Building an Information Retrieval System using Apache Lucene

November 29, 2019

Group Name: ***Dracarys***

Members: ***Nandish Bandi Subbarayappa, Shivam Singh, Abhishek Gopalrao, Danish Murad, Prateek Kumar Nayak***

Purpose:

The assignment is to develop an information Retrieval system using Apache Lucene of version (minimum 3.6) with JAVA programming language. Apache Lucene version 8.3.0 is used here with JAVA version 13.0.1. The flow and functionality of the program is explained in this document. Okapi BM25 is the ranking model used to score the relevance of each document based on user's query. The output retrieves top 10 relevant documents which includes parameters like filename, title, rank, and score respectively. HTML files are also included in which case along with the above parameters, an extra Path parameter is also retrieved.

CONTENTS

Building an Information Retrieval System using Apache Lucene

1. Software Requirements

In order to get started with building this IR system, we first need to have some software installed on our local machine. We are running this entire project on an IDE – IntelliJ Community Edition. It has the latest Java Development Kit updated in it (13.0.1) which basically supports all the JAVA related libraries. Apache Lucene libraries are imported as well to help support the required tasks each one of them performs.

2. a) What is Apache Lucene?

Apache Lucene is basically a search engine which helps in retrieving results for a user's query. It includes the processes of Indexing and Searching. Based on user's query, Indexing is carried out which analyzes the collection of documents being considered. Depending on the search, either a new index will be created or an existing one will be updated. This process of indexing helps in retrieving the expected keys instead of the entire document – basically increasing the search speed. This is then followed by the Searching process which is carried out on the documents that are indexed where the searching process is redirected to the instances where the directory is created.

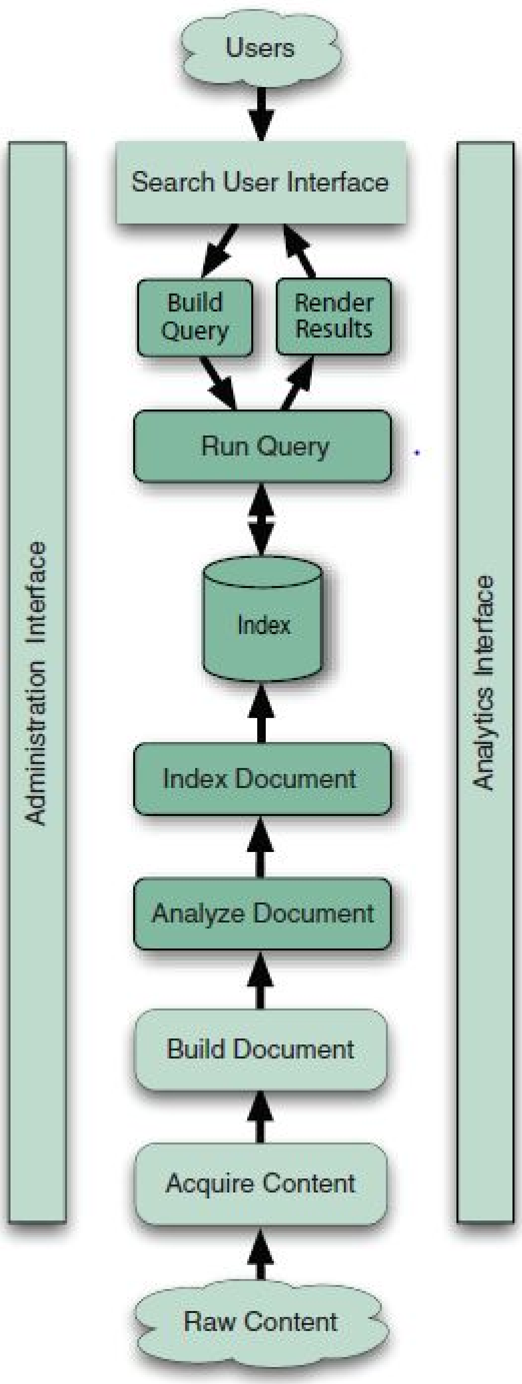
2. b) Term Normalization

Pre-processing of the documents is carried out. Steps usually including Tokenization and Stop Word Elimination and use of a Stemmer. Tokenization is a process where in each word in the document is considered as a Token on which Stop Word Elimination is applied where all the articles, prepositions, gerund forms, past tense words etc. are removed. We have considered an English dictionary which consists of 33 words which are referred to in the process. Finally, we have used a Stemmer (Porter Stemmer) with English Analyzer which carries out the Stemming process. This entire step basically removes terms from the English Corpus having morphological inflectional endings.

