This review has been setup to analyze and understand how Lucene can be used to evaluate the Cranfield data set, given a set of queries and their relevance judgement.

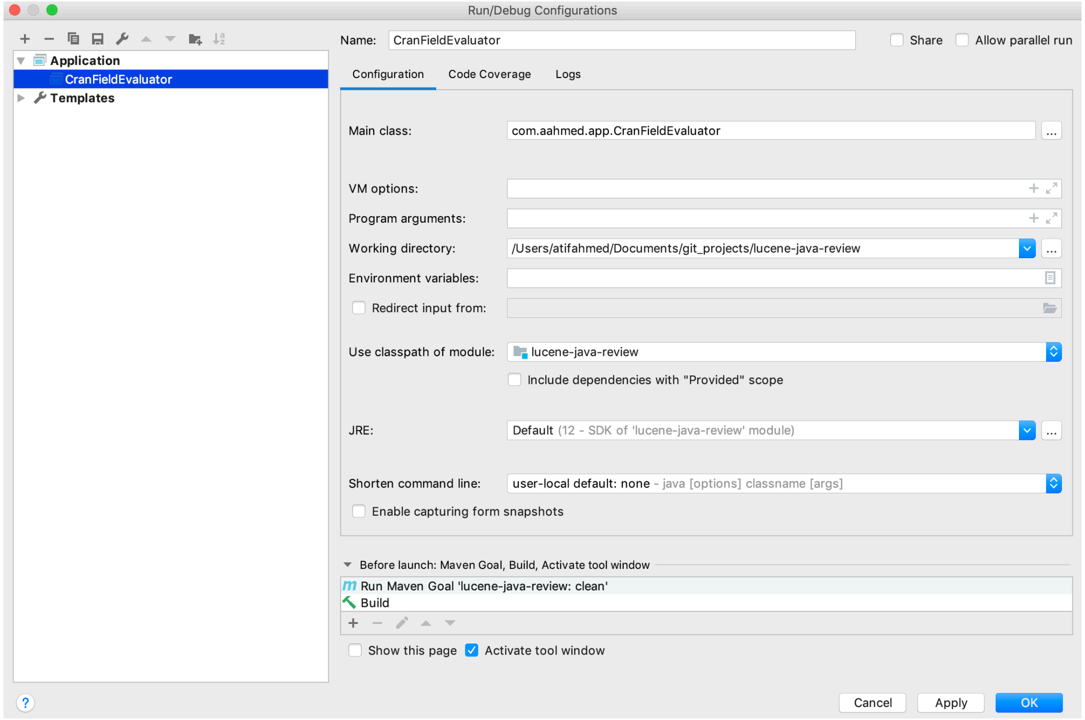
Lucene is an open source Java based search library. It is very popular and a fast search library. It is used in Java based applications to add document search capability to any kind of application in a very simple and efficient way. This tutorial will give you a great understanding on Lucene concepts and help you understand the complexity of search requirements in enterprise level applications and need of Lucene search engine.

Data Set:

- Cranfield Data containing 1400 documents
- About 225 Queries
- And finally, the relevance judgement of those queries

Execution:

Execution configuration for execution through IDE:



Run/Debug Configurations

Application
- CranFieldEvaluator

Templates

Name: CranFieldEvaluator □ Share □ Allow parallel run

Main class: com.aahmed.app.CranFieldEvaluator

VM options:

Program arguments:

Working directory: /Users/atifahmed/Documents/git_projects/lucene-java-review

Environment variables:

□ Redirect input from:

Use classpath of module: lucene-java-review

□ Include dependencies with "Provided" scope

JRE: Default (12 - SDK of 'lucene-java-review' module)

Shorten command line: user-local default: none - java [options] classname [args]

□ Enable capturing form snapshots

Before launch: Maven Goal, Build, Activate tool window

Run Maven Goal 'lucene-java-review: clean'

