King Saud University

College of Computer and Information Sciences

CSC212: Data Structure

1st Semester 1446 H

A Simple Search Engine

Table of Contents

Explanation of our search engine

Overview

In this project, we developed a simple search engine capable of indexing, retrieving, and ranking documents based on input queries. We used fundamental data structures such as lists and inverted indices. The engine supports simple Boolean queries (AND, OR), as well as term frequency (TF) for ranking documents by relevance. The index and inverted index were built using lists to create the search engine.

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.

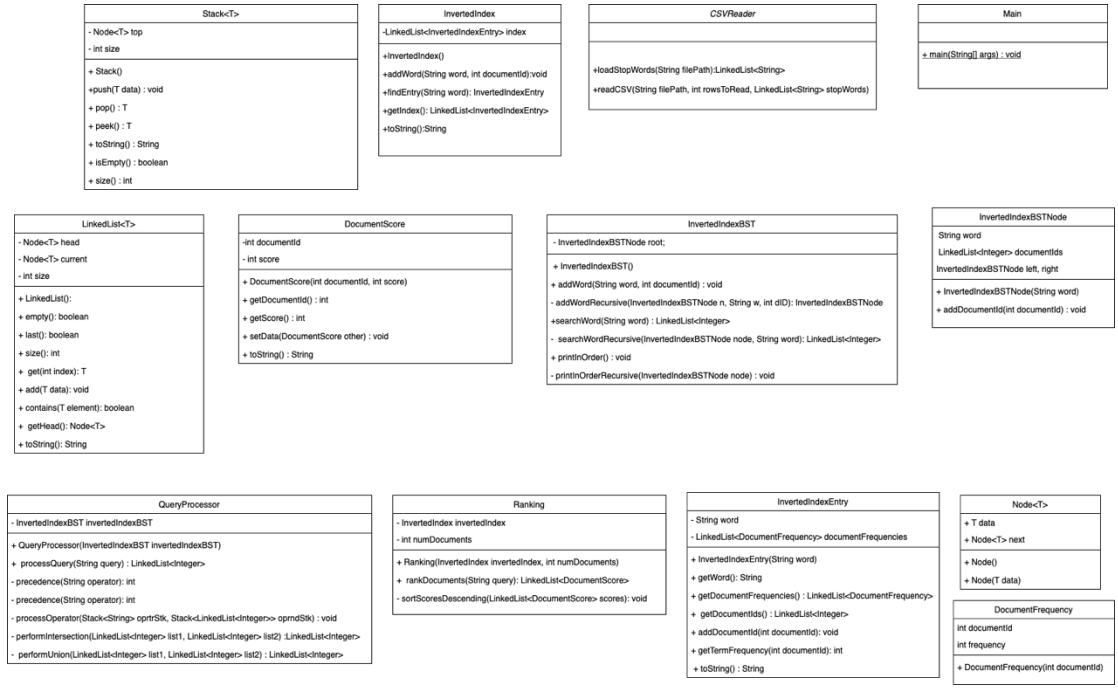
• **Lists**: Lists will be used to represent the document collection, the vocabulary, and the lists of documents associated with each term in the inverted index.

• **BSTs**: Trees will be used to enhance the search for an item in the collection making it look up items in logarithmic time

search engine

A search engine is a software tool or online platform that aims to help users find information on the Internet. It works by indexing vast amounts of content available on websites.

Class diagram showing your classes and methods



Classes

LinkedList:It allows you to store a sequence of elements where each element points to the next. This class provides functionality for adding elements, accessing elements by index, checking if an element exists in the list.

CSVReader:is designed to read and process data from CSV files. It uses a custom LinkedList implementation (LinkedList) to store and organize the data. This class has two main methods: one for loading stop words and another for reading and processing rows from a CSV file.

InvertedIndex: It constructs an inverted index, a data structure used in information retrieval systems to associate words with the documents they contain. Each unique word is stored with a list of document IDs that contain that word.

InvertedIndexEntry: This class represents a single entry in the **inverted index**. It maps a word to a collection of document IDs and their respective term frequencies.

Document frequency: A helper class used by InvertedIndexEntry to store a document ID and the frequency of the associated word in that document.

InvertedIndexBST: This class represents an inverted index using a Binary Search Tree (BST). It is used to store words that appear in a set of documents, along with the document IDs that contain each word.

InvertedIndexBSTNode: which represents a node in a Binary Search Tree (BST).

Node: is a generic class that represents a single node in a data structure.

QueryProcessor: is processes text-based search queries that include search terms and logical operators (like "AND" or "OR"). It searches for document containing, those words and then applies logical operations between the lists of document IDs.

