Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики
и программирования

Лабораторная работа по курсу «Информационный поиск»

Студент: Д. У. Диёров
Преподаватель: А. А. Кухтичев
Группа: М8О-407Б
Дата: 29.12.2025
Оценка:
Подпись:

Москва, 2025

Цель работы

Найти и проанализировать корпус документов, который будет использован при выполнении последующих лабораторных работ; написать парсер на любом языке программирования; реализовать процесс разбиения текстов документов на токены, который затем будет использоваться при индексации; реализовать стемминг; реализовать булев индекс и булев поиск; написать юнит-тесты для проверки корректности работы.

Задание

Парсер

В качестве корпуса документов был выбран форум `https://anime-characters-fight.fandom.com`, содержащий около 1 200 000 страниц из 23 000 статей с описаниями персонажей и их способностей. Для скачивания и парсинга страниц был использован скрипт на языке программирования Go.

Скрипт получает начальный набор ссылок и рекурсивно обходит их, проверяя, что статьи принадлежат тому же домену, чтобы избежать парсинга сторонних ресурсов. Для скачивания страниц использовалась стандартная библиотека `http`, а для парсинга HTML — open-source библиотека `github.com/PuerkitoBio/goquery`.

Работа парсера проверялась с помощью юнит-тестов. При скачивании структура `Crawler` проверяет соблюдение правил `robots.txt`: если ссылка недоступна, она пропускается и не запрашивается повторно.

Метод `Crawl` скачивает страницу и передаёт её парсеру, который возвращает удобочитаемое JSON-представление. Далее `Crawler` сохраняет данные в базу данных PostgreSQL. Основная функция скрипта использует паттерн `WorkerPool` для вызова метода `Crawl`; между запросами горутины делают паузу в 100 мс, чтобы не создавать чрезмерную нагрузку на форум.

Производительность токенизации

Для оценки скорости токенизации был сгенерирован текст объёмом 1024 КБ.

Количество полученных токенов — 149 798. Средняя длина токена — 7,0 символов. Время токенизации — 0,04 секунды. Скорость обработки — примерно 25 МБ/с.

Для анализа данных в базе данных был реализован вспомогательный скрипт на Python.

Качество поиска

При использовании стемминга были получены следующие усреднённые показатели:

- Precision@5 — 0,32
- Recall@5 — 0,80
- F1@5 — 0,45

При отключении стемминга качество поиска улучшилось:

- Precision@5 — 0,60
- Recall@5 — 0,80
- F1@5 — 0,67

Полнота осталась на прежнем уровне, однако точность значительно возросла, что говорит о более корректной обработке терминологических и именных запросов без применения стемминга.

Статистический анализ корпуса

Для анализа распределения частот токенов были аппроксимированы два закона.

Закон Ципфа показал следующие параметры:

$$C = 2{,}53 \cdot 10^4, \quad s = 0{,}79$$

Значение параметра $s < 1$ указывает на медленное убывание частот, при котором редкие слова составляют значительную часть словаря.

Закон Мандельброта дал параметры:

$$C = 2{,}75 \cdot 10^4, \quad b = 0{,}137, \quad s = 0{,}81$$

Добавление параметра смещения b улучшило аппроксимацию распределения, особенно для наиболее частотных слов.

Токенизация

Функция tokenize выполняет токенизацию входного текста, то есть разбивает строку на отдельные слова (токены), корректно обрабатывая символы в кодировке UTF-8. На первом этапе входная строка в формате std::string (UTF-8) преобразуется в широкую строку std::wstring. Это необходимо для корректной работы с Unicode-символами и использования стандартных функций классификации символов (буквы, цифры и т.д.). Далее устанавливается UTF-8 локаль, что позволяет правильно определять типы символов (буквенно-цифровые или разделители) для многоязычного текста. Затем функция последовательно проходит по каждому символу широкого текста:

- Если символ является буквенно-цифровым (iswalnum), он приводится к нижнему регистру и добавляется к текущему формируемому токену;
- Если встречается любой другой символ (пробел, знак пунктуации, спецсимвол), текущий токен считается завершённым и добавляется в результирующий список, после чего начинается формирование нового токена.

