# ForSearch

## CSC 575, Intelligent Information Retrieval, Final project

## Winter 2012, Author: Brian Fors

### Project Description

This project is an implementation project for a search/retrieval system. The system was built and tested against a local document test collections, including the Time, Medlars, and Cranfield test collections, all available on the common IR test collection website: <http://web.eecs.utk.edu/research/lsi/corpa.html>. The testing compared search results for this implementation against the first 10 reference set queries available in the Medlars collection.

The system parses the specified set of documents and creates an in-memory index using an inverted file structure. The system by default executes pre-processing on the documents and query text, doing both stemming and excluding words in the stop list. For stemming, the system uses an existing Java implementation by Martin Porter: <http://tartarus.org/martin/PorterStemmer/java.txt>. The small stop list referenced from the class website is used: <http://www.lextek.com/manuals/onix/stopwords1.html>.

The system normalizes term weights using the tf-idf term normalization algorithm. Retrieval match and ranking is done using a vector based model using cosine similarity measure for ranking results.

The query form is a simple list of space-delimited query terms. The system does not support any complex Boolean or Not logical operators. The query assumes the use of logical AND across the set of query terms provided by the end user. The top 50 search results are displayed in order of highest to lowest ranked results. The system display the document ranking #, document #, cosine similarity value and the first 80-characters for each document in the result list.

The system is built as a command-line based Java implementation. The user interface is a simple text-based interface.

### Instructions to BUILD, RUN the application

**Main class:** com.fors.ir.controller.Main

The application requires JRE6 (Java 1.6 compiler) and build file uses Ant 1.7. The application runs on Windows OS.

**To run**, extract files from the zip file, and execute the following from command line in the application root folder “ForSearch” (containing build.xml):

ant build

ant run

Once run, the application first prompts the user for which document set to be indexed. For example:

Which document set to index (0=TIME, 1=MEDLARS, 2=CRANFIELD):

> **1**

The system returns the number of terms and documents indexed from the collection –

11273 terms loaded.

1033 documents loaded.

Next, the application will prompt the user to enter a query. For example:

Enter query:

> **the crystalline lens in vertebrates, including humans**

The system returns the query text, and pre-processed query terms (stemmed and excluding stop words) and also returns the TOP 50 search results, ranked in descending order. For each result, the system displays the ranking #, document #, cosine similarity score and 1st 80-characters of the document –

Query: the crystalline lens in vertebrates, including humans

includ

vertebr

crystallin

len

human

Document Text

Document #

Ranking #

Cosine Similarity Score

============================

TOP 50

============================

1-965-5.72-posterior scalloping of vertebral bodies in uncontrolled hydrocephalus. two case

2-500-3.646-1949. studies on the soluble proteins of bovine lens. immuno- chemical analyses

3-499-2.515-1463. investigations of lens protein and microelectrophoresis of hydrosoluble pr

4-15-1.962-lens development.. the differentiation of embryonic chick lens epithel

5-511-1.766-1747. the problem of albuminoid albuminoid is the main constituent of the insolu

6-212-1.661-experiments dealing with the role played by the aqueous humor and retina in lens

7-637-1.613-some intraspecies differences in antigens on the surface of certain living human

8-181-1.5-the insoluble proteins of bovine crystalline lens . the i

9-206-1.486-isozymes of lactic dehydrogenase.. sequential alterations during develop

10-142-1.483-the effects of electrophoretically separated lens proteins on lens regener

...

The user can display the full context of a document by entering \*D###, where ### is the document number. For example:

Enter query:

