Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Информационный поиск»

Студент: В. М. Филиппов
Преподаватель: А. А. Кухтичев
Группа: М8О-410Б
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №4 «Булев индекс»

Необходимо разработать на языке C++ программу, которая индексирует корпус документов. Индекс корпуса необходимо хранить на диске в бинарном формате.

1 Описание

В рамках этой лабораторной работы я реализовал булев индекс, а также вспомогательную хэш таблицу, которая используется при индексации.

1 Булев индекс

Перед нами стоит задача поиска документов, в которых встречаются слова из запросов. В самом простом случае это можно реализовать так: для каждого документа составляем список токенов, которые в нём встречаются. Далее при поиске проходим по каждому документу, и если встречаем слово из запроса, то добавляем этот документ в выдачу. Такой способ индексации называется "прямым". Но этот способ хранения требует от нас хранить все токены всех документов, что может занимать очень много места. Также сам поиск будет медленным, поскольку при каждом запросе мы проходим по абсолютно всем документам, пытаясь найти нужный токен.

Более оптимизированным способом хранения является "обратный индекс". Его суть заключается в том, чтобы для каждого токена хранить список документов, в котором этот токен встречается. Это будет экономнее по памяти, а поиск будет быстрее, поскольку мы можем применять бинарный поиск по токенам или хэш-таблицы. Второе преимущество обратного индекса в том, что нам не нужно «листать» содержимое самих документов. Мы сразу получаем готовый список ID документов и работаем только с ним.

Булев индекс — это вариация инвертированного индекса, которая фиксирует лишь факт присутствия токена в документе без учета его веса или позиции. Такая структура позволяет выполнять поисковые запросы, используя операции булевой алгебры (И, ИЛИ, НЕ), путем пересечения или объединения списков документов.

2 Хэш-таблица

Поскольку по требованиям к лабораторным работам, использовать STL хэш-таблицы нельзя, я написал свою несложную реализацию. Для разрешения коллизий используется метод цепочек на основе односвязного списка. При возникновении коллизии (когда разные ключи имеют один и тот же хэш) новые элементы добавляются в начало списка соответствующего бакета. В качестве основы используется стандартный функтор std::hash<std::string>, результат которого приводится к размеру внутреннего массива бакетов с помощью операции взятия остатка по модулю.

3 Способ хранения на диске

Файл состоит из следующих блоков:

1. **Заголовок**, который содержит магическое число (Magic Number) для верификации формата, версию (используется для переключения между сжатым и несжатым режимами), а также общее количество проиндексированных документов и уникальных токенов.

2. **Таблица URL**, содержащая список исходных адресов документов в формате (length, string).

3. **Словарь токенов**, то есть набор структур фиксированного размера TermEntry. Каждая запись содержит:

   (a) 64-битный хэш токена;

   (b) Абсолютное смещение к строковому представлению токена;

   (c) Абсолютное смещение к списку вхождений;

   (d) Количество документов, содержащих токен.

   Благодаря фиксированному размеру TermEntry и предварительной сортировке записей по значению хэша, поиск конкретного слова в индексе осуществляется за O(logN) с помощью бинарного поиска.

4. **Блок строковых данных**, в котором хранятся сами токены, на которые ссылается словарь.

5. **Блок списков вхождений**, содержащий пары пары (doc_id, term_frequency). Использования term_frequency понадобится в будущем, когда появится необходимость в использовании TF-IDF. Пока там нули.

2 Исходный код

При реализации был написан интерфейс IIndexSource. Далее имеются два его наследника: RamIndexSource и MappedIndexSource, с помощью которых реализуется хранение индекса как в оперативной памяти, так и на диске при помощи маппинга. Внутри RamIndexSource используется HashMap в качестве хранилища, а внутри MappedIndexSource заммапленный файл на диске. уТакже имеется интерфейс IIndexator, который реализует конкретный метод индексации (булев и в последствии TF-IDF). Основной метод этого класса - это addDocument, который добавляет документ в индекс.

Кроме того, я решил удалить из индекса токены, которые встречаются один раз в одном документе. Это почти гарантированные опечатки.

Корпус размером в 305135 документов был проиндексирован за 466.515 секунды со скоростью 5842.03 КБ/с. Дампинг индекса занял 12.5987 секунд. Индекс получился размером в 1190578819 байт (1.19 ГБ)

3 Выводы

За время выполнения этой лабораторной работы я узнал о том, что такое булев индекс, а также поработал с отображением файла в память. Узнал, как сделать бинарный поиск прямо по данным, которые лежат на диске, создав специальный формат файлов для этого.

Список литературы

[1] Маннинг, К. Введение в информационный поиск / К. Маннинг, П. Рагхаван, Х. Шютце ; пер. с англ. Д. А. Клюшина. — М. : Вильямс, 2011. — 528 с. — ISBN 978-5-8459-1623-4.

[2] Алгоритмы поиска, обратный индекс — Часть 1 // Хабр : [сайт]. — 2011. — URL: https://habr.com/ru/articles/53987/ (дата обращения: 17.12.2025).