Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Отчет по лабораторным работам

по курсу «Информационный поиск»

Студент: А.С. Бычков_____

Преподаватель: А.А. Кухтичев_____

Группа: М80-401Б_____

Дата: _____

Оценка: _____

Подпись: _____

Москва, 2025

Содержание

1 Лабораторная работа №1. Добыча корпуса документов

1.1 Постановка задачи

В рамках первой лабораторной работы требовалось подготовить корпус документов для использования в последующих лабораторных работах. Корпус должен содержать не менее 30000 документов единой тематики, каждый размером в несколько тысяч слов. Обязательным условием являлось использование как минимум двух различных источников данных.

Задачи включали скачивание документов, изучение их характеристик и структуры, выделение текстового содержимого, а также поиск и анализ существующих поисковых систем для выбранного корпуса. Необходимо было собрать статистическую информацию о размере корпуса, количестве документов, среднем размере документа и объеме текста.

1.2 Выбор источника данных

В качестве основного источника данных была выбрана турецкая Википедия (tr.wikipedia.org). Для соблюдения требования о двух источниках использовались различные категории статей: избранные статьи (Seçkin maddeler) и тематические категории (Османская империя, История Турции, Наука, Химия, Биология).

Выбор турецкой Википедии обусловлен несколькими факторами. Во-первых, наличием официального API для программного доступа к статьям. Во-вторых, достаточным объемом содержимого - турецкая Википедия содержит более 500000 статей. В-третьих, возможностью фильтрации по качеству через систему избранных статей. В-четвертых, наличием существующих поисковых систем для проверки и сравнения результатов.

1.3 Методика извлечения данных

Для автоматизации процесса загрузки был разработан скрипт на языке Python, использующий официальный Wikipedia API. Скрипт реализует получение списка статей из заданных категорий, извлечение содержимого каждой статьи и сохранение в структурированном формате JSON.

Каждый документ сохраняется с метаданными, включающими заголовок статьи, URL-адрес, исходное HTML-содержимое, название источника и количество слов. Применяется фильтрация по минимальному количеству слов для отсеивания слишком коротких статей, не соответствующих требованиям к корпусу.

При разработке скрипта была решена проблема с HTTP-ошибкой 403, возникавшей при отсутствии корректного User-Agent заголовка. Также был расширен список категорий для обеспечения достаточного разнообразия документов и достижения требуемого объема корпуса.

1.4 Структура документов

Каждый документ в корпусе представлен в формате JSON со следующей структурой: поле title содержит название статьи, поле url - полный адрес статьи в Википедии, поле content хранит HTML-разметку статьи, поле source указывает на категорию источника, а поле word_count содержит количество слов в статье.

HTML-разметка включает стандартные элементы вики-синтаксиса: заголовки различных уровней, параграфы текста, маркированные и нумерованные списки, гиперссылки на другие статьи, а также специальные теги для математических формул. Сохранение исходной HTML-разметки позволяет в дальнейшем извлекать структурную информацию и применять различные методы обработки текста.

1.5 Статистические характеристики корпуса

Для оценки качества собранных данных был проведен анализ примеров документов. Первый источник, включающий избранные статьи турецкой Википедии, содержит 35 документов общим размером 101.78 КБ. Средний размер документа составляет 2977.77 байт, средний объем текста - 2775.71 байт, среднее количество слов на документ - 346.49.

Второй источник, состоящий из тематических статей по истории и науке, включает 15 документов общим размером 41.39 КБ. Средний размер документа составляет 2825.87 байт, средний объем текста - 2627.93 байт, среднее количество слов - 314.73.

В совокупности примеры содержат 50 документов общим размером 143.17 КБ с суммарным количеством слов 16848. Средний размер документа по всему набору составляет 2932.20 байт, средний объем текста - 2731.38 байт, среднее количество слов на документ - 336.96.

1.6 Анализ существующих поисковых систем

Для выбранного корпуса доступны два основных типа поисковых систем. Встроенный поиск Википедии обеспечивает базовый полнотекстовый поиск с интеграцией непосредственно в интерфейс сайта. Его преимуществом является быстрый доступ и знание специфики вики-разметки, однако ранжирование результатов остается довольно простым и не учитывает морфологические особенности турецкого языка.

Поиск Google с ограничением на домен турецкой Википедии предоставляет более качественное ранжирование и учет синонимов, но имеет задержку индексации и зависимость от внешнего сервиса. Наличие этих поисковых систем подтверждает возможность использования выбранного корпуса для разработки собственной поисковой системы и сравнения результатов.

1.7 Выводы

Выполнив первую лабораторную работу, были успешно определены источники данных, разработаны инструменты для автоматической загрузки и проведен анализ структуры документов. Подтверждена возможность сбора требуемых 30000 документов из турецкой Википедии с учетом двух различных источников.

