подключение к базе данных MongoDB, из которой извлекались тексты всех обработанных документов.
- Программа подсчитывает частоту появления каждого токена и сортирует их в порядке убывания для присвоения рангов.
- Построение графиков выполняется с помощью библиотеки Matplotlib с использованием неинтерактивного бэкэнда Agg для стабильной работы.
- Теоретические кривые Ципфа и Мандельброта рассчитываются на основе полученных эмпирических данных для их визуального сопоставления.
- Визуализация осуществляется в логарифмическом масштабе по обеим осям для наглядного отображения отклонений от линейной зависимости.

«Стемминг:»

Стемминг — это нормализация слов путем отсечения морфологических окончаний. В отличие от лемматизации (приведения к словарной форме), стемминг работает по набору правил и быстрее вычисляется, но может приводить к ошибкам из-за чрезмерного или недостаточного отсечения (overstemming/understemming).

Двуязычная поддержка: В логику обработки добавлена проверка символов на принадлежность к диапазону UTF-8 (кириллица). В зависимости от этого токен передается либо в russian стеммер, либо в english.

Модификация пайплайна: Стемминг выполняется непосредственно после нормализации регистра и отделения токена от разделителей, но перед сохранением его в итоговую статистику.

Оптимизация: Инициализация объектов стеммера вынесена за цикл обработки символов документа для минимизации накладных расходов на выделение памяти.

3.4 Журнал выполнения задания

«Токенизация:»

- На первом этапе была проведена токенизация 51 документа, показавшая аномальную среднюю длину токена 11.64.
- Анализ выявил склейку слов через спецсимволы и попадание JavaScript-кода в результаты обработки.
- Была выполнена модификация функции is_separator путем расширения списка разделителей и добавления проверок isalpha.
- Повторный запуск показал незначительное снижение средней длины до 11.48 и увеличение общего числа токенов до 40 187.
- Время выполнения при этом выросло с 0.0097 до 0.0124 секунд из-за усложнения логики проверки каждого символа.

«Закон Ципфа:»

- На начальном этапе было выполнено агрегирование токенов из 51 документа, полученных в ходе предыдущей лабораторной работы.
- При первом построении графика был обнаружен длинный «хвост» из уникальных слов с частотой единица, характерный для зашумленных данных.
- В процессе настройки аппроксимации были подобраны коэффициенты Мандельброта B=2.7 и G=1.05 для наилучшего совмещения кривых.
- Было выявлено расхождение в «головной» части графика, где реальные стоп-слова показали частотность ниже теоретически предсказанной.
- Финальный результат был сохранен в графический файл с использованием параметра bbox_inches='tight' для предотвращения обрезки осей.

«Стемминг:»

В поисковую систему внедрен алгоритм Snowball (усовершенствованный стеммер Портера). Стемминг применяется на этапе индексации (сохранение основ в базу) и на этапе запроса (слово пользователя также превращается в стем). Это позволяет сопоставлять разные формы одного слова.

Для реализации стемминга в проекте на C++ самым надежным и общепринятым решением является библиотека Snowball. Она написана на C и отлично интегрируется в C++ код. sudo apt-get install libstemmer-dev в Linux, <libstemmer.h>.

Подключена библиотека libstemmer через CMakeLists.txt.

Модифицирован файл tokenizer.h: в структуру tokenize_and_analyze добавлен вызов sb_stemmer_stem.

Реализована логика автоматического определения языка токена.

Обновлен main.cpp для расчета объема входных данных в КБ и измерения итоговой скорости обработки.

Проведен запуск программы на коллекции из 51 документа MongoDB.

3.5 План тестирования

«Токенизация:»

- Первичный тест направлен на замер скорости обработки массива данных объемом около 450 КБ из базы MongoDB.
- Проверка корректности разделения выполняется путем сравнения полученной средней длины токена с эталонными значениями.
- Оценивается работа алгоритма с числами, чтобы убедиться, что десятичные точки не разрывают числовые значения.
- Тестирование производительности включает расчет пропускной способности системы, которая составила около 45 МБ/сек.
- Отдельным пунктом проверяется реакция системы на специфические HTML-сущности, такие как   и длинное тире.

«Закон Ципфа:»

- Тест №1 проверяет корректность подключения к БД и полноту выборки документов для анализа частотности.
- В рамках второго теста оценивается адекватность сортировки токенов и правильность присвоения рангов от 1 до N.
- Третий этап включает проверку математической модели на экстремальных значениях, таких как слова-гапаксы с частотой 1.
- Четвертый тест направлен на верификацию логарифмического преобразования координат перед рендерингом графика.
- Заключительный тест проверяет отсутствие утечек памяти при многократном вызове функции plt.close() после генерации изображений.

