جامعة الملك سعود
King Saud University

كلية علوم الحاسب والمعلومات

CSC 212 Project - Fall 2024

A Simple Search Engine

Due date: 21/11/2024

10 marks

Project Overview

This project aims to develop a basic search engine capable of indexing, retrieving, and ranking documents based on given queries. The core data structures utilized will be lists and inverted indices. The engine will support simple Boolean queries (AND, OR) and incorporate term frequency for relevance ranking. The focus is on using ADT List to build the index and inverted index for building the search engine. Finally, use Binary Search Trees (BSTs) to enhance the inverted index performance.

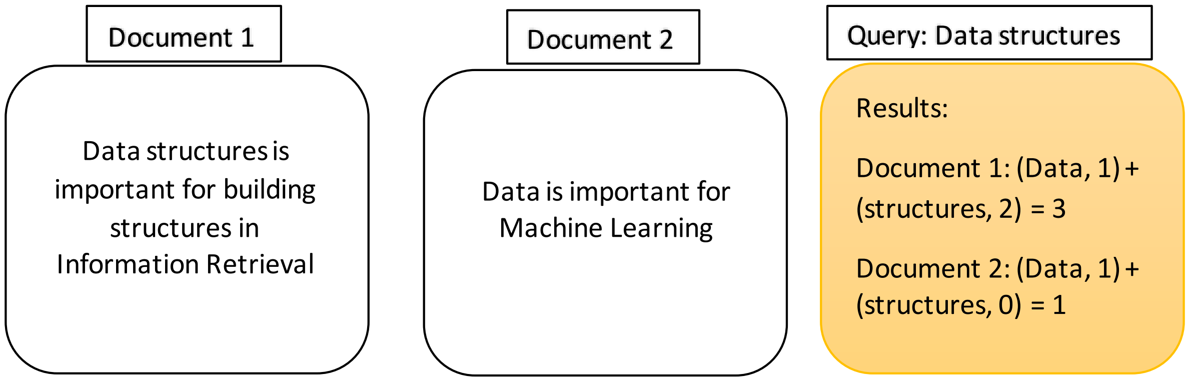
Data Structures and Algorithms

- **Index:** A mapping from document IDs to a list of words contained in the document.
- **Inverted Index:** A mapping from terms (unique words) to a list of documents containing those terms. Both of Index and Inverted Index will be implemented using list of lists.
- **Inverted Index with BSTs:** Enhance the implementation of Inverted Index by using BSTs instead of Lists.
- **Lists:** Lists will be used to represent the document collection, the vocabulary, and the lists of documents associated with each term in the inverted index.
- **BSTs:** Trees will be used to enhance the search for an item in the collection making it look up items in logarithmic time.

Problem Statements

1. **Document Processing:**
    - Read documents from a CSV file.
    - Split the text into a list of words based on whitespace.
    - Convert all text to lowercase
    - Preprocess the text by removing stop-words (e.g., "the," "is," "and"). The list of stop-words will be given to you.
    - Remove all punctuations and non-alphanumerical characters
    - Proceed to build the index
2. **Indexing:**
    - Create an index by mapping the documents IDs to list of words
    - Create the Inverted Index by mapping the words to list of Documents IDs
    - Create the BST Inverted Index which will enhance the search for items whether in terms or documents IDs.

3. **Query Processing:**
   - Support simple Boolean queries (AND, OR).
   - For AND queries, you need to find all documents that only contain both words using intersection of lists, and add the results in the result list of documents
   - For OR queries, you need to find all documents that contain either both words or one of them and add the results in the result list of documents
   - Both AND and OR will produce a set of documents to be the result. **Note that:** these documents will not be ranked or ordered, which is called Boolean Retrieval.
   - After implementing the Query Processor to return the results. Analyze the performance of the Boolean operation on all the required indices separately and show your analysis on which one is more efficient using Performance Analysis. The analysis should be included in the report.
4. **Ranking:**
   - Unlike the Boolean Retrieval where there are an unordered set of results for a given query, Ranked Retrieval orders the results of documents in response to a query without using "AND" and "OR". For example, each document will have a score for a given query and will be ranked accordingly. The first document will have the highest score for that query in an descending order. To calculate the score, use the Term Frequency for each term in the document and then, sum the score for all query terms for that document. Below is an example, where there are two documents and a query "Data structures" and a ranking of the documents is produced.


Document 1
Data structures is important for building structures in Information Retrieval
Document 2
Data is important for Machine Learning
Query: Data structures
Results:
Document 1: (Data, 1) + (structures, 2) = 3
Document 2: (Data, 1) + (structures, 0) = 1

   - Implement term frequency (TF) to rank documents based on the frequency of query terms within them.
   - The querying will contain several words, and you need to calculate a score for each document by summing the term frequency of each query word on that document.

Understanding Indexes

An Index is a data structure of mapping the documents to words, but this is not an efficient way of building search engines. A better way is to use the inverted index. An inverted index is a data structure that maps terms (unique words) to a list of documents containing those terms. It's a fundamental component of many search engines and information retrieval systems.

How it Works

1. **Tokenization:** Documents are broken down into individual words or tokens.
2. **Indexing:** Each word is associated with a list of documents that contain it.
3. **Inverted Index Creation:** A data structure is created to map words to their corresponding lists of documents.