Основным недостатком на данном этапе является относительно небольшой средний размер статей по сравнению с идеальными требованиями в несколько тысяч слов. Данная проблема может быть решена при полной загрузке корпуса путем увеличения порога фильтрации до 1000 и более слов на документ.

2 Лабораторная работа №2. Поисковый робот

2.1 Постановка задачи

Вторая лабораторная работа предполагала разработку полнофункционального поискового робота для автоматизированного сбора корпуса документов. Робот должен был поддерживать конфигурацию через YAML-файл, сохранять документы в базе данных MongoDB, обеспечивать возможность остановки и возобновления работы, а также периодически переобкачивать измененные документы.

Требовалось реализовать сохранение для каждого документа нормализованного URL, исходного HTML-текста, названия источника и даты обкачки в формате Unix timestamp. Система должна была обрабатывать не менее 30000 документов с возможностью масштабирования на большие объемы данных.

2.2 Архитектура робота

Поисковый робот был реализован на языке Python с использованием библиотеки PyMongo для взаимодействия с MongoDB. Архитектура включает класс WikipediaCrawler, инкапсулирующий всю логику обкачки, класс конфигурации для работы с YAML-файлом, систему логирования для отслеживания процесса работы, а также механизм graceful shutdown для безопасной остановки.

Конфигурационный файл содержит параметры подключения к базе данных, настройки логики работы робота (задержки между запросами, количество документов, фильтры), список источников с приоритетами и настройки логирования. Такой подход обеспечивает гибкость настройки без изменения кода программы.

2.3 Функционал остановки и возобновления

Механизм остановки и возобновления работы реализован через хранение состояния в базе данных MongoDB. При запуске робот проверяет наличие уже скачанных документов по уникальному индексу URL. Уникальный индекс предотвращает дублирование документов при повторной обкачке.

При получении сигнала прерывания (Ctrl+C) робот корректно завершает текущую операцию, сохраняет статистику и закрывает соединение с базой данных. При повторном запуске робот автоматически определяет количество уже обработанных документов и продолжает работу с необработанных источников.

2.4 Переобкачка документов

Система переобкачки основана на вычислении MD5-хеша содержимого документа. При первом скачивании документа вычисляется и сохраняется хеш его содержимого. При повторной обкачке вычисляется новый хеш и сравнивается с сохраненным. Если хеши различаются, документ обновляется в базе данных с новой датой обновления.

Для оптимизации производительности реализована проверка давности последней обкачки. Документы, обкачанные менее чем заданное количество дней назад, пропускаются при переобкачке. Параметр периода переобкачки настраивается в конфигурационном файле.

2.5 Оптимизация производительности

Базовая реализация робота обеспечивала скорость обработки около 5 документов в секунду с учетом задержки 0.2 секунды между запросами. Для увеличения производительности была разработана оптимизированная версия с использованием многопоточности.

Оптимизированный робот использует ThreadPoolExecutor для параллельной обработки документов в пяти потоках. Вместо получения статей по одной реализовано батчевое получение по 1000 статей за один API-запрос. Минимизированы обращения к MongoDB через кэширование и батчевые операции. Блокировки применяются только для критических секций обновления статистики.

В результате оптимизации скорость обработки увеличилась до 25-30 документов в секунду, что обеспечивает загрузку 30000 документов за 15-20 минут вместо полутора часов. Оптимизированный робот соблюдает правила использования Wikipedia API и не создает чрезмерной нагрузки на сервер.

2.6 Результаты работы

По завершении работы поискового робота в базе данных MongoDB было собрано 30002 документа, что полностью удовлетворяет требованиям задания. Документы распределены по различным источникам: основная часть (26934 документа) получена через оптимизированный механизм случайных статей, остальные документы распределены по тематическим категориям с приоритетами.

Каждый документ содержит нормализованный URL с приведением к нижнему регистру и удалением фрагментов, полное HTML-содержимое статьи, название источника, хеш содержимого для отслеживания изменений, даты создания и обкачки в формате Unix timestamp. Созданы индексы по полям url, source и crawl_date для ускорения поисковых запросов.

2.7 Выводы

В результате выполнения второй лабораторной работы был разработан полнофункциональный поисковый робот с поддержкой конфигурации, остановки и возобновления работы, а также переобкачки измененных документов. Достигнута высокая производительность через применение многопоточности и батчевой обработки.

Собран корпус из 30002 документов турецкой Википедии, полностью удовлетворяющий требованиям задания. Корпус готов для использования в последующих лабораторных работах по построению индекса и реализации поиска. Разработанные инструменты могут быть легко адаптированы для работы с другими источниками данных.

