UCI Information Retrieval

Project 3 – Search Engine

Due dates: 3/1, 3/8 and 3/15

This assignment is to be done in groups of 1, 2 or 3, preferably the same groups that were in place for Project 2. Although this is presented as one single project that will take until the end of the quarter to complete, internally it is organized in 3 separate milestones, each with a specific deadline, deliverables and score. In doing milestones #1 and #2, make sure to look at the evaluation criteria not just of those milestones but also of milestone #3 –part of the milestones' evaluation will be delayed until the final meeting with the TA.

You can use code that you or any classmate wrote for the *previous* projects. You cannot use code written for *this* project by non-group-member classmates. Use code found over the Internet at your own peril -- it may not do exactly what the assignment requests. If you do end up using code you find on the Internet, you must disclose the origin of the code. **As stated in the course policy document, concealing the origin of a piece of code is plagiarism**.

Use the Discussion Board for general questions whose answers can benefit you and everyone.

Goal: Implement a complete search engine.

In order to accommodate the various skill levels of students in this course, this project comes in four flavors:

(1) Librarian. In this flavor, there is no programming involved. Everything is to be done by hand using either pen and paper, or Excel. Warning: some parts of the project done in this manner are very tedious and time consuming. Be ready to spend many hours cataloguing words and documents.

(2) Information Analyst. In this flavor, there is some programming involved, but not much more advanced than what you already did so far. It's a mixture of the Text Processing project and stitching things together.

(3) Application Developer. In this flavor, there is programming to be done. You will interface with the powerful Lucene library, which already does everything necessary for a building an inverted index and searching it.

(4) Algorithms and Data Structures Developer. In this flavor, not only there is programming to be done, but everything is to be done by you, instead of using Lucene.

Milestones Overview

	Deadline	Goal	Deliverables	Contribution for score
#1	3/1	Produce an initial index for the corpus	Short report (no demo)	10%
#2	3/8	Develop a retrieval component	Short report (no demo)	10%
#3	3/15	Complete Search System	Code or artifacts + Demonstration	80%

Librarian

Programming skills required: none
Main challenges: process many documents and words by hand

Corpus: The pages of all ICS faculty, as reached from this link
http://www.ics.uci.edu/faculty/
Follow the links to each individual faculty page to get the contents. You don't need to follow any further links from those pages. Avoid misdirected pages.
Note: there is no programming involved, whatsoever. You are to look at the pages yourself and process them.

Index: Create an inverted index for all these pages. You can use Excel to store this index, or any other tool you see fit. You can also do it using a physical Rolodex. Whatever works!

The index should contain not just the pages where the words occur, but also the tfidf of the words in each page.

Also, words in bold should be treated as more important than non-bolded words. You can handle this as you want: create separate indexes, or add metadata about the words to the single index.

Extra credit will be given for ideas that improve the quality of the retrieval, so you may add more metadata to your index, if you think it will help improve the quality of the retrieval.

Search: you are the librarian. Others, like the TA, are the users of the Library. They will come to you and ask you to retrieve the best pages for a certain set of keywords. For example, give me all the pages related to artificial intelligence. At the time of the query, you will look up your index, perform some calculations (see ranking below) and give out the ranked list of pages that are relevant for the query.

Ranking: at the very least, your ranking formula should include tfidf scoring, but you should feel free to add additional components to this formula if you think they improve the retrieval.

Information Analyst

Programming skills required: Intro courses
Main challenges: light html parsing, read/write structured information from/to files.

Corpus: a small portion of the ICS web pages, simplified (download TBD soon)

Index: Create an inverted index for all these pages. You can use a database to store this index, or a simple file.

The index should contain not just the pages where the words occur, but also the tfidf of the words in each page.

Words in bold and in heading (h1, h2, h3) should be treated as more important than the other words. You can handle this as you want: create separate indexes, or add metadata about the words to the single index.

