1. **Document processing and indexing**.

You will be provided with a zip file that contains 63 HTML documents collected from Wikipedia.

* ~~First, pre-process the documents by removing all HTML tags and convert everything into lower case. Implement a stop list and a stemmer to pre-process the documents (for the stop list and stemmer, you are allowed to use third-party open source code).~~
* ~~Second, build an inverted index (including dictionary and posting lists) for the documents. Please make sure to keep all the frequency information.~~

2. **Vector Space model**

The goal is to provide a TF-IDF-based ranking for the documents.

* ~~Since you have already collected frequency information in step 1, please further compute IDF for each term.~~
* ~~For each document, find a way to calculate the length of the corresponding document vector.~~
* For each incoming query, pre-process the query with the stop list and stemmer. Identify candidate documents that contain at least one query term.
* ~~Meanwhile, compute the length of the query vector.~~
* Finally, compute the TF-IDF similarity score between the query and each candidate document [hint: there is no need to construct the complete document vector, or loop through all dimensions in the vector space], and sort the documents by the score.

3. **Niche crawler**.

* You should identify a domain of interest (e.g., ku, Wikipedia, nfl, etc.). Ideally, the size of the domain should be manageable, and the link structure is not too complicate to follow.
* Your crawler should contain at least three components: (1) a multi-threaded spider that fetches and parses webpages, (2) the URL frontier which stores to-be-crawled URLs; and (3) the URL repository that stores crawled URLs. Please be polite to the site.
* Please collect a few hundreds to a few thousands of pages.

4. Please feed the collected documents to the search engine that you implemented in step 2. Please implement a Web-based interface to take user queries and return answers (document names, snapshot with search term(s) highlighted, and URL) to the user. You only need to provide a reasonable (not so fancy) interface, you can use WYSIWYG editors to generate HTML. Keep this version of your search engine, since it will be compared with two future versions.

5. Add term proximity into your scoring mechanism. Define your own score that reflects the proximity of search terms in each document. Define your own algorithm to integrate term proximity score with the tf-idf score from step 2.

6. Add one of the following to your search engine:

6.1. Search personalization: use cookies to track users. Record each search and each click-through. For a new query, add a small component of the "search history" as query expansion.

6.2. Relevance feedback. For each query, allow the user to identify a set of "positive" and "negative" results. Use user feedback to update the query and return new (refined) results to the user.

7. Please evaluate and compare the performance of the original search engine (step 4), and the new versions (step 5 and 6).

You are expected to do a project presentation during the last two weeks of the semester. Please make a few powerpoint slides to introduce your search engine, especially the system architecture and design notes. Please also prepare a short demonstration of your search engine.

You need to submit a project report will all your source code (please do NOT include crawled documents). It is strongly suggested that you make the search engine available when we grade your project report.