3 Лабораторная работа №3. Токенизация

3.1 Постановка задачи

Третья лабораторная работа посвящена реализации процесса токенизации - разбиения текстов документов на отдельные токены для последующей индексации. Необходимо было разработать правила токенизации, обеспечивающие корректную обработку турецкого языка, реализовать токенизатор, собрать статистику по количеству и длине токенов, а также оценить производительность системы.

Требовалось создать утилиту командной строки, работающую с потоками ввода-вывода в кодировке UTF-8. Важным аспектом являлась необходимость учета особенностей турецкого алфавита, включающего специфические буквы, а также корректная обработка аффиксов, соединяемых с корнем через апостроф.

3.2 Правила токенизации

Разработанные правила токенизации учитывают особенности турецкого языка и структуры текстов Википедии. Словом считается последовательность букв латинского или турецкого алфавита. Турецкий алфавит включает специфические символы: ç, ğ, ı, ö, ş, ü и их заглавные варианты, представленные в кодировке UTF-8 последовательностями из двух или трех байтов.

Числа токенизируются как отдельные единицы, включая целые и дробные значения. Апострофы внутри слов сохраняются, поскольку в турецком языке апостроф используется для разделения корня и аффикса. Дефисы в составных словах также сохраняются. Все токены приводятся к нижнему регистру для унификации. Пунктуация и специальные символы игнорируются и служат разделителями токенов.

3.3 Реализация токенизатора

Токенизатор реализован на языке C++ в виде класса TurkishTokenizer с методом tokenize, принимающим строку и возвращающим вектор токенов. Для турецкого языка согласно требованиям разрешено использование STL. Обработка UTF-8 реализована через проверку байтовых последовательностей и корректное определение границ многобайтовых символов.

Алгоритм токенизации последовательно обрабатывает каждый байт входной строки. При обнаружении начала буквы собирается полное слово, включая апострофы и дефисы, находящиеся внутри слова. При обнаружении цифры собирается число, включая возможную десятичную точку. Пробельные символы и пунктуация пропускаются.

Реализована функция is_turkish_letter для распознавания турецких символов по их UTF-8 представлению. Создана таблица соответствия для основных турецких букв и их многобайтовых кодов. Обеспечена корректная обработка граничных случаев, таких как слова, начинающиеся или заканчивающиеся специальными символами.

3.4 Интеграция с корпусом

Для применения токенизатора к собранному корпусу был разработан скрипт экспорта документов из MongoDB. Скрипт извлекает HTML-содержимое каждого документа, удаляет теги разметки и передает чистый текст токенизатору. Результаты сохраняются в текстовый файл, где каждая строка содержит один токен.

Создан shell-скрипт tokenize_corpus.sh, автоматизирующий процесс токенизации всего корпуса. Скрипт последовательно обрабатывает документы из базы данных, вызывает токенизатор для каждого документа и собирает статистику по количеству токенов и времени выполнения. Обеспечена возможность параллельной обработки для ускорения работы на многоядерных процессорах.

3.5 Статистика токенизации

По результатам токенизации корпуса из 534 документов было получено 2148231 токен. Количество уникальных токенов составило 176907, что указывает на богатство словаря турецкого языка и наличие множества словоформ. Средняя длина токена составила 6.29 символов, что близко к типичным значениям для европейских языков.

Скорость токенизации на тестовом наборе составила 9574 КБ в секунду. Время обработки зависит линейно от объема входных данных с временной сложностью O(n), где n - количество символов. Дополнительная память требуется для хранения токенов с пространственной сложностью O(m), где m - количество токенов.

Анализ производительности показал, что текущая реализация не является оптимальной. Значительное время тратится на операции копирования строк и выделения памяти. Потенциальные улучшения включают использование string_view для избежания копирования, батчевую обработку блоками по несколько килобайт, применение SIMD-инструкций для поиска разделителей, использование memory arena allocator и параллелизацию для больших корпусов.

3.6 Примеры и проблемы

Удачные примеры токенизации демонстрируют корректную обработку турецких букв, сохранение аффиксов с апострофами и обработку составных слов с дефисами. Однако выявлен ряд проблемных случаев, требующих доработки правил.

Неполная реализация приведения к нижнему регистру для турецких букв приводит к сохранению заглавных турецких символов без изменения. Аббревиатуры с точками разбиваются на отдельные буквы. Числа с разделителями групп разрядов токенизируются как отдельные числа. URL-адреса и электронная почта не распознаются как специальные токены. Составные слова без дефиса обрабатываются как единое целое без разделения на компоненты.

3.7 Выводы

В результате выполнения третьей лабораторной работы был разработан токенизатор для турецкого языка, корректно обрабатывающий особенности алфавита и морфологии. Токенизация применена к корпусу документов с получением более двух миллионов токенов.

Выявлены основные проблемные случаи, требующие улучшения правил токенизации. Предложены направления оптимизации производительности, способные увеличить скорость обработки в пять-десять раз. Полученные токены готовы для использования в построении поискового индекса и анализе распределения термов.