2. c) Ranking Algorithm

An appropriate ranking algorithm is used to sort the collection of documents based on their similarity to retrieve the best possible results based on the user's query. OKapi BM25 is used her which is based on the probabilistic model that ranks the documents based on the probability of term occurring in the document is either 0/1. If the rank if the model is 1, then the given query term appears in the document collection. If the rank is 0, then query term is not present in the document collection. Another important aspect of these kind of models is that they don't consider the relationship between query terms with the corpus. In our case, we are retrieving the top 10 relevant documents.

3. Program Flow [1]



[1]: ***Lucene in Action*** by Michael McCandless, Erik Hatcher and Otis Gospodnetic’

4. Structure of the Program

Main: The main program takes input from the user and validates the length of the arguments given in the command line. If valid, *Indexer.java* is invoked to index all the text files in the given directory. This is followed by invoking the *Searcher.java* class, which would search the given query in all indexed files. Else, the program would exit with error message.

Indexer.java: Parameters document and index path are taken as input. It checks if the document file folder is readable. If yes, the indexing time is calculated.

PerformIndex(docsPath, indexPath): Validates whether the document file folder is readable. If the file is not readable, then the program would exit. Else, the program would start Indexing.

StartIndex(): Creates an *EnglishAnalyzer* which in turn creates *IndexWriterConfig*. This method calls *indexDocs* which carries out the indexing operation. It also prints the time taken to index all the files.

Setsimilarity: Score of the document is given by *BM25Similarity*.

IndexWriter: Writes a new index or updates the existing index.

indexDocs(writer, dir): Indexes the file in recursive manner over the directory of files. If directory is readable. If yes, this method in turn will in turn invoke the *indexDoc(IndexWriter writer, Path file, long lastModified)* file to index a document.

indexDoc(IndexWriter writer, Path file, long lastModified): It calls the Stemmer to perform the stemming operation. Creates an index for the new document or index would be updated for the path which matches query. For HTML files, additional fields such as Title, Content and Summary are added.

Searcher.java: Inputs are document path, index path, and user's query. If it is readable, it will call *parseQueryAndSearch*. Search is done on document's body and title.

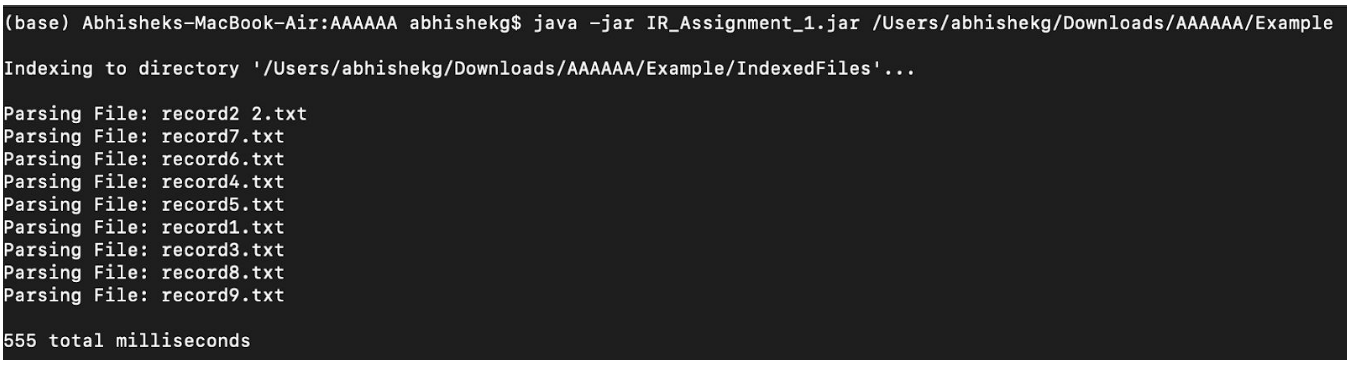
parseQueryAndSearch: *BM25Similarity* will be set to the query and pre-processing of the query is carried out using *EnglishAnalyzer*. Multi field query parser is used to query multiple fields concurrently i.e., content and title for paging search to specify how many number of records match. This is followed by invoking the *doSearch()*.

doSearch(docpath, indexPath, query): Searches the top 10 hits and prints the respective fields of the document. At first-go, only the top 10 results are printed followed by a question to the user if he/she wants the next set of results. This can be done by pressing the "n" button.

