Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Информационный поиск»

Студент: Я В. Ермаков
Преподаватель: А. А. Кухтичев
Группа: М8О-407Б

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

Целью работы является подготовка корпуса документов для последующего использования в лабораторных работах по информационному поиску. В ходе выполнения работы требуется выбрать не менее двух источников документов, загрузить примеры документов, изучить структуру данных, выделить текст из “сырого” формата и собрать статистические характеристики корпуса.

Реализация

В качестве источников данных были выбраны два независимых открытых научных источника: **ACL Anthology** и **arXiv**. Эти источники подходят для задач поиска, так как содержат большое количество документов с устойчивыми URL, имеют доступ к HTML-страницам публикаций.

Для первой ЛР использовалась утилита export_corpus.py, предназначенная для подготовки корпуса документов к дальнейшим лабораторным работам. Программа подключается к MongoDB, выбирает последнюю сохранённую версию HTML-документа для каждого URL, извлекает из HTML видимый текст и заголовок страницы, нормализует пробелы и сохраняет очищенный текст в файлы docs/*.txt. Дополнительно формируется таблица meta.tsv с метаданными (URL, источник, время загрузки, заголовок, длина текста), а при необходимости сохраняется и "сырой" HTML в сжатом виде.

Результаты

По результатам выгрузки корпуса получены следующие статистические характеристики:

- **Размер «сырых» данных:** 529 704 878 байт
- **Количество документов:** 58 866
- **Размер текста, выделенного из «сырых» данных:** 529 704 878 байт
- **Средний размер документа:** 8998,49 байт
- **Средний объём текста в документе:** около 9 КБ очищенного текстового содержимого

Выводы

В ходе первой лабораторной работы был выбран и проанализирован корпус документов из двух открытых источников — ACL Anthology и arXiv. Были изучены структура исходных документов, наличие метаинформации и существующие средства поиска по данным источникам. Документы сохранены в исходном формате и подготовлены для последующей обработки и использования в дальнейших лабораторных работах по информационному поиску.

Лабораторная работа №2 «Поисковый робот»

Необходимо написать парсер на любом языке программирования.

- 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.
- Сохранять в базе данных (например, MongoDB) документы со следующими полями:
  - url, нормализованный;
  - «сырой» html-текст документа;
  - название источника;
  - Дата обкачки документа в формате Unix time stamp.
- 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.

Описание

Целью второй лабораторной работы является разработка поискового робота для автоматической обкачки документов из открытых интернет-источников и сохранения их в базе данных для последующего использования в задачах информационного поиска. Поисковый робот реализует полный цикл загрузки документов: формирование очереди URL, выполнение HTTP-запросов, получение HTML-страниц и сохранение сырых данных вместе с метаинформацией. Управление параметрами работы робота осуществляется через YAML-конфигурационный файл, который передаётся программе в качестве единственного аргумента командной строки.

Журнал выполнения и решение проблем

В процессе выполнения лабораторной работы основное внимание было уделено обеспечению устойчивой и долговременной работы поискового робота. Для этого состояние обхода полностью хранится в базе данных: информация о URL, времени последней обкачки и запланированном времени следующей проверки сохраняется между запусками программы. Это позволяет при повторном запуске продолжить обработку документов, не начиная процесс заново.

Дополнительной задачей являлось предотвращение дублирования данных и избыточной переобкачки документов. Для её решения реализована логика повторной загрузки страниц только при изменении их содержимого. Были введены задержки между запросами и механизм временной блокировки URL при параллельной обработке.

Исходный код

Поисковый робот реализован на языке Python и использует стандартные средства работы с конфигурационными файлами и базами данных. В ходе реализованы компоненты нормализации URL, выполнения HTTP-запросов, сохранения сырых HTML-документов и метаинформации, а также логика управления очередью URL.

Выводы

В ходе выполнения второй лабораторной работы был разработан поисковый робот, обеспечивающий автоматическую обкачку документов, сохранение сырых HTML-данных и метаинформации в базе данных, а также возможность остановки и возобновления работы без потери прогресса. Реализованная система поддерживает переобкачку документов только при изменении их содержимого и удовлетворяет требованиям задания, создавая надёжную основу для формирования корпуса документов, используемого в последующих лабораторных работах по информационному поиску.

Лабораторная работа №3 «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?