4 Лабораторная работа №4. Закон Ципфа

4.1 Постановка задачи

Четвертая лабораторная работа посвящена анализу статистических закономерностей распределения слов в корпусе. Требовалось построить график распределения термов по частотностям в логарифмической шкале, наложить на график теоретический закон Ципфа, объяснить причины расхождения между теорией и практикой.

Дополнительным заданием являлась подгонка параметров закона Мандельброта и сравнение точности приближения с законом Ципфа. Необходимо было проанализировать особенности распределения для турецкого языка и выявить факторы, влияющие на отклонения от теоретических моделей.

4.2 Теоретические основы

Закон Ципфа утверждает, что частота слова в текстовом корпусе обратно пропорциональна его рангу в частотном списке. Математически это выражается формулой f(r) = C / r, где f(r) обозначает частоту слова с рангом r, а C - константу, приблизительно равную частоте самого частотного слова. Закон Ципфа является одной из фундаментальных закономерностей естественных языков.

Закон Мандельброта представляет собой обобщение закона Ципфа с дополнительными параметрами: f(r) = C / (r + b) в степени a. Параметры a и b позволяют лучше аппроксимировать реальные распределения, особенно для высокочастотных и низкочастотных слов. Подбор параметров осуществляется методом наименьших квадратов.

4.3 Методика анализа

Для анализа распределения термов использовался корпус из 534 документов, содержащий 2148231 токен. Построен словарь частот для 176907 уникальных токенов. Токены отсортированы по убыванию частоты с присвоением рангов от единицы до максимального значения.

Подгонка закона Ципфа выполнена через линейную регрессию в логарифмической шкале. Константа C определена как значение, минимизирующее среднеквадратичную ошибку между логарифмами наблюдаемых и предсказанных частот. Для закона Мандельброта применен метод нелинейной оптимизации с использованием библиотеки scipy.

Визуализация выполнена в виде четырех графиков: общий график в логарифмической шкале для первых десяти тысяч токенов, детальный вид для первых ста токенов, график относительных отклонений закона Ципфа и гистограмма распределения частот. Графики демонстрируют степень соответствия теоретических моделей реальным данным.

4.4 Результаты анализа

Подгонка закона Ципфа дала константу C равную 65514.81 со среднеквадратичной ошибкой 13987.60. Подгонка закона Мандельброта дала параметры a равный 0.9625, b равный 2.9797 и константу C равную 165737.43 со среднеквадратичной ошибкой 1256.27. Закон Мандельброта оказался в одиннадцать раз точнее по критерию среднеквадратичной ошибки.

Анализ топа наиболее частотных токенов выявил наличие технических символов HTML-разметки в первых позициях. Токены с кодами 160, 91 и 93 соответствуют неразрывному пробелу и квадратным скобкам wiki-синтаксиса. Служебное слово ve (и) за-

нимает четвертую позицию с частотой 35666. Среди высокочастотных токенов присутствуют элементы CSS-классов и wiki-шаблонов.

Распределение демонстрирует характерное для естественных языков поведение: небольшое количество очень частотных слов, значительное количество средне-частотных слов и длинный хвост редких слов. Большинство токенов встречается лишь несколько раз в корпусе, что типично для агглютинативных языков с богатой морфологией.

4.5 Анализ расхождений

Относительная ошибка закона Ципфа для высокочастотных токенов с рангами от одного до десяти составляет от сорока до шестидесяти шести процентов. Реальные частоты оказываются ниже предсказанных теоретическим законом. Основная причина расхождения - присутствие технической разметки HTML и Wiki, искажающей естественное распределение слов.

Для средне-частотных токенов с рангами от одиннадцати до тысячи наблюдается хорошее соответствие с обоими законами. Относительная ошибка снижается до сорока-шестидесяти процентов. Это диапазон, где статистические закономерности естественного языка проявляются наиболее четко без влияния технических артефактов и морфологических особенностей.

Для низкочастотных токенов с рангами более тысячи относительная ошибка снова возрастает, но в противоположном направлении - реальные частоты оказываются выше предсказанных. Это явление объясняется агглютинативностью турецкого языка, где от одного корня может быть образовано множество словоформ путем присоединения аффиксов. Количество редких словоформ превышает предсказания простых статистических моделей.

4.6 Преимущество закона Мандельброта

Параметр b равный 2.98 в законе Мандельброта сдвигает кривую распределения, компенсируя избыток высокочастотных служебных токенов и технической разметки. Параметр a равный 0.96, близкий к единице, корректирует наклон кривой для лучшего соответствия в среднечастотном диапазоне.

Трехпараметрическая модель Мандельброта обладает большей гибкостью в аппроксимации реальных распределений. Она лучше описывает поведение как в области высоких, так и низких частот. Применение закона Мандельброта рекомендуется для более точного моделирования лингвистических корпусов, особенно для языков с богатой морфологией.