Building an Information Retrieval System using Apache Lucene

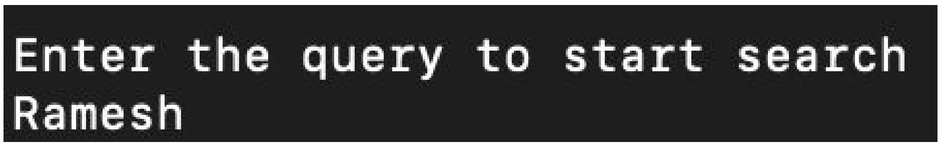
5. Output

Listing all the Parsed files:



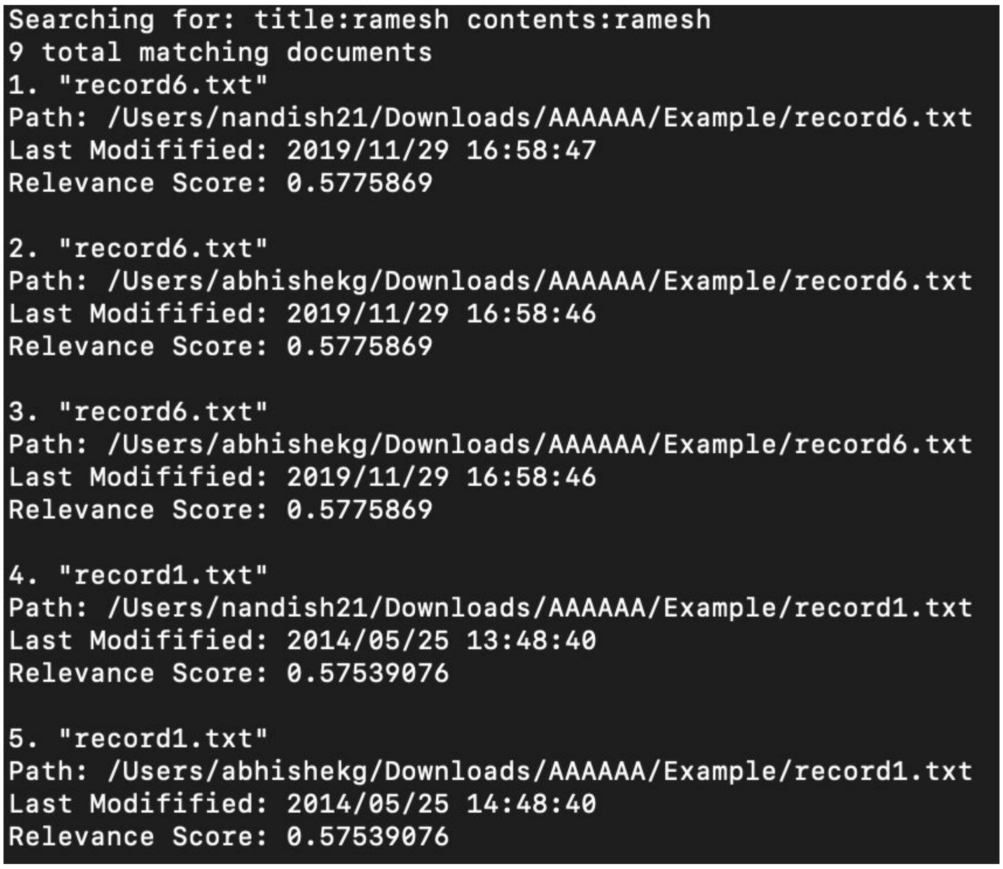
```
(base) Abhisheks-MacBook-Air:AAAAAA abhishekg$ java -jar IR_Assignment_1.jar /Users/abhishekg/Downloads/AAAAAA/Example

Indexing to directory '/Users/abhishekg/Downloads/AAAAAA/Example/IndexedFiles'...

Parsing File: record2 2.txt
Parsing File: record7.txt
Parsing File: record6.txt
Parsing File: record4.txt
Parsing File: record5.txt
Parsing File: record1.txt
Parsing File: record3.txt
Parsing File: record8.txt
Parsing File: record9.txt

555 total milliseconds
```

User's Query:



```
Enter the query to start search
Ramesh
```

Listing the Top 10 results:



```
Searching for: title:ramesh contents:ramesh
9 total matching documents
1. "record6.txt"
Path: /Users/nandish21/Downloads/AAAAAA/Example/record6.txt
Last Modified: 2019/11/29 16:58:47
Relevance Score: 0.5775869

2. "record6.txt"
Path: /Users/abhishekg/Downloads/AAAAAA/Example/record6.txt
Last Modified: 2019/11/29 16:58:46
Relevance Score: 0.5775869

3. "record6.txt"
Path: /Users/abhishekg/Downloads/AAAAAA/Example/record6.txt
Last Modified: 2019/11/29 16:58:46
Relevance Score: 0.5775869

4. "record1.txt"
Path: /Users/nandish21/Downloads/AAAAAA/Example/record1.txt
Last Modified: 2014/05/25 13:48:40
Relevance Score: 0.57539076

5. "record1.txt"
Path: /Users/abhishekg/Downloads/AAAAAA/Example/record1.txt
Last Modified: 2014/05/25 14:48:40
Relevance Score: 0.57539076
```