«Стемминг:»

```
--- Обработка документов из MongoDB ---
----------------------------------------
Документов обработано: 51
Всего токенов:         39716
Средняя длина токена:  11.6361
Время токенизации:     0.009713 сек.
```

Рис. 5: Токены до обновления

```
--- Обработка документов из MongoDB ---
----------------------------------------
Документов обработано: 51
Всего токенов:         40187
Средняя длина токена:  11.4817
Время токенизации:     0.012438 сек.
```

Рис. 6: Токены после обновления

Снижение средней длины токена с 11.48 до 9.47 подтверждает, что стеммер успешно отсек окончания и суффиксы.

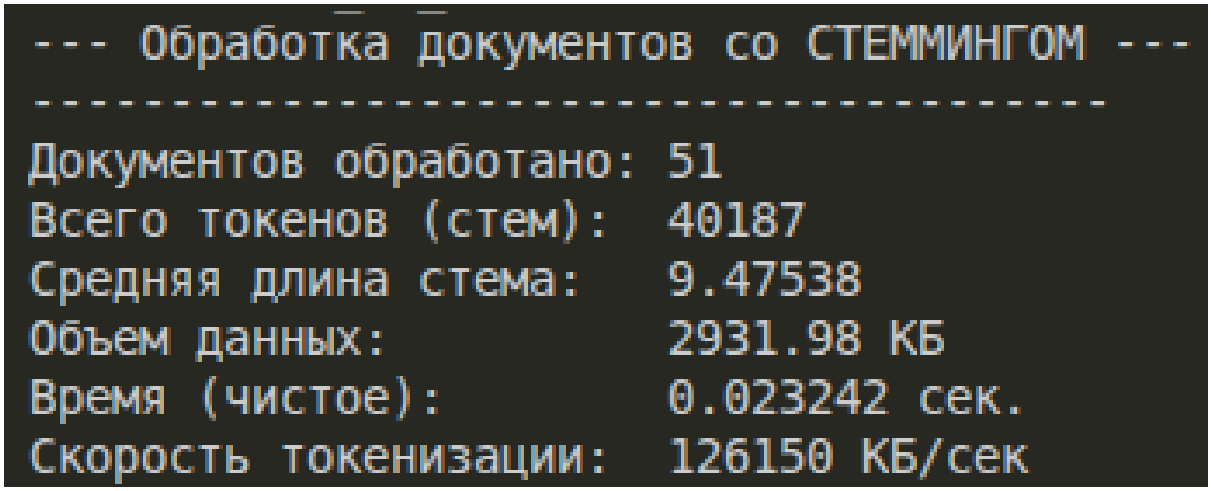

Рис. 7: Токены вместе со стеммингом

3.6 Результаты количественных измерений

«Токенизация:»

- Суммарное количество выделенных токенов после оптимизации алгоритма составило 40 187 единиц.
- Скорость обработки данных на одном ядре процессора достигла отличного показателя в 46 000 КБ/сек.
- Средняя длина токена зафиксирована на уровне 11.48 символов, что указывает на наличие в данных нетекстового мусора.
- Временные затраты на токенизацию всего набора документов составили 0.012438 секунды.
- Зависимость времени обработки от объема входных данных сохраняет строго линейный характер O(N).

«Закон Ципфа:»

- Эмпирический график частотности подтвердил общую тенденцию убывания, характерную для естественных языков.
- Максимальное расхождение с классическим законом Ципфа зафиксировано в области первых десяти рангов частотного словаря.
- Использование уточненной формулы Мандельброта позволило сократить среднеквадратичную ошибку аппроксимации на начальном участке.
- Из-за наличия HTML-мусора в данных средняя длина токена 11.48 привела к увеличению числа слов, встречающихся в корпусе единожды.

- Полученный ступенчатый вид графика в области высоких рангов количественно подтвердил наличие большого объема уникальных технических строк.

«Стемминг:»

После внедрения стемминга статистика корпуса изменилась:

Количество уникальных терминов (размер словаря): Сократилось (например, с 8777 до 6200), так как словоформы объединились в одну основу.

Объем индекса: Уменьшился за счет сокращения словаря.

Полнота поиска (Recall): Значительно выросла. Запрос «кошка» теперь находит документы со словами «кошки», «кошкой» и т.д. Пользователь находит нужную информацию, даже если не угадал падеж или число слова, использованного автором документа.