Extra credit will be given for ideas that improve the quality of the retrieval, so you may add more metadata to your index, if you think it will help improve the quality of the retrieval.

Search: your program should prompt the user for a query. This doesn't need to be a Web interface, it can be a console prompt. At the time of the query, your program will look up your index, perform some calculations (see ranking below) and give out the ranked list of pages that are relevant for the query.

Ranking: at the very least, your ranking formula should include tfidf scoring, but you should feel free to add additional components to this formula if you think they improve the retrieval.

Application Developer

Corpus: all ICS web pages (download TBD soon)

Programming skills required: Intermediate
Main challenges: full HTML parsing, understand the Lucene API.

Index: Create an inverted index for all these pages using the Lucene library or, if you prefer, its cousin, Elastic Search.

Words in bold, in headings (h1, h2, h3), and titles should be treated as more important than the other words.

Extra credit will be given for ideas that improve the quality of the retrieval, so you may add more metadata to your index, if you think it will help improve the quality of the retrieval.

Search: your program should prompt the user for a query. This doesn't need to be a Web interface, it can be a console prompt. At the time of the query, your program will look up your Lucene / Elastic Search index, perform some calculations (see ranking below) and give out the ranked list of pages that are relevant for the query.

Extra credit will be given if your search interface is a web interface.

Ranking: you are going to use the scoring formula of Lucene, so that formula is going to be more or less hidden from you. However, you should study how Lucene scores the documents wrt the queries, and adjust the several parameters of your index for better retrieval.

Note that if you use Lucene, you need to use Java for your program. Elastic Search has a REST API, so you can use any language for which there are bindings.

Algorithms and Data Structures Developer

Corpus: all ICS web pages (download TBD soon)

Programming skills required: advanced
Main challenges: full HTML parsing, design efficient data structures

Index: Create an inverted index for all these pages with data structures designed by you, and without using Lucene. Your index lookup during search should not be horribly slow, so pay attention to what kind of data structure you use that supports fast search. Your index should be stored in one or more files in the file system.

Words in bold, in headings (h1, h2, h3), and titles should be treated as more important than the other words.

Extra credit will be given for ideas that improve the quality of the retrieval, so you may add more metadata to your index, if you think it will help improve the quality of the retrieval. Suggestion: page rank.

Search: your program should prompt the user for a query. This doesn't need to be a Web interface, it can be a console prompt. At the time of the query, your program will look up your index, perform some calculations (see ranking below) and give out the ranked list of pages that are relevant for the query.

Extra credit will be given if your search interface is a web interface.

Ranking: at the very least, your ranking formula should include tfidf scoring, but you should feel free to add additional components to this formula if you think they improve the retrieval.

Milestone #1

Goal: Build an index

Deliverables: Submit a report (pdf) to with the following content: a table with assorted numbers pertaining to your index. It should have, at least the number of documents, the number of [unique] words, and the total size (in KB) of your index on disk.

Evaluation criteria:

- Did your report show up on time?
- Are the reported numbers plausible?

Milestone #2

Goal: Develop a retrieval component

At least the following queries should be used to test your retrieval:

1 – crista lopes
2 - machine learning
3 - ACM

Deliverables: Submit a report (pdf) to with the following content:

- the top 5 URLs for each of the queries above
- a picture of your search interface in action

Evaluation criteria:

- Did your report show up on time?
- Are the reported URLs plausible?

Milestone #3

Goal: complete search engine

Deliverables:

- Submit a zip file containing all the artifacts/programs you wrote for your search
- A live demonstration of your search engine

Evaluation criteria:

- Does your search engine work as expected of search engines?
- How general are the heuristics that you employed to improve the retrieval?
- How complete is the UI? (e.g. links to the actual pages, snippets, etc.)
- Do you demonstrate in-depth knowledge of how your search engine works? Are you able to answer detailed questions pertaining to any aspect of its implementation?

Additional criteria will be used for each of the 4 flavors of the project.