King Saud University
College of Computer and Information Sciences

CSC 212 Project - Fall 2024

جامعة الملك سعود
King Saud University
كلية علوم الحاسب والمعلومات
قسم تقنية المعلومات

A Simple Search Engine

Names	ID
Jood Alkhrashi	444203007
Najla Almazyad	444200948

Introduction

This project involves the development of a search engine that indexes documents and efficiently retrieves relevant results for a given query. The implementation relies on multiple data structures such as linked lists and binary search trees (BST) to demonstrate and compare the performance of different indexing and retrieval methods.

Implementation Details

1. Document Processing

Documents are read from a CSV file, tokenized into words, converted to lowercase, and cleaned of special characters. The words are then added to a Binary Search Tree (BST) for indexing.

2. Data Structures

The following custom data structures are used in the implementation:

- Binary Search Tree (BST): Efficiently stores and retrieves words and their associated documents.

- LinkedList: Maintains lists of document IDs for each word.

- Node: Represents individual elements in the LinkedList and BST.

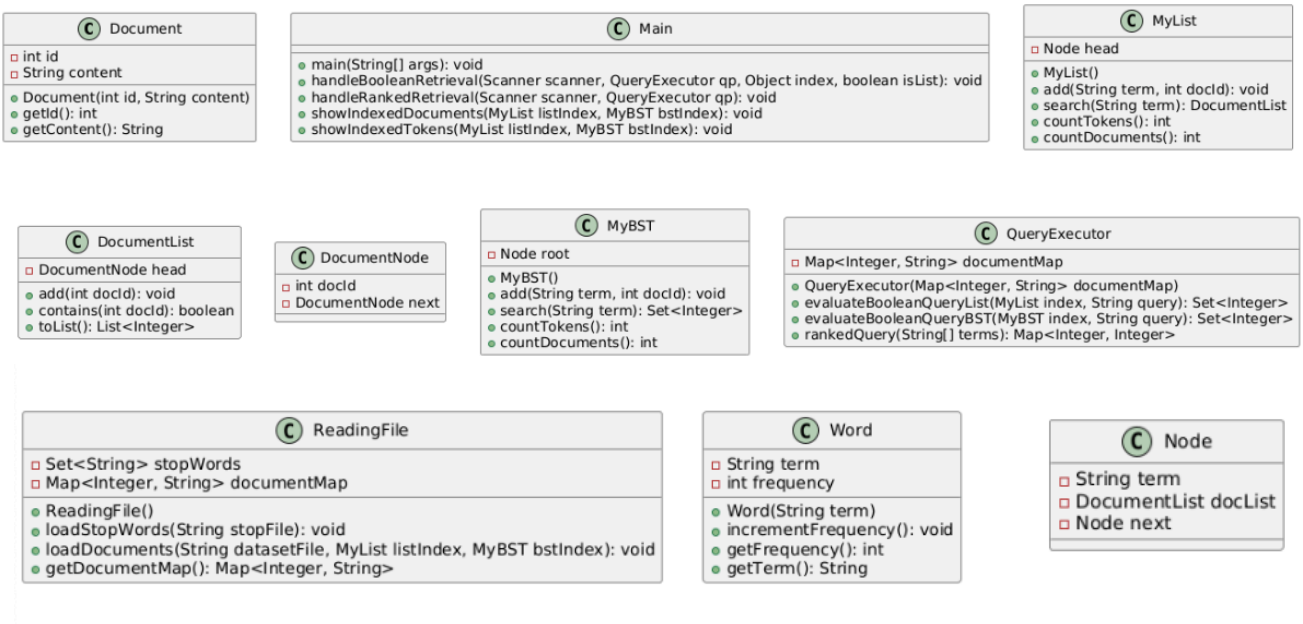
3. Query Processing

Boolean queries (AND/OR) are supported. For AND queries, the search engine finds documents containing both terms by intersecting their document lists. For OR queries, it returns documents containing at least one of the terms by merging their document lists.

4. User Interaction

The program provides an interactive menu allowing users to perform queries and view results. Input is validated to ensure queries contain exactly two terms. Results are displayed in the console.

Class diagram (UML)



Performance Analysis

The project includes three different indexing approaches: a basic index, an inverted index, and an inverted index enhanced with a Binary Search Tree (BST). Below is the performance analysis comparing these methods:

1. Basic Index:

This approach maps document IDs to words, which is not efficient for query processing. Searching for a specific term requires traversing all documents, leading to a linear time complexity of O(n), where n is the number of documents. This makes the basic index slow for large datasets.(least efficiency)

2. Inverted Index (using Lists):

The inverted index improves search efficiency by mapping terms to a list of document IDs containing those terms. This reduces the search time to O(d), where d is the number of documents containing the specific term. However, the use of lists for storing document IDs can make lookups and updates inefficient for very large datasets.

3. Inverted Index with BST:

The BST enhances the inverted index by storing document lists in a more efficient manner. This allows for logarithmic time complexity (O(log d)) for searching, insertion, and deletion of document IDs. The BST provides faster query processing compared to lists, especially for larger collections of documents.(best efficiency)

Conclusion

This implementation of a simple search engine demonstrates the use of custom data structures to process documents and handle queries efficiently. The system validates input effectively, ensuring accurate results for Boolean queries, and provides a solid foundation for further enhancements, such as incorporating ranked retrieval methods. Additionally, GitHub was utilized for collaborative development, enabling seamless teamwork, version control, and code sharing throughout the project.