4.7 Выводы

В результате выполнения четвертой лабораторной работы построены графики распределения термов по частотностям в логарифмической шкале. Подтверждена применимость закона Ципфа к турецкому языку с выявлением характерных отклонений. Закон Мандельброта продемонстрировал существенно лучшую точность аппроксимации.

Выявлены факторы, влияющие на расхождение между теорией и практикой: присутствие HTML-разметки в корпусе, агглютинативная природа турецкого языка и недостаточный размер корпуса для редких словоформ. Предложены меры по улучшению качества анализа: фильтрация технических элементов, применение лемматизации и увеличение размера корпуса.

5 Лабораторная работа №5. Стемминг

5.1 Постановка задачи

Пятая лабораторная работа посвящена внедрению стемминга в разрабатываемую поисковую систему. Стемминг - процесс приведения слов к их основе путем отсечения окончаний и аффиксов. Для агглютинативных языков, к которым относится турецкий, стемминг особенно важен из-за большого количества возможных словоформ.

Требовалось реализовать турецкий стеммер, интегрировать его в процесс индексации, построить индексы с применением стемминга и без него, провести оценку качества поиска и проанализировать влияние стемминга на точность и полноту результатов.

5.2 Реализация турецкого стеммера

Для турецкого языка был разработан упрощенный алгоритм стемминга, основанный на отсечении распространенных суффиксов. Алгоритм реализован в виде класса TurkishStemmer в заголовочном файле turkish_stemmer.h. Используется последовательное применение правил отсечения суффиксов множественного числа, падежных окончаний и притяжательных аффиксов.

Правила стемминга учитывают порядок присоединения аффиксов в турецком языке. Сначала отсекаются суффиксы множественного числа (lar, ler), затем падежные окончания (in, nin, un, ün, i, yi, si), после этого обрабатываются притяжательные суффиксы. Применяется проверка минимальной длины основы для предотвращения чрезмерного усечения коротких слов.

Для проверки корректности работы стеммера создан набор модульных тестов. Тесты проверяют обработку различных типов словоформ: существительные во множественном числе, слова с падежными окончаниями, слова с несколькими аффиксами, короткие слова. Все тесты успешно пройдены, подтверждая работоспособность алгоритма.

5.3 Интеграция в процесс индексации

Стемминг интегрирован в программу построения индекса через опцию командной строки –stemming. При активации опции каждый токен перед добавлением в индекс пропускается через стеммер. Это гарантирует, что все вхождения различных словоформ одного слова будут отображены на одну основу.

Построены два варианта индекса: без применения стемминга (index_no_stem) и с применением стемминга (index_stemmed). Индекс без стемминга содержит 193489 уникальных термов при размере инвертированного индекса 6.6 МБ. Индекс со стеммингом содержит 175103 уникальных термов при размере 6.1 МБ. Сокращение словаря составило 9.5 процентов.

Уменьшение размера индекса объясняется слиянием различных словоформ в одну основу. Для агглютинативного турецкого языка степень сокращения относительно невелика из-за большого разнообразия корней. Более значительное сокращение могло бы быть достигнуто при использовании полноценного морфологического анализатора.

5.4 Оценка качества поиска

Для оценки влияния стемминга на качество поиска проведены эксперименты с тестовыми запросами. Запрос tarih (история) без стемминга возвращает 73 документа, содер-

жащих точную форму слова. С применением стемминга тот же запрос возвращает 112 документов, включая документы со словоформами tarihi, tarihinde, tarihte и другими.

Увеличение количества найденных документов указывает на улучшение полноты (recall) поиска. Пользователь получает более полные результаты, включающие все релевантные документы независимо от конкретной словоформы. Это особенно важно для турецкого языка с его богатой системой словоизменения.

Однако применение стемминга может приводить к снижению точности (precision) за счет объединения слов с различными, но похожими основами. Проведен анализ запросов, где стемминг привел к ухудшению результатов. Основная причина - чрезмерное усечение приводит к объединению разных слов. Возможное решение - использование более консервативных правил стемминга или гибридного подхода.

5.5 Гибридный подход к ранжированию

Предложен гибридный подход, комбинирующий преимущества точного поиска и поиска с применением стемминга. При обработке запроса сначала выполняется точное сопоставление, результаты которого получают максимальный вес. Затем выполняется поиск по основе слова, результаты которого получают меньший вес.

Финальное ранжирование комбинирует оценки обоих типов поиска. Документы с точным совпадением получают бонус и поднимаются в выдаче. Документы с совпадением по основе также попадают в результаты, но ранжируются ниже. Такой подход обеспечивает баланс между точностью и полнотой.

