Document Search Engine

Session: 2021 – 2025

Submitted by:

Muhammad Usman Asghar 2021-CS-46

Submitted to:

Prof. Dr. Syed Khaldoon Khurshid

Department of Computer Science

University of Engineering and Technology

Lahore

Abstract

This report details the design and implementation of a simple document search engine created for educational and demonstration purposes. The search engine is designed to allow users to search documents by title or content using an index-based approach. Python was selected for its simplicity and strong text processing capabilities. This report discusses the methodology, structure, optimization strategies, and testing practices involved in the development of the search engine.

Introduction

Efficient document retrieval is a key aspect of information retrieval (IR) systems. Linear search methods are not scalable for large datasets, making the use of indexes essential. This project explores the construction of an index that supports quick lookups and updates, forming the basis of a functional search engine. The implementation demonstrates how foundational indexing and retrieval mechanisms work, providing a stepping stone for more complex IR systems.

Objectives

1. **Develop a Functional Search Engine**: Create a search engine capable of retrieving documents by title and searching within document content.
2. **Indexing**: Implement an index structure for fast lookups and efficient updates.
3. **Performance Optimization**: Ensure the system performs efficiently, even when scaling up.
4. **Comprehensive Testing**: Verify the accuracy and robustness of the search engine.
5. **Documentation**: Provide clear documentation to facilitate understanding and future enhancements.

Methodology

Step 1: Designing the Index Structure

The search engine is built using three primary data structures:

- **Index**: A dictionary where each word is a key, and the value is a list of document identifiers where the word appears.
- **Title Index**: A dictionary mapping document titles to their corresponding filenames.
- **Document Storage**: A dictionary storing the content of each document, enabling quick retrieval when needed.

Step 2: Index-Building Process

The index is created through the following steps:

- **Document Loading**: A function retrieves the list of document filenames to be indexed.
- **Content Reading**: The content of each document is read and stored.
- **Tokenization**: Text is parsed to extract individual words, filtering out punctuation and converting text to lowercase to ensure consistency.

- **Index Construction**: Each word from the documents is added to the index along with the document identifier. Repeated words in a single document are recorded only once for space efficiency.

Step 3: Search Operations

1. **Title-Based Search**: A function allows retrieval of a document by its title. If a matching title is found in the title index, the document content is returned.
2. **Content-Based Search**: A function processes the user's query, tokenizes the input, and looks up each word in the index. The function identifies and returns a list of documents containing the query words.

Step 4: User Interface

A simple command-line interface (CLI) was developed to interact with the search engine:

- **Search by Title**: Users can enter a document title to retrieve the document's content.
- **Search by Content**: Users can input search terms, and the search engine returns a list of documents containing those terms.
- **Exit**: An option to close the search engine.

The interface ensures user-friendly interaction by presenting clear options and displaying results in a readable format.

Step 5: Optimization Strategies

Optimization was considered throughout development:

- **Efficient Data Structures**: Using dictionaries allows for average-case O(1) time complexity for lookup operations.
- **Tokenization Refinement**: The tokenizer ensures that only relevant alphanumeric characters are considered, improving parsing accuracy.
- **Index Compression**: Duplicate entries for the same document within the index are avoided, conserving memory.

Step 6: Testing and Validation

Testing was conducted to ensure the search engine's reliability:

- **Functional Tests**: Various searches were performed to verify correct operation, including searches with common, rare, and non-existent terms.
- **Edge Cases**: Tests included scenarios with repeated words, words with special characters, and empty searches.
- **Performance Assessment**: The response time was measured for different document volumes to confirm that the search engine scales reasonably with data size.

Results

The search engine met the project's baseline requirements:

- **Title-Based Search**: Accurately retrieves documents when provided with the correct title.
- **Content-Based Search**: Efficiently returns documents containing the queried words.
- **Performance**: Demonstrated fast search response for small to medium-sized document collections.

Future Enhancements

To build on this foundation, the following features could be added:

1. **Phrase Search Support**: Implement search capabilities for exact phrases.
2. **Ranking Mechanism**: Introduce a ranking algorithm to prioritize results based on relevance.
3. **Stemming and Synonym Expansion**: Include stemming algorithms to group words by their root and handle synonyms.
4. **GUI Interface**: Create a graphical interface for an improved user experience.
5. **Advanced Parsing**: Use natural language processing (NLP) to better handle complex queries and differentiate between word meanings.

Conclusion

This project demonstrated the principles of building a simple, yet functional, document search engine. By focusing on the basics of indexing and retrieval, the project lays the groundwork for more sophisticated search engines. The process of indexing, searching, and optimization provided valuable insights into real-world IR systems, showcasing the importance of efficient data structures and algorithmic precision.