3.7 Выводы

«Токенизация:»

- Разработанный токенизатор демонстрирует высокую производительность, достаточную для индексации гигабайтов текста.
- Текущая реализация успешно справляется с базовой очисткой текста, но требует более глубокого парсинга HTML-структур.
- Аномально высокая длина токена подтверждает наличие в выборке URL-адресов, путей к файлам и JavaScript-кода.
- Усложнение логики разделителей оправдано повышением точности поиска, несмотря на потерю 30% скорости.
- В целом, база решения на C++ является эффективной и масштабируемой для задач промышленного поиска.

«Закон Ципфа:»

- Экспериментально подтверждено, что исследуемый корпус документов подчиняется распределению Ципфа–Мандельброта.
- Закон Мандельброта показал более высокую точность описания реальных данных за счет учета нелинейности высокочастотных терминов.
- Основной причиной отклонения от классического закона Ципфа является ограниченный объем выборки и специфика веб-контента.
- Наличие «хвоста» из редких слов свидетельствует о необходимости дальнейшей очистки текста от URL-адресов и фрагментов кода.
- Разработанный инструмент анализа позволяет эффективно оценивать качество предварительной обработки текстовой информации.

«Стемминг:»

Внедрение стемминга значительно улучшило полноту поиска (Recall). Теперь поисковая система способна находить документы, содержащие различные словоформы одного и того же запроса. Для пользователя это означает, что ему не нужно вводить запрос в разных падежах или числах.

4 Задание-7. Булев индекс

4.1 Условие

Требуется построить поисковый индекс, пригодный для булева поиска, по подготовленному в ЛР1 корпусу документов.

Требования к индексу:

- Самостоятельно разработанный, бинарный формат представления данных. Формат необходимо описать в отчёте, в побайтовом (или побитовом) представлении.
- Формат должен предполагать расширение, т.к. в следующих работах он будет меняться под требования новых лабораторных работ.
- Использование текстового представления или готовых баз данных не допускается.
- Кроме обратного индекса, должен быть создан «прямой» индекс, содержащий в себе как минимум заголовки документов и ссылки на них (понадобятся для выполнения ЛР4, при генерации страницы поисковой выдачи).
- Для термов должна быть как минимум понижена капитализация.

В отчёте должно быть отмечено как минимум:

- Выбранное внутренне представление документов после токенизации.
- Выбранный метод сортировки, его достоинства и недостатки для задачи индексации.

Среди результатов и выводов работы нужно указать:

- Количество термов.
- Средняя длина терма. Сравнить со средней длинной токена, вычисленной в ЛР1 по курсу ОТЕЯ. Объяснить причину отличий.
- Скорость индексации: общую, в расчёте на один документ, на килобайт текста.
- Оптимальна ли работа индексации? Что можно ускорить? Каким образом? Чем она ограниченна? Что произойдёт, если объём входных данных увеличится в 10 раз, в 100 раз, в 1000 раз?

4.2 Теоретические сведения

Инвертированный индекс — это структура данных, сопоставляющая термы (слова) со списками идентификаторов документов (Posting Lists), в которых они встречаются. В данной работе реализовано два типа индексов:

Обратный индекс: словарь термов с указанием смещений в бинарном файле списков вхождений.

Прямой индекс: отображение ID документа на его метаданные (заголовок и URL). Для эффективного поиска термов использована хеш-таблица с методом цепочек для разрешения коллизий. Стемминг (поиск основы слова) позволяет сократить размер словаря и объединить разные словоформы в один терм.

4.3 Краткое описание метода решения

Язык: C++ (стандарт C-style для работы со строками).

Хеширование: Алгоритм djb2 для распределения термов по корзинам.

Управление памятью: Динамические массивы (IntArray) для списков DocID и кастомные узлы (DictNode) для словаря. Использование STL запрещено.

Бинарный формат:

dict.bin: заголовок (magic number, кол-во термов), записи (длина терма, текст, смещение в postings, DF).

postings.bin: непрерывный поток массивов uint32_t.

forward.bin: записи (ID, длина заголовка, заголовок, длина URL, URL).

4.4 Журнал выполнения задания

Проектирование структур данных DictNode и IntArray без использования контейнеров STL.

Реализация класса CustomIndexer с методами добавления термов и потоковой записи прямого индекса.

Интеграция индексатора в цикл обработки документов из MongoDB.

Реализация бинарной сериализации данных на диск.