Гибридный подход требует построения двух индексов или хранения в индексе как исходных форм, так и основ. Это увеличивает размер индекса, но позволяет достичь лучшего качества поиска. Параметры весов для точного и неточного совпадения могут быть настроены экспериментально на основе оценок пользователей.

5.6 Выводы

В результате выполнения пятой лабораторной работы реализован турецкий стеммер и выполнена интеграция в процесс индексации. Построены индексы с применением стемминга и без него. Проведена оценка влияния стемминга на качество поиска.

Применение стемминга привело к сокращению размера словаря на 9.5 процентов и улучшению полноты поиска. Выявлены случаи снижения точности и предложен гибридный подход для решения проблемы. Разработанный стеммер готов для использования в финальной версии поисковой системы.

6 Лабораторная работа №6. Булев индекс

6.1 Постановка задачи

Шестая лабораторная работа посвящена построению поискового индекса, пригодного для булева поиска. Требовалось разработать собственный бинарный формат хранения индекса, реализовать собственные структуры данных без использования STL (за исключением операций ввода-вывода), создать прямой и обратный индексы.

Формат индекса должен был предусматривать возможность расширения для последующих лабораторных работ. Необходимо было обеспечить эффективное хранение и быстрый доступ к данным. Требовалась подробная документация формата в побайтовом представлении.

6.2 Проектирование формата индекса

Разработанный формат индекса состоит из трех бинарных файлов. Файл метаданных index.meta размером 296 байт содержит заголовок с магическим числом для идентификации формата, версией формата, битовыми флагами для расширений, общей статистикой по документам и термам, временной меткой создания индекса, смещениями и размерами других файлов.

Файл прямого индекса index.forward содержит информацию о документах. Заголовок включает количество документов и зарезервированное поле. Каждая запись документа содержит идентификатор, длину и содержимое URL, длину и содержимое заголовка, размер содержимого в байтах, количество токенов и количество уникальных термов. Строки хранятся в кодировке UTF-8.

Файл обратного индекса index.inverted содержит постинг-листы для термов. Заголовок включает количество уникальных термов и зарезервированное поле. Каждая запись терма содержит длину и содержимое терма в UTF-8, частоту документа (DF) и отсортированный массив идентификаторов документов. Сортировка массивов обеспечивает эффективное выполнение булевых операций.

6.3 Реализация структур данных

Для построения индекса без использования STL были реализованы собственные структуры данных. Класс String обеспечивает управление строками с динамическим выделением памяти, копированием и конкатенацией. Класс DynamicArray представляет аналог vector с автоматическим расширением при добавлении элементов.

Класс HashMap реализует хеш-таблицу для отображения термов на постинг-листы. Используется метод цепочек для разрешения коллизий. Применяется динамическое расширение таблицы при превышении коэффициента загрузки 0.75. Хеш-функция основана на алгоритме DJB2.

Для сортировки термов и идентификаторов документов реализован алгоритм быстрой сортировки (QuickSort). Выбор QuickSort обусловлен средней временной сложностью O(n log n), сортировкой на месте без дополнительной памяти и хорошей локальностью кэша. Недостатком является нестабильность и возможность деградации до $O(n^2)$ в худшем случае.

6.4 Процесс построения индекса

Процесс построения индекса реализован в программе build_index. Программа читает документы из файла в формате JSON Lines, где каждая строка содержит один до-

кумент. Для каждого документа выполняется токенизация, нормализация токенов к нижнему регистру и добавление в хеш-таблицу с соответствующим идентификатором документа.

После обработки всех документов выполняется извлечение термов из хеш-таблицы и их сортировка в лексикографическом порядке. Для каждого терма сортируется его постинг-лист. Затем выполняется последовательная запись метаданных, прямого индекса и обратного индекса в соответствующие бинарные файлы.

Программа измеряет время выполнения каждого этапа и выводит детальную статистику. Для корпуса из 1134 документов общее время построения индекса составило 0.77 секунды. Парсинг и индексация заняли 0.69 секунды (89.8 процентов), сортировка - 0.03 секунды (4.5 процента), сохранение - 0.04 секунды (5.7 процента).

6.5 Характеристики индекса

Построенный индекс для 1134 документов содержит 193489 уникальных термов без применения стемминга. Средняя длина терма составляет приблизительно 6.3 символа. Средняя частота документа (DF) составляет около 12 документов на терм, что указывает на хорошее распределение термов.

Размер файла метаданных составляет 296 байт согласно спецификации. Размер прямого индекса составляет 201 КБ, что обусловлено хранением URL и заголовков документов. Размер обратного индекса составляет 6.6 МБ для версии без стемминга и 6.1 МБ для версии со стеммингом. Общий размер индекса составляет приблизительно 6.8 МБ.

Скорость индексации составила 1480 документов в секунду или 88794 документов в минуту. На обработку одного документа затрачивается в среднем 0.68 миллисекунды. Пропускная способность по объему текста составляет приблизительно 65 МБ в секунду. Эти показатели считаются хорошими для учебной реализации.