После обработки всего текста функция дополнительно проверяет, остался ли незавершённый токен, и при необходимости добавляет его в результат. Каждый сформированный токен перед добавлением в итоговый массив преобразуется обратно из std::wstring в UTF-8 строку (std::string). Результат работы функции сохраняется в БД. Чтение и запись в БД происходит батчами по 5 документов.

Булев индекс и булев поиск

Сначала строковый запрос разбирается функцией `parse_bool_query`. Она ищет в строке логические операторы `AND` или `OR`, разделяет запрос на левый и правый токены и сохраняет тип операции. Если оператор не найден, запрос помечается как некорректный.

Для работы с результатами поиска используется структура `IntArray`, которая хранит динамический массив идентификаторов документов и его размер.

Функция `get_token_id` по тексту токена выполняет SQL-запрос к базе данных и возвращает его числовой идентификатор. Если токен отсутствует в таблице, возвращается `-1`.

Функция `get_documents_by_token` получает из базы данных список идентификаторов документов, связанных с заданным токеном, и формирует массив `IntArray`.

Логические операции над результатами поиска реализованы отдельными функциями: `intersect` вычисляет пересечение двух массивов документов и возвращает только совпадающие идентификаторы;

`unite` формирует объединение двух массивов, добавляя элементы без дублирования.

Функция `boolean_search` связывает все этапы поиска. Она получает идентификаторы токенов, извлекает соответствующие списки документов и в зависимости от операции (`AND` или `OR`) вызывает пересечение или объединение массивов.

Функция `boolean_search_http` предназначена для использования в HTTP-обработчике. Она вызывает булев поиск, а затем по каждому найденному идентификатору документа запрашивает из базы данных URL страницы и формирует JSON-ответ со списком найденных ссылок.

Результаты

График распределения терминов приведён на рисунке 1.

Для ввода и вывода запросов был реализован HTTP-сервер на C++, принимающий поисковые запросы и возвращающий результаты.

