Project 3 – Search Engine

Due dates: 2/20 and 3/6

In this assignment you will be building a simple search engine based off of the concepts you have learned in the lectures so far. When working on this project, you will be working on achieving two milestones each with its deliverables and deadline. Bear in mind that when planning for milestone #1, you are also cognizant of the requirements of milestone #2. This is to ensure you are on the right track to completing this project successfully. When you will have completed milestone #2, you will be required to demonstrate the functioning of your search engines to the TAs (F2Fs).

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 Piazza to post your questions about this assignment so that the answers can help you and other students as well.

Goal: Implement a complete search engine.

Milestones Overview

Milestone	**Deadline**	**Goal**	**Deliverables**	**Score** (out of 60)
#1	2/20	Produce an initial index for the corpus and a basic retrieval component	Short report (no demo)	15 points
#2	3/6	Complete Search System	Code or artifacts + Demonstration	45 points
		Extra Credit	Extra Credit	6 points

PROJECT: SEARCH ENGINE

Corpus: all ICS web pages

We will provide you with the crawled data as a zip file (webpages.zip). This contains the downloaded content of the ICS web pages that were crawled by us. You are expected to build your search engine index off of this data.

Main challenges: Full HTML parsing, File/DB handling, handling user input (either using command line or desktop GUI application or web interface)

COMPONENT 1 - INDEX:

Create an inverted index for all the corpus given to you. You can either use a database to store your index (MongoDB, Redis, memcached are some examples) or you can store the index in a file. You are free to choose an approach here.

The index should store more than just a simple list of documents where the token occurs. At the very least, your index should store the TF-IDF of every term/document.

Sample Index:

> Note: This is a simplistic example provided for your understanding. Please do not consider this as the expected index format. A good inverted index will store more information than this.
>
> Index Structure: token – docId1, tf-idf1 ; docId2, tf-idf2
>
> Example: informatics – doc_1, 5 ; doc_2, 10 ; doc_3, 7

You are encouraged to come up with heuristics that make sense and will help in retrieving relevant search results. For e.g. - words in bold and in heading (h1, h2, h3) could be treated as more important than the other words. These are useful metadata that could be added to your inverted index data.

Optional:

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. For this, instead of storing a simple TF-IDF count for every page, you can store more information related to the page (e.g. position of the words in the page). To store this information, you need to design your index in such a way that it can store and retrieve all this metadata efficiently. Your index lookup during search should not be horribly slow, so pay attention to the structure of your index

COMPONENT 2 – SEARCH AND RETRIEVE:

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.

Optional:

Extra credit will be given if your search interface has a GUI.

COMPONENT 3 - RANKING:

At the very least, your ranking formula should include tf-idf scoring, but you should feel free to add additional components to this formula if you think they improve the retrieval.

Optional:
Extra credit will be given if your ranking formula includes parameters other than tf-idf

---

Milestone #1

Goal: Build an index and a basic retrieval component

By basic retrieval component; we mean that at this point you just need to be able to query your index for links (The query can be as simple as single word at this point).

These links do not need to be accurate/ranked. We will cover ranking in the next milestone.

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

1 – Informatics

2 – Mondego

3 – Irvine

4 – artificial intelligence

5 – computer science

Note: query 4 and 5 are for milestone #2

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

1. 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.
2. URLs retrieved for each of the queries above

Evaluation criteria:

- Was the report submitted on time?
- Are the reported numbers plausible?

- Are the reported URLs plausible?

Milestone #2

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 program 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 Information:

Understanding the data dump:

In Assignment-2, crawlers of all the groups collectively crawled 37,497 URLs. We collected these URLS and are providing them to you as 'webpages_clean.zip' file. This zip file contains the following:

1. bookkeeping.json
2. bookkeeping.tsv
3. Folders 0 to 74

Folders:

The 37,497 URLs are organized into 75 folders, each folder having 500 files. Every file has the extracted HTML source code of a particular URL.

Bookeeping files:
bookkeeping.json and bookkeeping.tsv are two different formats of the same file. These files maintain a list of all the URLs that have been crawled. Every URL has an identifier associated

with it. This identifier helps locate the HTML code of the URL. The identifier is of the format: “folder_number/file_number”
For example, consider the entry on line 13 of bookkeeping.json:

"0/108": "vision.ics.uci.edu/papers/RamananBK_ICCV_2007"

This means that the HTML code extracted for the link "vision.ics.uci.edu/papers/Ramanan BK_ICCV_2007" is located at folder 0, file number 108.

Broken HTML:

The HTML source code of the URLs may not be well formed. This means that the code may not necessarily have a pair of opening and closing tags. For example, there might be an open
tag but the associated closing tag might be missing. The HTML parsers that you will use to parse the documents should be able to handle broken HTML. Hence, as mentioned above, while selecting the parser for your project, please ensure that it can handle broken HTML.

Use of libraries:

It is strictly not allowed to use libraries that perform the entire task of index creation or ranking for you. Hence, libraries such as Lucene or Elastic Search are not allowed.

You may use libraries that help you achieve specific tasks. For example, you can use a tokenizer such as NLTK to tokenize your content.

The HTML files

- A Zip file that contains crawl-able HTML files which you may parse/process for extracting tokens.
- The HTML files have been organized and stored in numbered directories. The file names are numbers as well.
- The bookkeeping.json and bookkeeping.tsv files represent the index of all the HTML files.
- The key value of the json file is essentially the relative file path of the HTML content. The value is the web URL of the HTML content.
- Do not confuse bookkeeping with the inverted index. It simply provides you a means to access the crawl-able HTMLs programmatically. The key values in
- bookkeeping can also be used to uniquely identify the files. This will be useful when you need to retrieve the web page and the content while displaying your search engine results.

Building the inverted index

- Now that you have been provided the HTML files to index. You may build your inverted index off of them.
- As most of you may already know, the inverted index is simply a map with the token as a key and a list of its corresponding postings.
- A posting is nothing but the representation of the token's occurrence in a document.
- The posting would typically (not limited to) contain the following info (you are encouraged to think of other attributes that you could add to the index) :
- The document name/id the token was found in.
- The word frequency.
- Indices of occurrence within the document
- Tf-idf score etc

Inverted Index

- When designing your inverted index, you will think about the structure of your posting first.
- You would normally begin by implementing the code to calculate/fetch the elements which will constitute your posting.
- Modularize. For eg:- If you're using python, use scripts that will perform a function or a set of closely related functions. This helps in keeping track of your progress, debugging, and also dividing work amongst teammates if you're in a group.
- You are free to choose any database system to store your inverted index.
- Some possible options - Redis, MongoDB, memcached, MySQL etc.
- Pro-tip : If you have a hard time choosing between the database systems. Read about their performance and learning curves of the libraries available with the language of your choice.

Search and Retrieve

- Once you have built the inverted index, you are ready to test document retrieval with queries.
- At the very least, the documents retrieved should be returned based on tf-idf scoring. This can be done using
- the cosine similarity method. Feel free to use a library to compute cosine similarity once you have the term frequencies and inverse document frequencies.
- You may add other weighting/scoring mechanisms to help refine the search results.

Tips

- Start as early as possible (esp. if you have slow computers/laptops).
- When asking questions on Piazza, please be as specific as possible. We cannot help you with very broad/open-ended questions as this assignment is pretty free-form.
- Read the other assignment documents carefully. If there are mistakes/ambiguities, please bring it to our notice.
- Google and Debugging are your best friends!