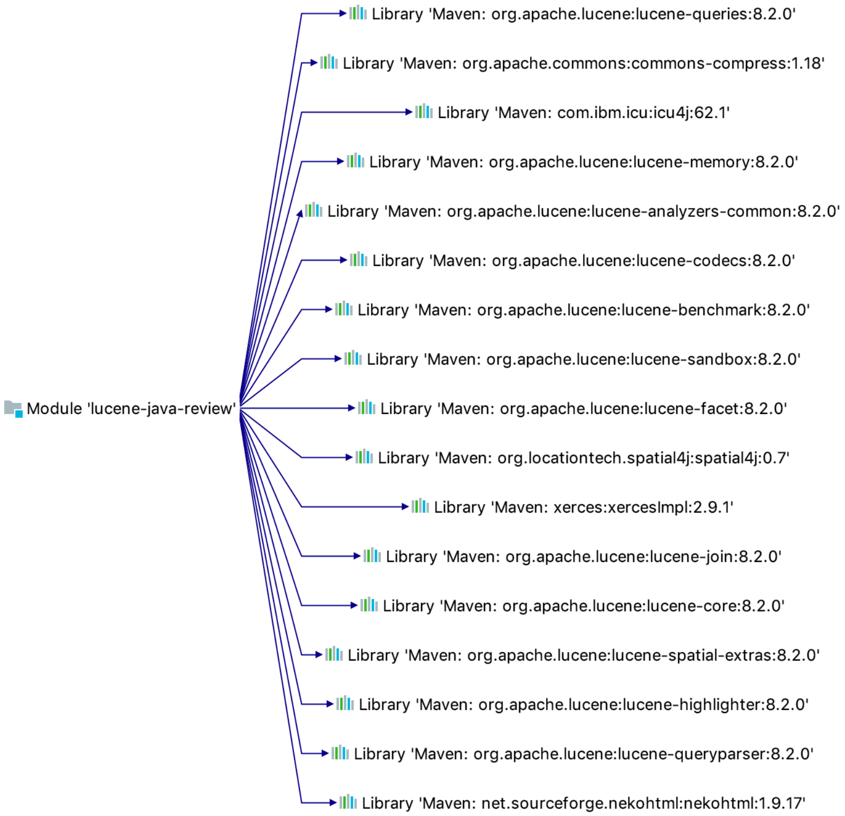
Build

□ Show this page ☑ Activate tool window

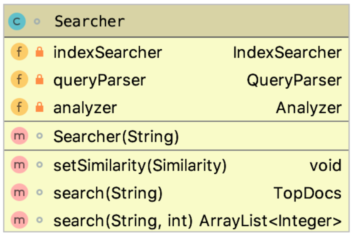
Using maven:

- mvn compile
- mvn exec:java -Dexec.mainClass="com.aahmed.app.CranFieldEvaluator"

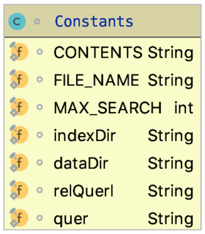
Library dependencies:


Module 'lucene-java-review'
Library 'Maven: org.apache.lucene:lucene-queries:8.2.0'
Library 'Maven: org.apache.commons:commons-compress:1.18'
Library 'Maven: com.ibm.icu:icu4j:62.1'
Library 'Maven: org.apache.lucene:lucene-memory:8.2.0'
Library 'Maven: org.apache.lucene:lucene-analyzers-common:8.2.0'
Library 'Maven: org.apache.lucene:lucene-codecs:8.2.0'
Library 'Maven: org.apache.lucene:lucene-benchmark:8.2.0'
Library 'Maven: org.apache.lucene:lucene-sandbox:8.2.0'
Library 'Maven: org.apache.lucene:lucene-facet:8.2.0'
Library 'Maven: org.locationtech.spatial4j:spatial4j:0.7'
Library 'Maven: xerces:xercesImpl:2.9.1'
Library 'Maven: org.apache.lucene:lucene-join:8.2.0'
Library 'Maven: org.apache.lucene:lucene-core:8.2.0'
Library 'Maven: org.apache.lucene:lucene-spatial-extras:8.2.0'
Library 'Maven: org.apache.lucene:lucene-highlighter:8.2.0'
Library 'Maven: org.apache.lucene:lucene-queryparser:8.2.0'
Library 'Maven: net.sourceforge.nekohtml:nekohtml:1.9.17'

