Table of Contents

Objectives

The aim of this project is to develop a simple Crawler- based search engine that demonstrates the main features of a search engine (web crawling, indexing and ranking) and the interaction between them. Also it is intended to enhance your Java programming skills.

Search Engine Modules

Web Crawler [20%]

The web crawler is a software agent that collects documents from the web. The crawler starts with a list of URL addresses (seed set). It downloads the documents identified by these URLs and extracts hyper-links from them. The extracted URLs are added to the list of URLs to be downloaded. Thus, web crawling is a recursive process.

Care should be taken when implementing web crawlers. At minimum, you have to take care of the following issues:

- The crawler must not visit the same URL more than once.
- The crawler can only crawl documents of specific types (HTML is sufficient for the project).
- The crawler must maintain its state so that it can, if interrupted, be started again to crawl the documents on the list without revisiting documents that have been previously downloaded.
- Some web administrators choose to exclude some pages from the search such as their web pages check for robot.txt
- Provide a multi-threaded crawler implementation where the user can control the number of threads before starting the crawler.
- The choice of the seed set can vary from one design to another.
- When the crawler finishes one iteration by reaching stopping criteria, it restarts again. The frequency of crawling is an important design aspect of the web crawler. Some sites will be visited more often than others. You have to set some criteria to the sites. In another words, during re-crawling, you don't have to repeat all the sites again.
- The number of crawled pages must be at least 5000 pages (for the sake of this project).
- The crawler is an independent program or process, and different from the Indexer.

Indexer [30%]

The output of web crawling process is a set of downloaded HTML documents. To respond to user queries fast enough, the contents of these documents have to be indexed in a data structure that stores the words contained in each document and their importance (e.g. whether they are in the title, in a header or in plain text). This data structure has to satisfy the following properties:

- Persistence: The index has to be maintained in secondary storage. You can implement your own file structure or use a database.
- Fast Retrieval: The index must be optimized for responding to queries like:
    - The set of documents containing a specific word (or set of words)
    - The set of words contained in a specific document.
- Incremental Update: It must be possible to update an existing index with a set of newly crawled HTML documents.
- When designing the Indexer, consider how you will store your result by looking ahead on Ranker and Searching.

Query Processor [10%]

This module receives search queries, performs necessary preprocessing and searches the index for relevant documents. Retrieve documents containing words that share the same stem with those in the search query. For example, the search query “travel” should match (with lower degree) the words “traveler”, “traveling” … etc.

Phrase Searching [5%]

Search engines will generally search for words as phrases when quotation marks are placed around the phrase.

Ranker [20%]

The ranker module sorts documents based on their popularity and relevance to the search query.

1. **Relevance:**
   Relevance is a relation between the query words and the result page and could be calculated in several ways such as tf-idf of the query word in the result page or simply whether the query word appeared in the title, heading, or body. And then you aggregate the scores from all query words to produce the final page relevance score.

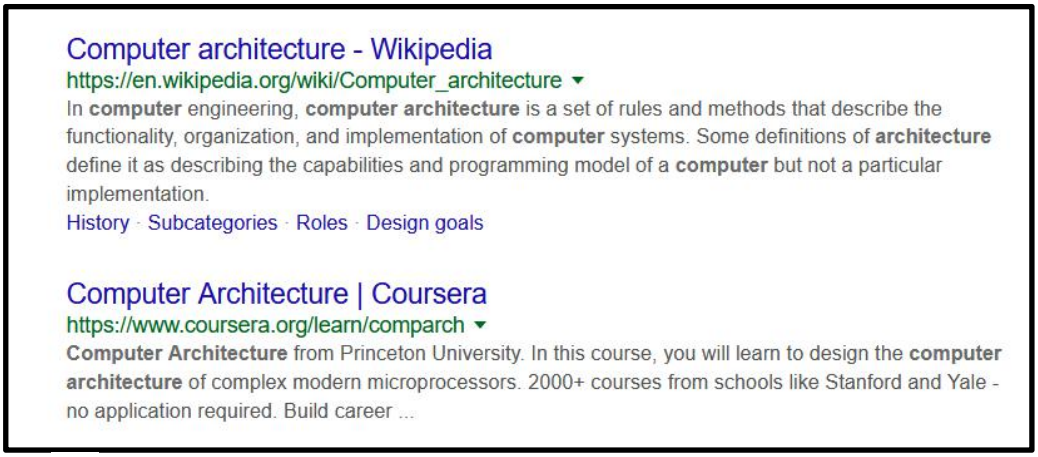
2. **Popularity:**
   Popularity is a measure for the importance of any web page regardless the requested query. You can use PageRank algorithm (as explained in the lecture) or other ranking algorithms to calculate each page popularity .

Grading criteria (20%): 5% for efficiency, 10% for correctness/understanding, 5% for implementation

Android/Web Interface [15%]

You have to implement a web interface or a mobile interface for your search engine. It is your choice.