Ranking: Is calculates a relevance score for each document based on how frequently the terms in the query appear in the document.

Stack: represents a stack data structure implemented using a linked list. It provides the fundamental stack operations such as push, pop...

Main: The class that has the main method and has the menu code that prompts user to choose an operation to perform from one of the 6: 1- Retrieve a term by one of the three options: Using index with lists ,Using inverted index with lists or Using inverted index with BST, 2- Boolean retrieval (AND/OR) , 3- ranked retrival, 4- indexed documents, 5- indexed tokens, 6- exit

Performance analysis

Time Complexity Index retrieval (using list of list)

Statements	S/E	Frequency	Total
System.out.println("Documents containing '" + term + "':");	1	1	1
for (int docId = 0; docId < documents.size(); docId++) {	1	n+1	n+1
LinkedList<String> document = documents.get(docId);	1	n	n
if (document.contains(term)) {	1	n*m	n*m
System.out.println("Document " + docId);	1	n*m	n*m
}	0	-	0
}	0	-	0
		Big-O complexity is:	O(n)

Time Complexity Inverted index retrieval (using list of list)

Statements	S/E	Frequency	Total
System.out.println("Documents containing '" + term + "' (using inverted index):");	1	1	1
InvertedIndexEntry entry = invertedIndex.findEntry(term);	1	1	1
if (entry != null) {	1	1	1
System.out.println("Document IDs and frequencies:");	1	1	1
LinkedList<DocumentFrequency> docFreqs = entry.getDocumentFrequencies();	1	1	1
for (int i = 0; i < docFreqs.size(); i++) {	1	n+1	n+1
DocumentFrequency df = docFreqs.get(i);	1	n	n
System.out.println("Doc " + df.documentId + ": frequency = " + df.frequency);	1	n	n
}	0	-	0
}	0	-	0
else {	1	1	1
System.out.println("Term not found");	1	1	1
}	0	-	0
		Big-O complexity is:	O(n)

Time Complexity Inverted index retrieval (using BST)

Statements	S/E	Frequency	Total
System.out.println("Documents containing '" + term + "' (using BST):");	1	1	1
LinkedList<Integer> bstResults = invertedIndexBST.searchWord(term);	1	1	log n
if (bstResults != null && bstResults.size() > 0) {	1	1	1
System.out.println("Document IDs: " + bstResults);	1	n	n
} else {	1	1	1
System.out.println("Term not found");	1	1	1
}	0	-	0
	Big-O complexity is:		Best O(log n) worst O(n)

Time Complexity Boolean Retrieval

*O(k*n)*

K is the number of terms in the Boolean query
N is the number of documents that are processed for each term in the query

Comparison Table

Index retrieval (using list of list)	Inverted index retrieval (using list of list)	Inverted index retrieval (using BST)
O(n)	O(n)	O(logn)
		The Best

Sample Run

```
run:

--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 2

Boolean Retrieval (Note: Stop words are excluded)
Enter Boolean query (e.g., 'term1 AND term2' or 'term1 OR term2'): weather OR warming
Matching documents: [2, 14, 18, 30, 34, 38, 46]

--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 2

Boolean Retrieval (Note: Stop words are excluded)
Enter Boolean query (e.g., 'term1 AND term2' or 'term1 OR term2'): market OR sports
Matching documents: [0, 9, 12, 17, 21, 25, 29, 33, 34, 41, 45, 49]

--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 2

Boolean Retrieval (Note: Stop words are excluded)
Enter Boolean query (e.g., 'term1 AND term2' or 'term1 OR term2'): market OR sports AND warming
Matching documents: [0, 12, 34, 41]
```

```
--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 3

Ranked Retrieval
Enter query for ranking: market sports

Ranked results
(DocID  Score):
33      2
41      2
0       1
9       1
12      1
17      1
21      1
25      1
29      1
34      1
45      1
49      1

--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 2

Boolean Retrieval (Note: Stop words are excluded)
Enter Boolean query (e.g., 'term1 AND term2' or 'term1 OR term2'): weather warming
Matching documents: [34]
```

```
--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 3

Ranked Retrieval
Enter query for ranking: weather warming

Ranked results
(DocID  Score):
34      2
2       1
14      1
18      1
30      1
38      1
46      1

--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 3

Ranked Retrieval
Enter query for ranking: business world market

Ranked results
(DocID  Score):
8       2
0       1
12      1
21      1
41      1
44      1
```

```
--- Search Engine System ---
1. Retrieve a term
2. Boolean retrieval (AND/OR)
3. Ranked retrieval
4. Indexed documents
5. Indexed tokens
6. Exit
Enter your choice: 6
Exiting...
```