> **\*D965**

The system returns the complete document text for the document number entered –

posterior scalloping of vertebral bodies in uncontrolled hydrocephalus. two cases of extensive posterior scalloping of the vertebral bodies are presented in men aged 17 and 23 years, having long-standing hydrocephalus. two additional cases with scalloping of only one lumbar vertebra when partially controlled hydrocephalus has been present for a shorter time are also noted. no previous association between these entities has been recorded. it is supposed that the increased intraspinal pressure which must have been present in the first 2 patients for many years, was present near the time of closure of the epiphysis at the junction of the arch and the bodies and caused not only widening of the spinal canal but also excavation of the vertebral bodies. scalloping of vertebral bodies has been described in: (1) neoplasms (neurofibromas, meningiomas, gliomas, hemangio-endotheliomas, hemangiomas, lipomas): (2) intraspinal cysts (intradural arachnoid cysts, tarlov's perineural cysts, thoracic extradural cysts in kyphosis dorsalis juvenilis): (3) congenital anomalies of the spine and cord (fusion defects, myelodysplasia, hydromyelia, absence of a single vertebral pedicle, eningoceles): and (4) neurofibromatosis (with or without a thoracic meningocele).

Finally, the user can quit the application by entering Q. For example:

Enter query:

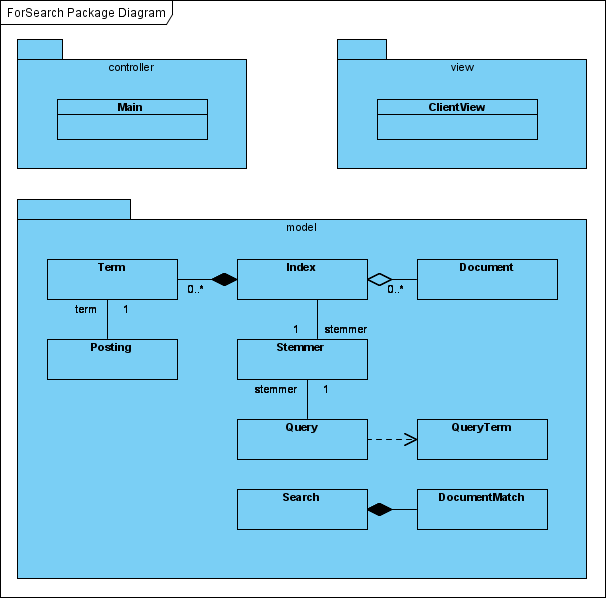
> **Q**

End.

## System Design Description

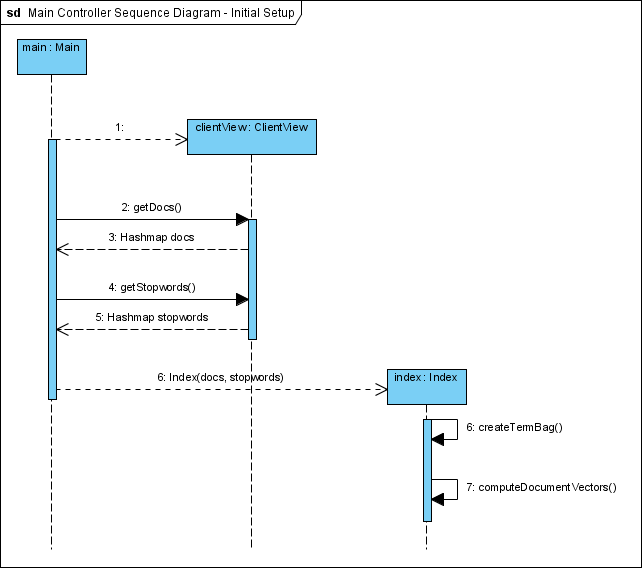
The system has 3 packages:

1. **Controller** – contains the Main.java class, creates the ClientView for reading parameters from user, and creates the Index and initiates the loop for executing queries
2. **Model** – contains classes that create the inverted document index, execute searches and rank search results
3. **View** – contains ClientView class which prompts users for input parameters and understands how to parse the IR datasets (TIME, MEDLARS and CRANFIELD)



#### Initial Setup

The following sequence diagram describes the initial system setup.



**Step 1:** The controller Main creates an instance of ClientView.

**Step 2 and 3:** The controller Main requests the test document collection from the ClientView, based on the user-specified test collection set. The ClientView returns a HashMap containing the set of Documents indexed by document number.