Описание

Цель работы — реализовать токенизацию текстов документов корпуса, чтобы затем использовать полученные токены при построении индексов и поиске. Токенизация выполняется как последовательный проход по тексту документа с формированием токенов по заданным правилам, с приведением к единой форме и дополнительной нормализацией для русского языка.

Правила токенизации

В качестве токена рассматривается непрерывная последовательность символов, относящихся к «словным» или «числовым» (буквы латиницы/кириллицы и цифры). Разделителями считаются пробельные символы и знаки пунктуации. Перед добавлением токена выполняются нормализации: понижение регистра, приведение некоторых вариантов написания к одному виду (например, ё -> е). Для технических терминов допускаются составные формы: дефис внутри токена сохраняется. Токены короче минимального порога (например, 1 символ) отбрасываются.

Достоинства подхода: простая и быстрая реализация, линейная сложность по длине текста, устойчивая работа на больших объёмах данных, единая нормализация регистра и русских вариантов написания.

Недостатки: часть “шумных” токенов всё равно остаётся, а также возможны спорные случаи с апострофами, сокращениями и HTML/служебными фрагментами.

Примеры неудачных токенов и возможные улучшения

- Чистые числа и «нулевые» группы: 00, 000 и т.п. Улучшение: отбрасывать токены, состоящие только из цифр.
- Слова с апострофом и сокращения: 0’s и подобные формы. Улучшение: определить правило для апострофов.
- Составные технические идентификаторы: 000-document, 0-8b-instruct. Улучшение: оставить как есть или вводить отдельную нормализацию.
- Артефакты HTML/кодировок в тексте. Улучшение: усилить очистку HTML до токенизации.

Результаты и статистические данные

Токенизация была выполнена на полном корпусе из 58 866 документов. Получены следующие измерения:

- **docs = 58866** — сколько документов из списка было обработано токенизатором.
- **total_bytes = 529704878** — суммарный объём входного текста всех документов (в байтах).
- **token_count = 70584169** — общее число выделенных токенов по всему корпусу.
- **avg_token_len_chars = 6.00442** — средняя длина одного токена в символах.
- **time_sec = 6.38815** — время работы программы токенизации на всём корпусе (в секундах).
- **tokens_per_kb = 136.45** — плотность токенов: сколько токенов в среднем приходится на 1 килобайт входного текста.
- **stemming = 1** — стемминг был включён при токенизации.

Время выполнения, зависимость от объёма и скорость

Алгоритм токенизации выполняет один линейный проход по тексту каждого документа, поэтому время работы пропорционально общему объёму входных данных. Скорость составила порядка 81 000 KiB/сек, или 11 млн токенов/сек.

Такая скорость для однопроходной токенизации близка к практическому пределу. Потенциальные способы ускорения: чтение крупными буферами/mmap, уменьшение числа выделений памяти, упрощение проверок символов, распараллеливание по документам.

Выводы

В лабораторной работе реализована токенизация корпуса документов с едиными правилами выделения токенов и нормализацией. Получены статистические показатели, а также выявлены типичные примеры “шумных” токенов и направления для улучшения качества токенизации на смешанных HTML-данных.

Лабораторная работа №4 «Стемминг»

Добавить в созданную поисковую систему демматизацию. В простейшем случае, это просто поиск без учёта словоформ. В более сложном случае, можно давать бонус большего размера за точное совпадение слов.

Лемматизацию можно добавлять на этапе индексации, можно на этапе выполнения поискового запроса. В отчёте должна быть включена оценка качества поиска, после внедрения демматизации. 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?

Описание

Цель лабораторной работы — добавить в поисковую систему стемминг, чтобы поиск не зависел от словоформ и разные грамматические формы слова сопоставлялись как один терм.

Журнал выполнения и решение проблем

Стемминг был внедрён как дополнительный шаг после токенизации. Основная практическая проблема при внедрении заключалась в том, что стемминг может не только улучшать выдачу, но и ухудшать её. Это проявляется в случаях, когда разные слова сводятся к одному стему, из-за чего в результатах появляются нерелевантные документы.

Исходный код

В системе реализован режим включения стемминга через параметр запуска (например, `-stemming 0/1`). При построении индекса каждый токен перед добавлением в индекс проходит одинаковую цепочку преобразований, включая стемминг в соответствующем режиме.