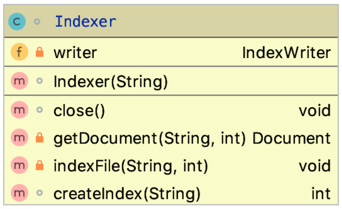
Core class implementations:



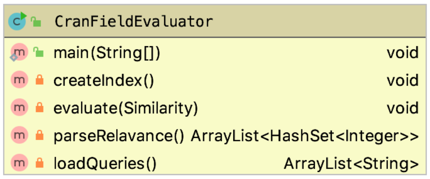
Searcher	
indexSearcher	IndexSearcher
queryParser	QueryParser
analyzer	Analyzer
Searcher(String)	
setSimilarity(Similarity)	void
search(String)	TopDocs
search(String, int)	ArrayList<Integer>



Constants	
CONTENTS	String
FILE_NAME	String
MAX_SEARCH	int
indexDir	String
dataDir	String
relQuerl	String
quer	String



Indexer	
writer	IndexWriter
Indexer(String)	
close()	void
getDocument(String, int)	Document
indexFile(String, int)	void
createIndex(String)	int



CranFieldEvaluator	
main(String[])	void
createIndex()	void
evaluate(Similarity)	void
parseRelavance()	ArrayList<HashSet<Integer>>
loadQueries()	ArrayList<String>

Explanation

Building the index in the target directory:

- Initialize the index directory, with index writer
- Create index for each document (run the iterator for each line on dataset document)
  - Set the document first to identify the content and a unique field name to identify the document.
- Use the index writer to write the above processed document

Evaluator:

- Read the created index via Searcher class.
- Set the similarity(). This can be any scorer function, here its BM25 with default configuration of k1=1.2,b=0.75.
- Load the queries (as list of individual queries) and relevance
  - Relevance has 3 components, the query, document ID, the relevance judgement.
  - VERY IMPORTANT -> filter the relevant document based on relevance score i.e. 1 or 2, else consider its irrelevant.
- For each query:
  - Search in the created index for MAX result (here it's 10)
  - Get the relevance judgement and its components for this given query
  - If the "top 10" resulted document ID is inclusive in the set of the relevance

        - Then count how many “true positive” out of this “max 10” is contained in the relevance judgement set.
        - Also calculate the cumulative precision by normalizing each precision as current “true positive” / “the loop value of hit document ID”
    - The finally calculate the average precision to be cumulative precision of above step by count of “true positives”
    - Recall can be calculated as count of “true positives” by the size of relevance judgement for that query.
- Mean average precision will be cumulative average precisions by size of overall relevant document judgement.
- Mean recalls will be cumulative recalls by size of overall relevant document judgement.

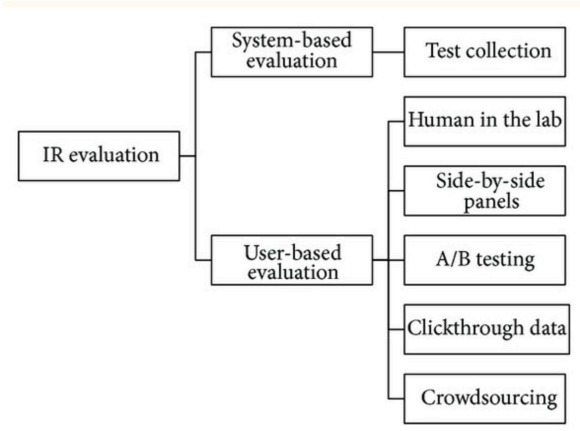
Some useful information

What’s index writer?
https://lucene.apache.org/core/8_0_0/core/overview-summary.html

What’s index reader?
https://lucene.apache.org/core/8_0_0/core/org/apache/lucene/index/IndexReader.html

What’s similarity?
http://lucene.apache.org/core/8_0_0/core/org/apache/lucene/search/similarities/Similarity.html?is-external=true

What’s relevance?
https://en.wikipedia.org/wiki/Relevance_(information_retrieval)



What’s average precision, recall and MAP?

https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)#Mean_average_precision

For detailed implementation follow, GitHub:
https://github.com/atif-github-venture/lucene-java-cranfield-analysis

Source:
https://www.tutorialspoint.com/lucene/
https://github.com/PointerFLY/Lucene-Example
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4055211/