- This interface receives user queries and displays the resulting pages returned by the engine
- The result appears with **snippets** of the text containing queries words. The output should look like google/bing's results page



- Pagination of results (i.e. if you got 200 results, they should appear on 20 pages, each page with 10 results)
- Add suggestion mechanism that stores queries submitted by all users. As the user types a new query, your web application should suggest popular completions to that query using some interactive mechanism such as AJAX.
- The web interface should display suggestions while the user is typing their search query. For example, if the user typed 'World', then a list of suggestions should be displayed **'World Cup'**, **'World Health Organization'**, **'World War'**, **'World Meter'**, .. etc.
- If your web interface is having any problem showing results, you should make sure that the query retrieval is working correctly by displaying the results in a file/console to make sure that everything is working except for the interface.

Grading criteria(15%): 5% for neatness, 5% for correctness/implementation, 5% for suggestion mechanism

Additional Features (Mandatory):

1. *Image Search:*

The user can search the web for **images** on a given search query. For example, if the user used this feature and searched for "World Cup", then the ranker should return the most relevant images to this query.

2. *Relevance Score:*

Your relevance score has to include the following aside from **word similarity**:

- **Geographic location of the user:** You should increase the score of web pages related to the user's location. A web page(s) can be related to certain location(s) in many ways (server location, company's location, visitors' location, URL extension, etc). It will be sufficient to consider one of these ways to score the geographic relevance of web page. For example, a web page having the .uk extension is more relevant to users in UK, a web page having the .cn extension is more relevant to users in China, and so on.

- **How recent is the web page?** A web page's score should increase because it was published recently. It should be noted that some websites do not mention the web page's creation date in the HTML

source, so your scoring equation should handle the case of no dates. We don't require you to extract the published date from the text of the web page because this could get messy. We only require you to extract a date from a web page if the '***datePublished***' HTML property or ***pubdate*** HTML attribute are present in the HTML document of the crawled web page.

3. Personalized Search:

Your search engine should perform a personalized search for its user. A personalized search is a search that is tailored to the user's interests by taking into consideration some information about the user such as the user's previous search queries, frequently visited/clicked-on web pages, frequently visited domain names, etc. You can think of many design options for personalized search.

For example, when a Wikipedia-obsessed user searches for a query (e.g. World Health Organization), then a personalized search engine should rank the results returned by the ranker in a way that favors Wikipedia over other websites, and so on.

4. Trends:

Your query processor should keep track of search trends. We need to view the trends about the **most searched persons** in each country. We recommend that you use an NLP library like the Stanford CoreNLP library for this feature. The interface of this feature should be a separate page in which we can view the top 10 persons as a histogram after we select the country we are examining its trends.

For this feature, you will have to include in the homepage of the search engine a drop-down list of all the world countries to mimic the behaviour of searching within another country. (or you can use a fake GPS and let the server auto-locate the user).

5. Voice Recognition Search:

The user can use a voice query instead of a typed query. You can use NLP Libraries and APIs (such as the Stanford CoreNLPlibrary) to recognize and understand a voice query, transform it into textual query and perform the search accordingly.

6. Performance Analysis Module:

This module should test the different modules you implemented and show relevant information to analyze the performance. We require that this module makes you able to answer the following questions:

- How many simultaneous search requests can your solution handle?
- How is the latency of your solution affected by the number of simultaneous search requests?
- How is the search request latency of your solution affected by the number of web pages crawled?
- How is the search request latency of your solution affected by the size of the index table?
- How is the search request latency of your solution affected by the ranking process?

Notes:

1. In the discussion, you will have to explain your results (i.e. what the bottlenecks are, why the produced results are (not) normal, ….). You will also have to say how your solution could be improved.

2. We are aware that the machines running your solution affect the produced result; so don't worry.

Implementation and Deliverables

Deadlines

The deadline for submitting the project completely functioning will be on 3rd June 2020 11:59 PM.

You will have to submit a link to your github or gitless source project including:

- Your code files
- A readme.txt, explaining how to run your code
- A members.txt containing the names and IDs of each student in the group and whether you're semester or credit
- A PDF file containing any algorithms you've used

Also provide a zipped folder containing the same material, Name the zipped folder [Semester or Credit][Team Number].zip

Teams

- Work in groups of 3~4.

Implementation

- Implementation is done mainly in **Java**.
- It is your responsibility to select the best technique and tool that enhances the performance of your project.

Libraries and Packages Regulations

You can use a library only if you followed **all** of the points below:

- You **can**'t use a library that do the whole module functionality (i.e. you can't use a library to do the crawler as whole) if you are in doubt ask.
- You are responsible for the library accuracy. (If it does a bad job, then it is your responsibility).
- You should understand how the library works.
- You can use any database management system **BUT** you should be the one creating the scheme and adding the data. Don't use a database to do the whole indexing (i.e. don't give it the document and let it do the indexing for you, however you can get a bonus if you compared your implementation with the database automatic indexing)

Evaluation and Grading Criteria

- The project is graded as a whole and the discussion decides what's the grade of each student (there's no piggybacking)
    - 50% of the project grade will be on requirement completeness
    - 40% of the project grade will be on understanding how everything works in your system
    - 10% of the project grade will be on code organization, neatness, using source control and naming convention.
- Any delay in any phase will be penalized by losing 10% of the grade for each late day
- Code must be original, and may not be copied or shared from any other source, except as provided by the class instructor
- Plagiarism will be very strictly punished.
- Note: A plagiarized project means ZERO in the project and hence failing the course. Please, don't put yourself in such situation.

~~~~~ Good Luck ~~~~~
~~~~~