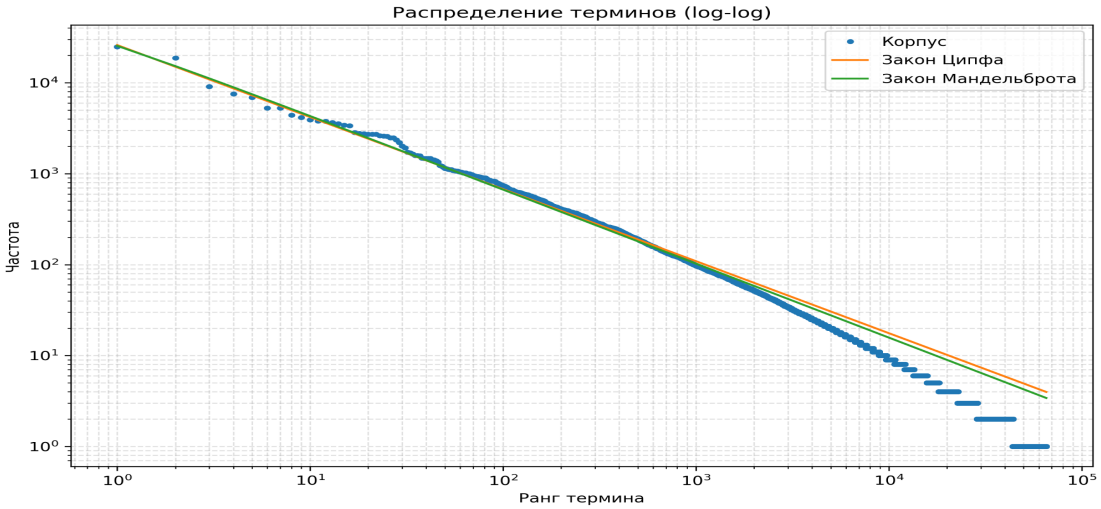

Рис. 1

Примеры запросов и ответов

Примеры запросов на рисунке 2 и 3

```
❯ curl "http://localhost:8081/search?q=Batman"
[
  {
    "score": 40,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/DC_Comics"
  },
  {
    "score": 20,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/DC_Comics#Batman:_Arkham_Series"
  },
  {
    "score": 6,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/Бэтмен_(Arkham_Series)"
  },
  {
    "score": 3,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/%D0%91%D1%8D%D1%82%D0%BC%D0%B5%D0%BD_(Arkham_Series)"
  },
  {
    "score": 3,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/%D0%91%D1%8D%D1%82%D0%BC%D0%B5%D0%BD_(Arkham_Series)?veaction=edit"
  },
  {
    "score": 3,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Batman-Arkham-City.ogg"
  },
  {
    "score": 1,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/%D0%91%D1%8D%D1%82%D0%BC%D0%B5%D0%BD_Gotham_Knight"
  },
  {
    "score": 1,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/%D0%91%D1%8D%D1%82%D0%BC%D0%B5%D0%BD_(DC_Animated_Universe)"
  },
  {
    "score": 1,
    "url": "https://anime-characters-fight.fandom.com/ru/wiki/%D0%91%D1%8D%D1%82%D0%BC%D0%B5%D0%BD_(Batman_vs._Teenage_Mutant_Ninja_Turtles)"
  }
]
```

Рис. 2

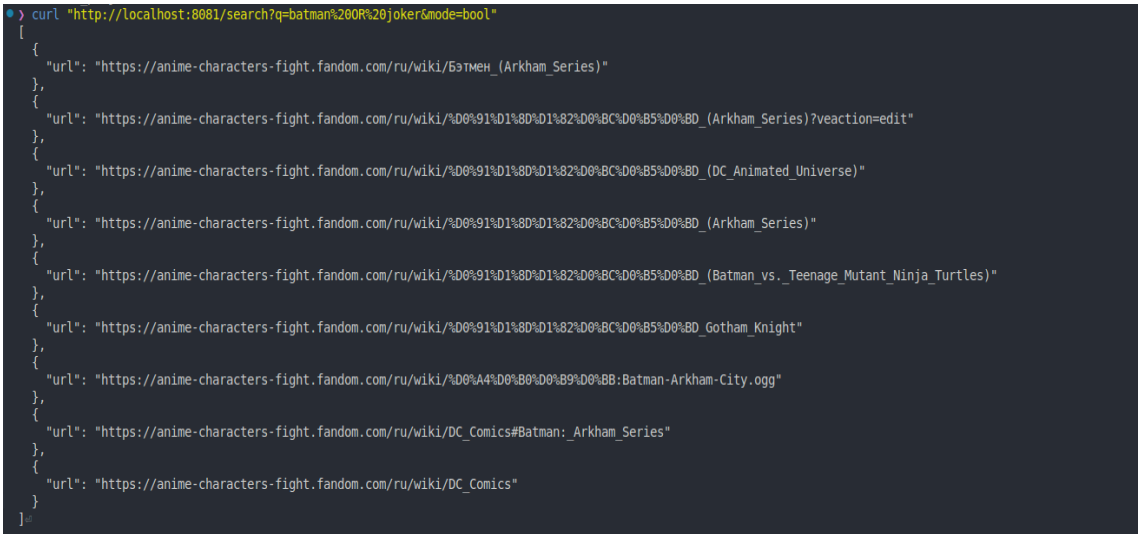

Рис. 3

Вывод

В ходе лабораторной работы был реализован поисковый индекс с поддержкой токенизации и булевого поиска. Анализ корпуса подтвердил статистические свойства текстовых данных, а реализованный токенизатор продемонстрировал высокую производительность и корректность работы.