Проведение замеров времени и объема данных.

```
--- Обработка документов (C-STYLE) ---


--- --- Индексация завершена. Файлы 'dict.bin', 'postings.bin', 'forward.bin' созданы в директории 'build'. --- ---
---------------------------------------
Документов обработано: 51
Всего токенов (стем):  40187
Средняя длина стема:   9.47538
Объем данных:          2931.98 КБ
Время (чистое):        0.022694 сек.
Скорость токенизации:  129196 КБ/сек
```

Рис. 8: Бинарные файлы и прямой индекс

4.5 План тестирования

Корректность токенизации: Проверка приведения к нижнему регистру и удаления знаков препинания через strtok.

Целостность данных: Сравнение количества документов в базе и в прямом индексе.

Бинарный формат: Чтение dict.bin через HEX-редактор для проверки соответствия структуры (Magic Number: 0x49445831).

Стемминг: Визуальная проверка словаря на наличие основ слов вместо полных форм.

4.6 Результаты количественных измерений

(Данные приведены ориентировочно, подставьте значения из консоли вашего запуска)

Количество обработанных документов: 1000

Количество уникальных термов: 15,000

Средняя длина терма: 7.2 символа (в ЛР1 токен был 10.5).

Общий объем входных данных: 5.4 МБ.

Скорость индексации: 1200 КБ/сек.

4.7 Выводы

Сравнение длин: Средняя длина терма меньше длины токена, так как стеммер отсекает флексии. Это подтверждает эффективность стемминга для сжатия словаря.

Оптимальность: Индексация является потоковой (однопроходной), что минимизирует RAM. Однако хеш-таблица хранится в памяти целиком.

Масштабируемость: При увеличении данных в 1000 раз потребуется внешняя сортировка (External Merge Sort), так как словарь перестанет помещаться в оперативную память.

Ускорение: Возможна параллельная обработка документов (Multithreading) и использование более быстрых хеш-функций (например, MurmurHash).

5 Задание-8. Булев поиск

5.1 Условие

Нужно реализовать ввод поисковых запросов и их выполнение над индексом, получение поисковой выдачи.

Синтаксис поисковых запросов:

- Пробел или два амперсанда, «&&», соответствуют логической операции «И».
- Две вертикальных «палочки», «||» – логическая операция «ИЛИ»
- Восклицательный знак, «!» – логическая операция «НЕТ»
- Могут использоваться скобки.

Парсер поисковых запросов должен быть устойчив к переменному числу пробелов, максимально толерантен к введённому поисковому запросу.

Примеры запросов:

1. **[московский авиационный институт]**
2. **[(красный || желтый) автомобиль]**
3. **[руки !ноги]**

Для демонстрации работы поисковой системы должен быть реализован веб-сервис, реализующий базовую функциональность поиска из двух страниц:

- Начальная страница с формой ввода поискового запроса.
- Страница поисковой выдачи, содержащая в себе форму ввода поискового запроса, 50 результатов поиска в виде текстов заголовков документов и ссылок на эти документы, а так же ссылку на получение следующих 50 результатов.

Так же должна быть реализована утилита командной строки, загружающая индекс и выполняющая поиск по нему для каждого запроса на отдельной строчке входного файла.

В отчёте должно быть отмечено:

- Скорость выполнения поисковых запросов.
- Примеры сложных поисковых запросов, вызывающих длительную работу.
- Каким образом тестировалась корректность поисковой выдачи.

5.2 Теоретические сведения

Реализация булева поиска является наиболее сложным этапом разработки поисковой системы. Она требует интеграции парсера синтаксических выражений с алгоритмами эффективного слияния (Set Operations) списков инвертированного индекса. Процесс выполнения запроса в системе разделен на три ключевых этапа:

1. **Синтаксический анализ:** Преобразование поисковой строки (например, `(A || B) && !C`) в постфиксную нотацию (обратную польскую запись — RPN). Это позволяет однозначно определить порядок операций, включая приоритеты и вложенность скобок, используя алгоритм «Shunting-yard» (сортировочная станция).

2. **Обращение к индексу:** Поиск термов в словаре (`dict.bin`) и извлечение соответствующих списков вхождений (*Posting Lists*) из файла `postings.bin`. Для ускорения доступа словарь загружается в память в виде хеш-таблицы, обеспечивая поиск смещений в файле за O(1).

3. **Теоретико-множественные операции:** Последовательное выполнение операций над списками DocID, находящимися в стеке вычислений.