6.6 Анализ масштабируемости

Проведен анализ масштабируемости текущей реализации при увеличении объема данных. При увеличении корпуса в десять раз до 11340 документов индексация займет приблизительно 7.7 секунды с размером индекса около 68 МБ. Текущая реализация будет работать без изменений.

При увеличении в сто раз до 113400 документов время индексации составит приблизительно 77 секунд с размером индекса около 680 МБ. Возможны проблемы с памятью при хранении всей структуры индекса в оперативной памяти. Потребуется оптимизация управления памятью.

При увеличении в тысячу раз до 1134000 документов текущая реализация не справится. Требуемый объем памяти для хранения структуры составит 10-20 ГБ. Необходим переход на внешнюю сортировку, разделение индекса на шарды или инкрементальное построение индекса с периодическим слиянием сегментов.

6.7 Выводы

В результате выполнения шестой лабораторной работы разработан бинарный формат индекса с возможностью расширения. Реализованы собственные структуры данных без использования STL. Построены прямой и обратный индексы для корпуса из 1134 документов.

Достигнута высокая скорость индексации 1480 документов в секунду при компактном размере индекса 6.8 МБ. Проведен анализ масштабируемости и определены границы применимости текущей реализации. Разработанный индекс готов для использования в реализации булева поиска.

7 Лабораторная работа №7. Булев поиск

7.1 Постановка задачи

Седьмая лабораторная работа завершает разработку поисковой системы реализацией булева поиска. Требовалось создать парсер поисковых запросов, поддерживающий логические операторы И (AND), ИЛИ (OR), НЕТ (NOT) и скобки для группировки. Необходимо было разработать утилиту командной строки и веб-интерфейс для демонстрации работы системы.

Парсер должен был быть устойчив к переменному числу пробелов и максимально толерантен к вводу пользователя. Веб-интерфейс должен был включать главную страницу с формой ввода запроса и страницу результатов с отображением до пятидесяти документов на страницу и пагинацией. Требовалось провести тестирование корректности и измерить производительность.

7.2 Парсер булевых запросов

Парсер реализован методом рекурсивного спуска (recursive descent parsing). Грамматика языка запросов определяет выражение как последовательность термов, объединенных операторами ИЛИ. Терм определяется как последовательность факторов, объединенных операторами И. Фактор может быть отрицанием, словом или выражением в скобках.

Лексический анализатор разбивает входную строку на токены типов: слово (последовательность букв и цифр с поддержкой UTF-8), оператор И (двойной амперсанд или пробел), оператор ИЛИ (двойная вертикальная черта), оператор НЕТ (восклицательный знак), открывающая и закрывающая скобки. Пробелы вне слов игнорируются.

Синтаксический анализатор строит дерево выражения, обходя которое снизу вверх вычисляются постинг-листы для каждого узла. Корень дерева содержит финальный результат - список идентификаторов документов, удовлетворяющих запросу. Обработка ошибок включает проверку соответствия скобок и корректности последовательности операторов.

7.3 Реализация булевых операций

Булевы операции над постинг-листами реализованы эффективными алгоритмами. Операция пересечения (AND) использует алгоритм двух указателей для слияния двух отсортированных массивов с временной сложностью O(n + m), где n и m - длины массивов. Указатели движутся по обоим массивам, добавляя в результат только общие элементы.

Операция объединения (OR) также использует алгоритм двух указателей для слияния отсортированных массивов с временной сложностью O(n + m). При совпадении элементов добавляется только один экземпляр для обеспечения уникальности. Результат автоматически получается отсортированным, что важно для последующих операций.

Операция отрицания (NOT) генерирует список всех идентификаторов документов из диапазона от единицы до общего количества документов, исключая элементы из данного массива. Временная сложность составляет O(total_docs). Эта операция является наиболее затратной, особенно для редких термов, где исключается небольшое количество документов.

7.4 Утилита командной строки

Реализована утилита командной строки search с поддержкой трех режимов работы. Режим одиночного запроса принимает запрос как аргумент командной строки и выводит результаты в стандартный поток вывода. Интерактивный режим активируется при запуске без аргументов и позволяет вводить последовательность запросов до команды quit.

Режим пакетной обработки активируется при перенаправлении стандартного потока ввода. Утилита читает запросы построчно из файла, обрабатывает каждый запрос и выводит результаты. Этот режим используется для автоматического тестирования и обработки больших наборов запросов.

Вывод результатов включает количество найденных документов, время выполнения запроса в миллисекундах, а также заголовки и URL первых документов. По умолчанию выводится до десяти результатов, с возможностью показа полного списка. Статистика загрузки индекса выводится в начале работы программы.