Example for Boolean Retrieval using Inverted Index

Consider these four documents:

- Document 1: "The national flag of the Kingdom of Saudi Arabia color is green and white"
- Document 2: "The green color extends from the pole to the end of the flag"
- Document 3: "The bright green color known as emerald green"
- Document 4: "The flag of Saudi Arabia has an Arabic shahada and a sword in snow white"

The inverted index would look something like this:

Word	Documents	Word	Documents	Word	Documents
National	[1]	**Green**	[1,2,3]	**known**	[3]
Flag	[1,2,4]	**White**	[1,4]	**emerald**	[3]
Kingdom	[1]	**Extends**	[2]	**Arabic**	[4]
Saudi	[1,4]	**Pole**	[2]	**shahada**	[4]
Arabia	[1,4]	**End**	[2]	**sword**	[4]
color	[1,2,3]	**Bright**	[3]	**snow**	[4]

Input/Output Example

Input: Search query: "color AND green AND white"

Output:

- **Document 1:** "The national flag of the Kingdom of Saudi Arabia color is green and white"

This document contains all three terms, so it is the only one returned as a result.

Input: Search for query: “green OR shahada”

Output:

- **Document 1:** "The national flag of the Kingdom of Saudi Arabia color is green and white"
- **Document 2:** "The green color extends from the pole to the end of the flag"
- **Document 3:** "The bright green color known as emerald green"
- **Document 4:** "The flag of Saudi Arabia has an Arabic shahada and a sword in snow white"

Document 1, 2 and 3 contains “green” while document 4 contains “shahada” so a concatenation between them is done to produce the final list of results.

Advantages of Inverted Indexes

- **Efficient Searching:** Quickly find documents containing specific terms.
- **Relevance Ranking:** Support ranking of search results based on relevance.
- **Boolean Queries:** Handle complex search queries involving Boolean operators.
- **Scalability:** Can handle large collections of documents.

Deliverables

1. **Index:** A structure to map documents IDs to words using ADT List. A basic approach that is not effective for query processing.
2. **Inverted Index:** A well-structured and efficient implementation of the index to enhance the performance of answering queries. This should be implemented using ADT List.
3. **Inverted Index with BST:** to enhance the performance of query processing, use Binary Search Trees to redo the Inverted Index.
4. **Analysis:** compare the performance of Boolean Retrieval between Index and Inverted Index, Inverted Index and BST. Also, show why a given index is better than others in Big-Oh. This should be included in the documentation.
5. **Query Processor:** A module capable of processing simple Boolean queries and returning relevant results. Also, the choice of doing Ranked Retrieval.
6. **Ranking Algorithm:** Implementation of term frequency for ranking documents.
7. **User Interface:** A basic interface for users to input queries and view search results.
8. **Documentation:** Clear and concise documentation explaining the design, implementation, and usage of the search engine.
9. **Class Diagram:** A Clear design for your project from classes or interfaces to methods to be included in your documentation.
10. **A test Menu:** The menu should show choices for the following:

    - **Boolean Retrieval:** to enter a Boolean query that will return a set of unranked documents
    - **Ranked Retrieval:** to enter a query that will return a ranked list of documents with their scores
    - **Indexed Documents:** to show number of documents in the index
    - **Indexed Tokens:** to show number of vocabulary and tokens in the index

Additional Considerations

- **Efficiency:** Consider the efficiency of your data structures and algorithms, especially for large datasets.
- **Stop Word List:** use the given stopwords list to remove them as they don't add information to the ranking, and it should focus on topical words.
- **Lowercase all letters:** make all letters lower case to make sure to count all occurrences.
- **Boolean Ranking:** this will not rank the resulting document but will output the set of documents to be an answer without and ranking.
- **Relevance Ranking:** Consider using term frequency in the inverted index and how to store it. This should be done in a way that doesn't delay the query processing.
- **Index:** You should have one index for both Boolean Retrieval and Ranked Retrieval. Storing the frequencies in an efficient manner is required.

By addressing these points, you can create a robust and effective search engine that demonstrates your understanding of data structures and algorithms.

Search Process

1. **Query Parsing:** The search query is broken down into individual terms.
2. **Term Lookup:** Each term is looked up in the inverted index (one of the 3 choices for Inverted Index).
3. **Document Retrieval:** The lists of documents associated with each term are retrieved.
4. **Boolean Operation:** If the query involves Boolean operators (AND, OR, NOT), the lists are combined accordingly.
5. **Ranking:** The retrieved documents are often ranked based on relevance factors like term frequency.

Bonus:

- Implement more efficient data structure than BSTs.
- Use GitHub private repo to show you collaborated on the project.

Rules:

- All data structures used in this assignment must be implemented by the student. The use of Java collections or any other data structures library is strictly forbidden.
- Posting the code of the assignment or a link to it on public servers, social platforms or any communication media including but not limited to Facebook, Twitter or WhatsApp will result in disciplinary measures against any involved parties.
- All submitted code will be automatically checked for similarity, and if plagiarism is confirmed penalties will apply.
- You may be selected to discuss your code with an examiner at the discretion of the teaching team. If the examiner concludes plagiarism has taken place, penalties will apply.
- You are allowed up to 3 team members per project.