Information Retrieval
Programming Task

Team
Michael Mario Kubicki

Used Software

- Java
  - Tested on Oracle Java 1.8 and OpenJDK 11

- Apache Lucene (Version 7.6.0)
  - core
  - analyzers-common
  - codecs
  - queryparser

- Jsoup (Version 1.11.3)

Usage:

java -jar IR_P.jar [optional: flags -h/-v] [document_dir] [index_dir] [VS/OK] [query] [optional: path_to_settings.xml]

optional flags:
- h display short help message
- v „verbose“ output from indexer

document_dir:
Path to documents
Will be checked if it exists. Program will exit if it does not exist.

index_dir:
Path to index
Will be created if it does not exist. Program will try to use existing index

VS/OK:
Ranking Model
- VS = Vector Space (Lucene ClassicSimilarity)
- OK = Okapi BM25 (Lucene BM25Similarity)

query:
Query to search for in documents
Put into quotation marks if more than one word is used

path_to_settings.xml:
Can be used to set additional settings in program

Layout:
```
<settings>
        <number_results>10</number_results>
        <file_types>.txt;.htm;.html</file_types>
</settings>
```

Both fields are optional
number_results is the number of ranks to be displayed. (Integer)
file_types are the extensions the program will look for. It will already always search for .txt, .htm and .html (semicolon separated list)

How it works

The two main classes of this program are Indexer and Searcher. They use functions and methods from the Lucene library to index and query the documents.
Both use the pattern, that they need to be setup in order to use them with their respective main-method (index and search).

For setup the indexer needs to know the used index directory, the ranking model, a general FileIndexer and a CheckedList. With this information a IndexWriterConfig and IndexWriter from Lucene will be created.
With the IndexWriterConfig some information is set:

- Analyzer for text: CustomAnalyzer setup from different Filters
- Open mode: Create or Append index
- Commit changes to index on close
- Similarity: Classic (Vector Space) or BM25 (Okapi BM25)

The Analyzer used for the whole process has the following parts:

- StandardTokenizer (grammar-based tokenizer using rules from Unicode Text Segmentation algorithm)
- LowercaseFilter (lowercase words)
- StopFilter (filtering standard english stopwords)
- PorterStemFilter (apply Porter stemming)

After the setup using the constructor or the SetUp(...)-method, Index(...) can be called with the directory containing the to be indexed documents and a set of file extensions to index.
The Index(...)-method takes every file recursively from the given document directory, filters them for the given extensions and checks in the CheckedList if the file is new, is already indexed, was updated since the last indexing or a file was removed. The new files will simply be added to the index with the IndexWriter.addDocument(doc)-method, the updated ones will be updated with IndexWriter.updateDocument(term_to_identify, doc) and the removed ones will be removed through IndexWriter.deleteDocuments(term_to_identify, doc).
To Index the files they need to be put in a Lucene-Document object. For this FileIndexers will be added to the Indexer object. They get called if their registred extension matches the specific file and it is expected, that they return a Document object containing the data of the file.
In the program there are two FileIndexer used: PlainIndexer and HTMLIndexer.
The HTML Indexer uses Jsoup to take the title and unformatted body text from an html or htm file, which are put into the title- and content-TextField. Also the path to the file will be stored in a StringField. The path and title of a file will be stored in the index, while only the title and content will be indexed.
The PlainIndexer also uses the title- and content-textfield and the path-StringField. In this case the title holds the filename and the content the whole document.

After every file is indexed Indexer.Close() needs to be called to commit the changes to the index and after that call returns the index is ready to be used.

The Searcher is SetUp with a given index directory, a ranking model—which should match with the one given to the indexer—and at least one field identifier to search within. In the SetUp a Lucene IndexReader on the index directory, a Lucene IndexSearcher using the ranking model, and a MultiFieldQueryParser using the field identifiers and the same CustomAnalyzer as above are created. Those are needed to search the index for a query.
After the setup Search(...) can be used to search for the query. Search returns an array of Lucene ScoreDocs containing the searchresult with their score. Those only need to be displayed after they are returned.
Of every result (in the top 10) the rank, score, filename and path are displayed. For hypertext files even the title will be shown.

After it has been used once it can be continued to be used in the same way and the index will only be modified if anything in the document directory has changed and even then only the modifications will be reindexed.