Оценка качества поиска и анализ ухудшений

В большинстве случаев стемминг улучшает полноту: увеличивается количество релевантных документов. При этом обнаруживаются запросы, где качество ухудшается. Основные причины ухудшения связаны с “слипанием” разных слов в один стем.

Чтобы улучшить качество на таких запросах, не ухудшая остальные, можно использовать комбинированный подход: выполнять поиск по стемам, но дополнительно по-

вышать приоритет документов, где встречается точная форма слова из запроса.

Выводы

В лабораторной работе в поисковую систему был добавлен стемминг, применяемый согласованно при индексации и при обработке запросов. В результате поиск стал менее зависимым от словоформ, что повысило полноту выдачи. Для снижения негативного эффекта целесообразно ограничивать стемминг для отдельных классов токенов и учитывать точные совпадения при формировании выдачи.

Лабораторная работа №5 «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.

Описание

Целью лабораторной работы является проверка выполнения закона Ципфа на собранном корпусе. Для этого строится распределение терминов по частотам: термы сортируются по убыванию встречаемости, каждому терму присваивается ранг, после чего строится график зависимости частоты от ранга в логарифмической шкале.

Журнал выполнения и решение проблем

Для построения распределения по частотам сначала были посчитаны частоты терминов по всему корпусу после токенизации и нормализации. Основной сложностью является наличие “шумных” терминов, которые заметно влияют на хвост распределения и усиливают расхождение с идеальным законом.

Результаты и объяснение расхождения с законом Ципфа

График распределения терминов по частотам в логарифмической шкале демонстрирует характерную для текстов картину. Расхождения с идеальным законом Ципфа объясняются следующими факторами: корпус является смесью разных источников и стилей текста; в данных присутствуют шумовые токены; стемминг и правила токенизации объединяют часть словоформ и изменяют частоты некоторых групп терминов.

ТОП-10 терминов (term-count):

1. the 1914451
2. and 1654312
3. type 1465962
4. namepart 1341016

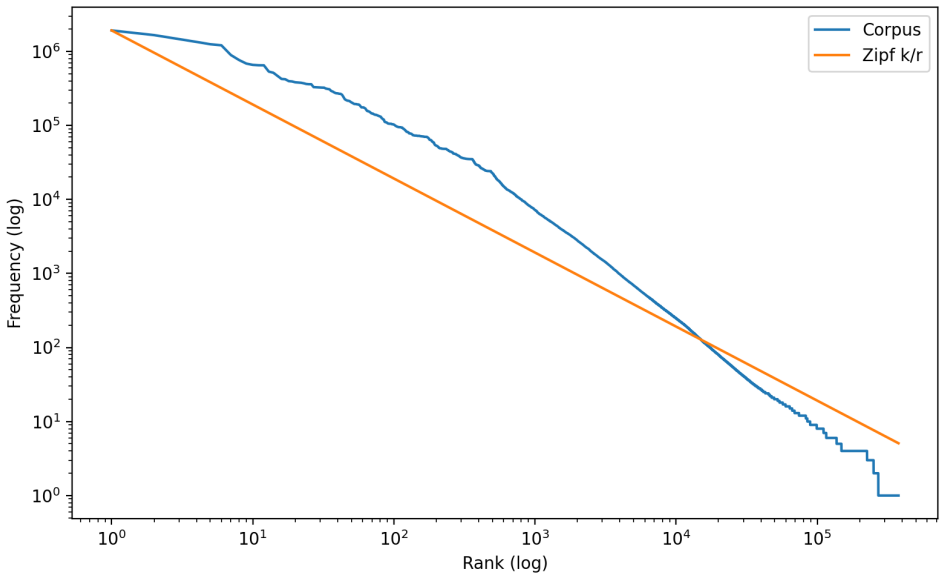

Рис. 1: Распределение терминов по частотам в логарифмической шкале с наложенным законом Ципфа

5. to 1245211
6. of 1204498
7. for 891166
8. is 759947
9. in 683674
10. role 656863

Выводы

В ходе лабораторной работы было построено распределение терминов по частотам в логарифмической шкале и выполнено сравнение с законом Ципфа. Распределение в целом соответствует ожидаемой форме, а основные расхождения объясняются неоднородностью корпуса, наличием шумовых токенов и влиянием предобработки текста.