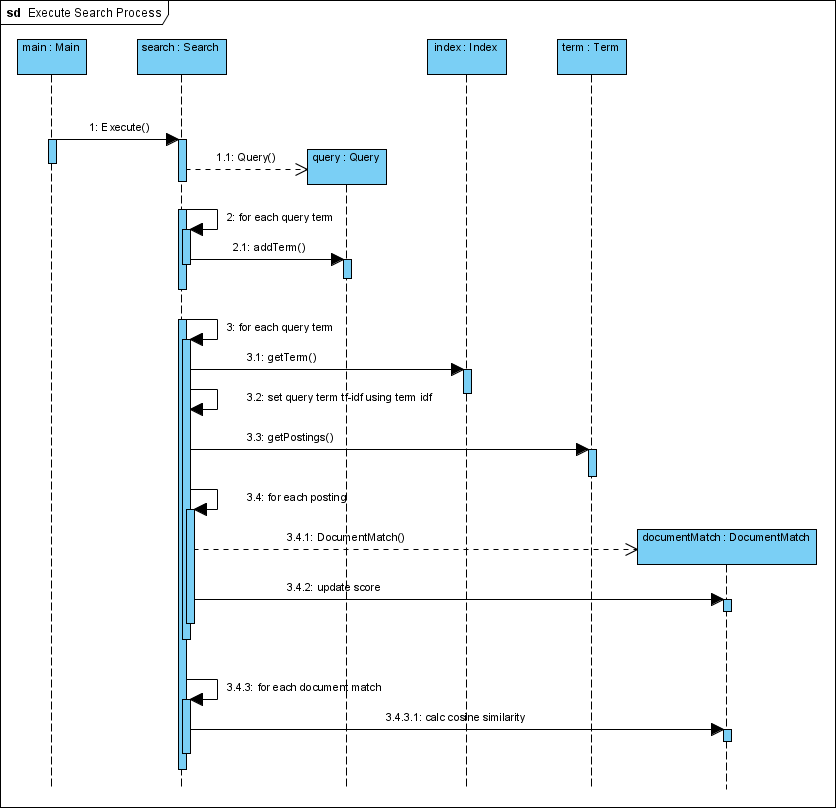
**Step 4 and 5:** The controller Main requests the stop word collection from the ClientView. The ClientView returns a HashMap containing the set of stopword Strings, indexed by stop word.

**Step 6:** The controller Main creates an instance of Index, passing the documents HashMap and stop words HashMap into the Index. The Index creates a local TermBag by iterating through the set of documents, and iterating through each term in the document. Each term maintains a Postings HashMap indexed by Document ID and storing the term frequency for each document. Once this step is complete the td-idf value for each posting is available.

**Step 7:** Lastly, the Index computes the document vectors, by iterating through the Terms set, and for each term, iterating through the set of Postings, cumulatively adding the td-idf value for each posting to document vector.

#### Search Process

The following sequence diagram describes the search execution process.



**Step 1:** The controller Main calls the Execute method on the Search class, passing a reference to the Index and passing the query string

**Steps 1.1 thru 2.1:** The Search class creates an instance of Query class, adding each query term to the instance

**Steps 3.1 thru 3.2:** TheSearchclass iterates through each query term, getting a reference to the Term from the Index, in order to set the query term tf-idf value.

**Steps 3.3 thru 3.4.2:** For each Term from the Index, the Search class then iterates through the set of Postings for them Term, adding each Posting to a collection of document matches. The score for each document match is accumulated using the query term tf-idf multiplied by the posting tf-idf. The value is calculated as:

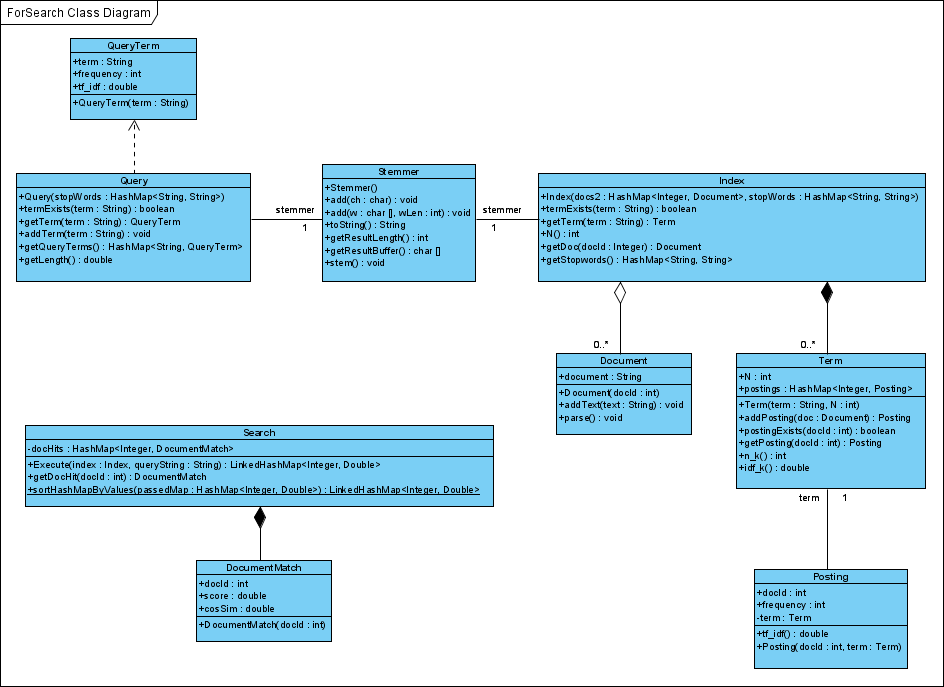
docMatch.score += queryTerm.tf\_idf \* posting.tf\_idf();

**Steps 3.4.3 thru 3.4.3.1:** Finally the Search class then iterates through each document match, calculating the cosine similarity for the document. This value is calculated as:

docMatch.cosSim = docMatch.score / (query.getLength() \* doc.getLength());

Finally, the document match results are sorted and returned in descending order of cosSim.

#### Detailed Class Diagram for the Model Package



## Sample Test Runs – Summary

The following are results for sample test queries against the Medlars test collection. The first 10 queries and relevance judgments from the test collection are compared against the query results from the ForSearch system implementation.

See the attached Excel file – “ForSearch Test Results - Medlars collection.xlsx” for detailed results. Below is a summary of the test results, showing average recall and precision of the ForSearch implementation against the first 10 queries included with the Medlars collection.

Below is a snippet from the Excel file to show the format of the test data captured.



…



For each of the ten queries, the spreadsheet shows the query text, the documents judged relevant from the test collection, and the documents returned by ForSearch. If a relevant document was not found, it is highlight in red to show a gap. The total number of relevant documents found is summed, along with the total number of relevant documents and the total number of documents returned by ForSearch. These values are used to calculate Recall and Precision for each query.

The result shows **Total Average Recall = 71.55%** and **Total Average Precision = 29.45%**.

The Precision of the system could be tweaked by determining an appropriate threshold value for document cosine similarity, in order to determine when to exclude a document from the search results. The system currently uses a fixed value to return the top 50 search results.

The Recall of the system could be improved by returning more than the top 50 search results, and by improving the search algorithm. Some ideas for improving the algorithm are included in the Future Enhancements section below.

## Future System Enhancements

1. Add term positioning to the model in order to improve the precision of search. This would add precision by accounting for term co-location in order to gauge relevance
2. Add a cosine similarity cutoff threshold for determining whether to include a document in the search results. Currently, any document with a non-zero value is included
3. Add the ability to store the index to a permanent data store
4. Add the ability to add delta (new) documents to the index without having to re-index the complete document set
5. Add ability to partition the indexing process and split the indexing across multiple parallel execution threads
6. Load all of the test queries and their relevance judgments into the system so that a calculation of precision and recall can be done by the system, rather than manually, across all sample queries
7. Tune the values that the system uses as term delimiters in parsing the documents
8. Modify the system to parse a set of documents from the local file system, beyond just loading test collections (including Word, Text, Excel, Outlook documents and files, etc.)
9. Create a graphical user interface for navigating and viewing search results
10. Incorporate additional models for ranking results, beyond just cosine similarity (Dice’s, Jaccard’s, etc.)