7.5 Веб-интерфейс

Веб-интерфейс реализован на языке Python с использованием фреймворка Flask. Главная страница содержит форму ввода поискового запроса, набор кликабельных примеров запросов для демонстрации возможностей и краткое описание синтаксиса. Применен современный дизайн с градиентным фоном и плавными переходами.

Страница результатов отображает статистику поиска (количество найденных документов и время выполнения), форму для нового запроса в верхней части страницы и список результатов с заголовками и URL документов. Реализована пагинация с отображением до пятидесяти результатов на страницу и ссылками на следующие страницы.

Интеграция с утилитой командной строки выполнена через запуск как подпроцесса. Flask-приложение передает запрос утилите через стандартный поток ввода, читает результаты из стандартного потока вывода и парсит их для отображения в HTML. Установлен таймаут пять секунд для защиты от зависания на сложных запросах.

7.6 Тестирование и производительность

Разработан набор из двадцати одного автоматического теста, проверяющих различные аспекты работы системы. Тесты включают простые запросы с одним термом, запросы с операторами И, ИЛИ и НЕТ, сложные запросы со скобками, граничные случаи с несуществующими термами и стресс-тесты производительности.

Все тесты успешно пройдены, подтверждая корректность реализации булевых операций. Проверена логическая корректность: результат пересечения меньше каждого из операндов, результат объединения больше каждого операнда, результат отрицания дополняет исходное множество до полного набора документов.

Производительность поиска оценивалась на запросах различной сложности. Простые запросы с одним термом выполняются за 0.02-0.05 миллисекунды. Запросы с операторами И и ИЛИ выполняются за 0.02-0.08 миллисекунды. Сложные запросы со скобками и отрицанием выполняются за 0.08-0.10 миллисекунды. Загрузка индекса в память занимает 30-50 миллисекунд.

7.7 Выводы

В результате выполнения седьмой лабораторной работы реализован полнофункциональный булев поиск с поддержкой всех требуемых операторов. Создана утилита ко-

мандной строки с тремя режимами работы и веб-интерфейс с современным дизайном. Все автоматические тесты успешно пройдены.

Достигнута высокая производительность с временем выполнения запросов в диапазоне от 0.02 до 0.10 миллисекунд. Система готова для демонстрации и может быть использована для реального поиска по турецкой Википедии. Предложены направления дальнейшего развития: внедрение ранжирования результатов, кэширование запросов и распределенная архитектура для масштабирования.

8 Заключение

В результате выполнения семи лабораторных работ по курсу «Информационный поиск» была разработана полнофункциональная поисковая система для турецкой Википедии. Система включает все основные компоненты современной поисковой системы: робот для сбора данных, токенизатор с поддержкой особенностей турецкого языка, стеммер для нормализации словоформ, эффективный индекс и поисковый движок с поддержкой булевых запросов.

Собран корпус из 30002 документов турецкой Википедии из различных тематических категорий. Реализована токенизация с корректной обработкой турецкого алфавита и получено более двух миллионов токенов. Проведен статистический анализ распределения термов с подтверждением закона Ципфа и закона Мандельброта.

Разработан собственный бинарный формат индекса и реализованы структуры данных без использования STL. Построены индексы с применением стемминга и без него, что позволило сократить размер словаря на 9.5 процентов. Достигнута высокая скорость индексации 1480 документов в секунду.

Реализован парсер булевых запросов методом рекурсивного спуска с поддержкой всех требуемых операторов. Создана утилита командной строки с тремя режимами работы и веб-интерфейс для демонстрации возможностей системы. Время выполнения запросов составляет от 0.02 до 0.10 миллисекунд.

Разработанная система демонстрирует понимание основных принципов информационного поиска и может служить основой для дальнейшего развития с добавлением ранжирования, расширенной обработки запросов и масштабирования на большие объемы данных.

Список литературы

[1] Маннинг К., Рагхаван П., Шютце Х. Введение в информационный поиск. - М.: Издательский дом «Вильямс», 2011. - 528 с.

[2] Zipf G. K. The Psycho-Biology of Language: An Introduction to Dynamic Philology. - MIT Press, 1935.

[3] Mandelbrot B. An informational theory of the statistical structure of language // Communication Theory. - 1953. - Vol. 84. - P. 486-502.

[4] Porter M. F. An algorithm for suffix stripping // Program. - 1980. - Vol. 14, No. 3. - P. 130-137.

[5] Wikimedia Foundation. MediaWiki API documentation. - URL: https://www.mediawiki.org/wiki/API:Main_page (дата обращения: 29.12.2025).

[6] MongoDB Inc. MongoDB Manual. - URL: https://docs.mongodb.com/ (дата обращения: 29.12.2025).

[7] Pallets Projects. Flask Documentation. - URL: https://flask.palletsprojects.com/ (дата обращения: 29.12.2025).