Лабораторная работа №7 «Обратный индекс»

Описание

Целью работы является построение обратного индекса, пригодного для булева поиска, по подготовленному корпусу документов. Индекс строится в собственном бинарном формате и предназначен для дальнейшего расширения в следующих лабораторных работах. Помимо обратного индекса формируется прямой индекс, содержащий ссылки на документы и их заголовки.

Журнал выполнения и решение проблем

При построении индекса основная сложность заключалась в ограничениях на структуры данных: нельзя использовать ассоциативные контейнеры. Для решения использована внешняя сортировка: из документов формируются пары “терм-doc_id”, далее они сортируются по частям и сливаются в итоговый индекс.

Исходный код

Индексатор реализован на C++ и формирует три файла: прямой индекс docs.bin, словарь terms.bin и постинги postings.bin. Для построения используется внешняя сортировка: данные разбиваются на несколько “run”-файлов, каждый run сортируется, затем выполняется k-way merge.

Выводы

В ходе работы построен булев индекс по корпусу из 58866 документов. В процессе внешней сортировки было создано 10 промежуточных run-файлов, итоговое число уникальных термов составило 376586, а размер файла постингов — 79223688 байт. Полученный индекс соответствует требованиям.

Лабораторная работа №8 «Булев поиск»

Описание

Целью работы является реализация булевого поиска по построенному обратному индексу. Поисковая система принимает запрос с логическими операторами и возвращает документы, удовлетворяющие булевому выражению.

Журнал выполнения и решение проблем

Основной задачей было обеспечить корректную обработку логических операторов и скобок, а также согласованность обработки запроса и индекса. В ходе тестирования было учтено, что в оболочке bash символ ! интерпретируется как служебный, поэтому запросы с отрицанием корректно запускать в кавычках или экранировать !.

Исходный код

Поисковый модуль реализован как CLI-утилита, которая загружает бинарные файлы индекса, разбирает запрос в булево выражение и вычисляет результат через операции над отсортированными списками doc_id. Для AND выполняется пересечение списков, для OR — объединение, для NOT — разность с универсальным множеством документов.

Примеры работы программы

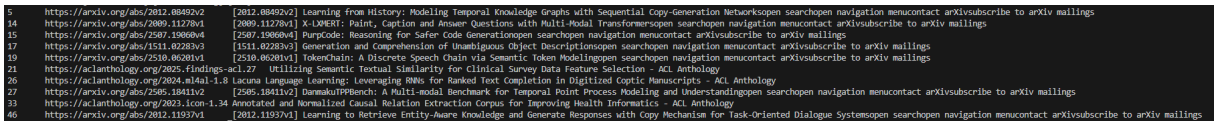

Рис. 2: Запрос: make search Q='machine AND learning'

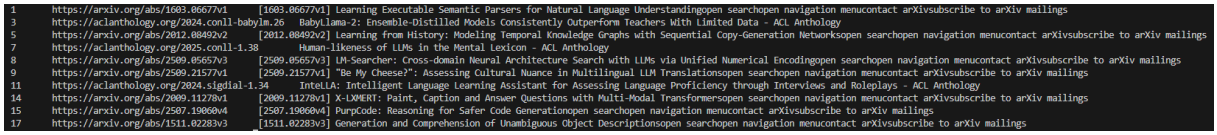

Рис. 3: Запрос: make search Q='machine | learning'

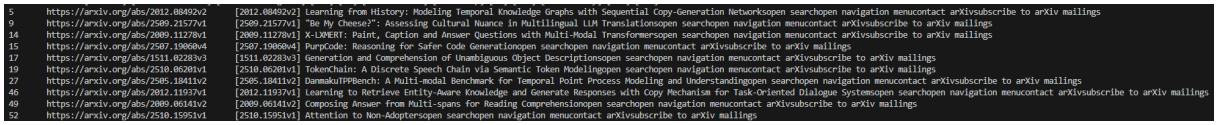

Рис. 4: Запрос: make search Q=’(ai OR artificial intelligence) AND (machine OR deep) AND ! survey’

Выводы

Реализован булев поиск по собственному бинарному индексу. Поддерживаются операторы AND/OR/NOT (а также их символьные формы & | !) и скобки, выдача формируется через прямой индекс и содержит ссылки и заголовки документов. Реализация корректно работает на полном корпусе и является базой для последующих улучшений поиска и ранжирования.