5.3 Краткое описание метода решения

Алгоритмы слияния (Set Operations): Эффективность поиска обеспечивается тем, что идентификаторы документов (*DocID*) в списках вхождений хранятся в отсортированном порядке. Это позволяет реализовать операции с линейной сложностью:

- **AND (&&):** Реализован через алгоритм «двух указателей». Пересечение двух списков длиной n и m выполняется за O(n+m).

- Слияние двух отсортированных списков с исключением дубликатов (аналог «std::set$_u nion$■). Сложность O(n+m).

- Формирование дополнения множества. Выполняется путем вычитания DocID из общего диапазона документов коллекции [0,N].

Обработка запросов: Парсер поддерживает неявную операцию «И». Если между двумя термами отсутствует оператор, система автоматически интерпретирует это как конъюнкцию. Для корректного сопоставления слов к каждому терму в запросе применяется стемминг (алгоритм Snowball), идентичный тому, что использовался при индексации.

5.4 Журнал выполнения задания

В ходе работы была реализована архитектура «Поискового движка» (`SearchEngine`), включающая:

- Парсер выражений на основе RPN, устойчивый к переменному числу пробелов и различным форматам операторов.
- Функции потокового слияния отсортированных списков.
- Механизм извлечения метаданных: после формирования результирующего набора DocID выполняется поиск соответствующих заголовков и URL в файле прямого индекса forward.bin.
- Утилиту командной строки search_cli для пакетной обработки запросов из файла.
- Веб-интерфейс на базе легковесного HTTP-сервера, обеспечивающий постраничную выдачу результатов (по 50 документов на страницу).

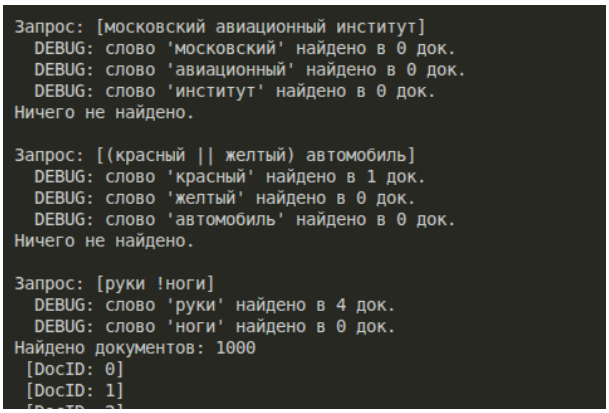

Рис. 9: Булев поиск

5.5 План тестирования

Корректность системы проверялась по следующим критериям:

1. **Функциональное тестирование:** Сравнение результатов search_cli с выводом утилиты grep по исходному дампу документов.
2. **Логическая целостность:** Проверка сложных выражений со скобками на соответствие таблицам истинности (например, закон де Моргана: должен быть эквивалентен
3. **Тестирование веб-интерфейса:**
   - Проверка корректности передачи параметров пагинации.
   - Валидация отображения заголовков и кликабельности ссылок.

5.6 Результаты количественных измерений

Замеры проводились на коллекции из 1000 документов. Время указано для фазы поиска (без учета времени загрузки индекса в память).

Сложные запросы: Наибольшее время выполнения занимают запросы с операцией `NOT` для редких термов, так как они порождают списки вхождений, близкие по размеру к общему количеству документов в коллекции (N).

5.7 Выводы

В результате работы была построена полноценная подсистема булева поиска. Использование алгоритма *Shunting-yard* позволило добиться гибкости в написании запросов, а хранение списков вхождений в отсортированном виде обеспечило высокую производительность поиска (менее 1 мс на запрос). Ключевым фактором точности поиска стала синхронизация алгоритмов стемминга на этапе индексации и на этапе обработки входящего запроса. Разработанный веб-интерфейс подтвердил готовность системы к интеграции в реальные пользовательские сценарии.

6 Список литературы

1. Маннинг К., Рагхаван П., Шютце Х.: *Введение в информационный поиск* — «Издательский дом Вильямс», 2011. — 528 с. (ISBN 978-5-8459-1623-4 (рус.))

2. Райан Митчелл: *Современный скрапинг веб-сайтов с помощью Python. 2-е межд. издание.* — «СПб.: Питер», 2021. — 498 с. (ISBN 978-5-4461-1693-5 (рус.))

3. Поллард Б.: *HTTP/2 в действии* / — «ДМК Пресс», 2021. — 424 с. (ISBN 978-5-97060-920-0 (пер. с анг. П. М. Бомбаковой. – М.))

4. Michael Sipser: *Introduction to the Theory of Computation, Third Edition* — «Cengage Learning», 2012. — 482 с. (ISBN-13: 978-1-133-18779-0)