1 Overview

In this homework assignment, you will implement components related to document processing and indexing, as well as refactor your ranking components developed in HW1 to take advantage of improved indexing components. We will provide you with the skeleton code that you will augment with the desired functionalities.

1.1 Architecture Review

Our search engine consists of the following main components.

- **Server**: This component includes SearchEngine and QueryHandler. It starts the server, accepts the queries from the users, and renders the results back to the users. The server decides on which Indexer to use at server starting time based on a configuration, and which Ranker to use at query time based on the CGI arguments.

- **Indexer**: This component handles the document indexing offline and document retrieval online. It includes the abstract class Indexer and all sub-classes inherited from it. In HW1, instructors provided an implementation that loads the whole corpus from a text file during server starting time and scans all the documents in the corpus each time a query is issued. This simple implementation is now the class IndexerFullScan.

- **Ranker**: This component handles the ranking of the documents. It includes the abstract class Ranker and all sub-classes inherited from it. In HW1, instructors provided an implementation that gives a score of 1.0 to a document as long as a query term matches one of its title terms. As part of HW1, you provided a few alternative implementations of the Ranker class.

- **Evaluator**: This component takes user judgements and evaluates the performance of the search engine.

- **Document** and **Query**: Representations of documents and queries, respectively. Document stores information about the documents necessary for both ranking and result presentation. ScoredDocument is a simple wrapper of Document to provide a score. Query stores the raw queries and any state resulting from query processing.

1.2 Download and Verify Instructor Code

1. Location: `/home/congyu/cs2580/hw2/instructor/` on linserv machines. Inside the directory, we have:

   - **src**: contains the skeleton code from the instructors. You should copy this into your group directory (again, maintaining the directory hierarchy).
   - **conf**: this is where configuration files (e.g., `conf/engine.conf`) sit. The server starts by reading the options in the configuration file and learn where to construct/load the index and what kind of indexer to use. You should copy this into your group directory.
   - **data**: contains three sub-directories. **data/simple** is the simple corpus from HW1, provided here to facilitate code refactoring. **data/wiki** is the wiki corpus to be used in HW2, which contains a dump of about 10K Wikipedia articles. **data/index** is where the constructed index should reside. You should create the same directory structure in your own group directory and copy over the simple corpus.

For the wiki corpus, the total size is 1.3G and they are read-only, do **NOT** copy it over to your own code directory. Instead, create a link inside your group data directory to the wiki directory, i.e.,
2. Compile.
congyu@linserv1[code]$ javac src/edu/nyu/cs/cs2580/*.java

3. Construct the index.
congyu@linserv1[code]$ java -cp src edu.nyu.cs.cs2580.SearchEngine \
   --mode=index --options=conf/engine.conf
You should see the following message:
Using Indexer: IndexerFullScan
Construct index from: data/simple/corpus.tsv
Indexed 659 docs with 2070842 terms.
Store index to: data/index/corpus.idx

4. Start the server.
congyu@linserv1[code]$ java -cp src -Xmx256m edu.nyu.cs.cs2580.SearchEngine \
   --mode=serve --port=258XX --options=conf/engine.conf
You should see the following message:
Using Indexer: IndexerFullScan
Load index from: data/index/corpus.idx
659 documents loaded with 2070842 terms!
Listening on port: 258XX
Note that you are allowed to use up to -Xmx512m for your implementations.

5. Test the server through another terminal:
congyu@linserv1[~]$ curl 'localhost:258XX/search?query=web&ranker=fullscan'
You should see 10 results returned, the first one being “comparison of web search engines.”

2 Tasks

2.1 Implementing Three Concrete Indexer Classes
This is your main task. Based on the abstract Indexer class, each group will implement three concrete classes, IndexerInvertedDoconly, IndexerInvertedOccurrence, and IndexerInvertedCompressed. As the names suggest, the first concrete implementation should index terms with the documents they appear in. The second one should index terms with their occurrences (i.e., offsets) in the documents. And the last one should perform occurrences indexing but with the resulting postings lists compressed.

Each implementation should have constructIndex and loadIndex functionalities. The former is used offline (i.e., mode=index) to construct the index from the corpus. The latter is used online (i.e., mode=serve) to load necessary data for serving search traffic.

Each implementation should have the nextDoc functionality for document retrieval as discussed in class. This functionality should support both conjunctive retrieval and phase retrieval. In particular, phrase retrieval should no longer be limited to phrases of size 2 as in HW1.

2.2 Implementing One Concrete Ranker Class
Each group will also refactor one of their Ranker implementations (except RankerPhrase) from HW1 to use the new indexer implementations. This should be done via the RankerFavorite class. A simple non-tested class RankerConjunctive is provided for illustrations purpose.
2.3 Improving Document and Query Representations

In HW1, instructors’ Document implementation stores all tokens within a document during query retrieval, which is very inefficient. In HW2, each group should implement DocumentIndexed so that only the necessary information are retrieved from index for ranking purposes. The base Document class maintains four fields, title, url, pageRank, numViews. The first two should be populated properly and the last two can be ignored for HW2.

In HW1, instructors’ Query implementation maintains a simple token vector. In HW2, each group should implement QueryPhrase so that phrase based queries can be supported. Phrase queries are represented using quotes, e.g., “new york city.” Unlike in HW1, phrases can be arbitrary length and be combined with other query tokens.

More details are inside the base class files: Indexer, Ranker, Document, and Query classes.

3 Grading

General grading instructions can be found in the comment section inside the Grader class. Note that we will use hidden sets of queries to evaluate your implementation. A few important notes. First, don’t try to guess what queries we will use for grading, we don’t even know yet. Second, absolutely do not use full scan in HW2, it will be slow and easy to detect both in the code and in execution. Third, make sure your index can be used without the original raw corpus, we will remove the corpus before testing your index loading and serving. Forth, as a reminder, your final project will be using your own implementations of the rankers and indexers.

A bonus task is also provided, see Grader class for details.

4 Submitting Your Code

- Follow the same instructions as in HW0 and HW1. Important: The directory data/wiki has 10K files with a total size over 1G, do NOT submit that. We will create a softlink to the instructor’s data/wiki directory for grading.

- Make sure your works on linserv machines, together with all the necessary configurations. Provide a readme file within the submission to give us any additional information.