COMP4321 Project Description

1 Description

You are required to develop a web-based search engine with the following functions:

1. A spider (or called a crawler) function to fetch pages recursively from a given web site:
   - Given a starting URL and the number of pages to be indexed, recursively fetch the required number of pages from the given site into the local system using a breadth-first strategy (BFS).
   - For each page fetched into the local system, extract all the hyperlinks so that the spider can recursively process more pages, and proceed to the indexing functions in Step 2.
   - Build a file structure containing the parent/child link relation. As noted elsewhere, every URL is represented internally as a page-ID, so the file structure should be able to return the page-IDs of the children pages given the page-ID of the parent page and vice versa.
   - Before a page is fetched into the system, it must perform several checks:
     - If the URL doesn't exist in the inverted index, go ahead to retrieve the URL
     - If the URL already exists in the index but the last modification date of the URL is later than that recorded in the index, go ahead to retrieve the URL; otherwise, ignore the URL
     - To handle cyclic links gracefully
     - We may run your crawler several times with some web page modification in between to test your crawler robustness.
   - **Resource**: The **HTML parser** library from http://htmlparser.sourceforge.net/ provides the basic functions to fetch the we bpages and to extract the keywords and links from the web pages; notice that the HTML parser is a very large library, but we need no more than a few basic functions from it.
2. An indexer which extracts keywords from a page and inserts them into an inverted file
   - The indexer first removes all stop words from the file; a dictionary of stop words will be provided
   - It then transforms words into stems using the Porter's algorithm
   - It inserts the stems into the two inverted files:
     - all stems extracted from the page body, together with all statistical information needed to support the vector space model (i.e., no need to support Boolean operations), are inserted into one inverted file
     - all stems extracted from the page title are inserted into another inverted file
   - The indexes must be able to support phrase search such as "hong kong" in page titles and page bodies.
   - **Resource**: A Java Implementation of Porter's algorithm is available in lab 3. The JDBM library from http://jdbm.sourceforge.net/ is suggested to be used to create and manipulate the file structures for storing the inverted file and other file structures needed.
3. A retrieval function (or called the **search engine**) that compares a list of query terms against the inverted file and returns the top documents, up to a maximum of 50, to the user in a ranked order according to the vector space model. As noted about, phrase must be supported, e.g., "hong kong" universities
   - Term weighting formula is based on tfxidf/max(tf) and document similarity is based on cosine similarity measure.

- Derive and implement a mechanism to favor matches in title. For example, a match in the title would significantly boost the rank of a page

4. A web interface that accepts a user query in a text box, submits the query to the search engine, and displays the returned results to the user

   - Since only the vector space model is required, you don't need to support logical operators; the query is just a list of keywords that allows phrases to be specified in double quotes
   - The results are ranked in descending order of the document scores
   - Each item returned is displayed in the following format:

score	page title
	url
	last modification date, size of page
	keyword 1 freq 1; keyword 2 freq 2; ...
	Parent link 1
	Parent link 2
	... ...
	Child link 1
	Child link 2
	... ...
score	page title
	... ...

     – The title and URL are hyperlinked to the actual page on the remote server
     – The list of keywords displays up to 5 most frequent stemmed keywords (excluding stop words) in the page together with their occurrence frequencies
     – Some pages do not contain the last modified date field and the size field in their headers. In these cases, you can consider the date field to be the last modified date and consider the length of the content (i.e. directly count the number of characters) to be the size of the page.
   - While the interface doesn't have to be fancy, each result item should be format with clarity
   - All of the results (up to 50 web pages) can be displayed on one page
   - In order to interface your search engine with the web interface, you need to write a JSP page to pass the query string to the search engine.

2 Submissions

2.1 Phase 1

- It worths 10% of the course marks. However, if you fail to submit the phase 1, you will get zero marks for you entire project.
- You need to:
  – implement a spider (integrated with an indexer) for fetching (using BFS) and indexing
  – index 30 pages from `https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm` or `https://comp4321-hkust.github.io/testpages/testpage.htm` (backup website)
  – implement a test program which read data from the jdbm and outputs a plain-text file named spider_result.txt. The format of the spider_result.txt file should be as follows:

  - Page title
    URL
    Last modification date, size of page
    Keyword1 freq1; Keyword2 freq2; Keyword3 freq3; ... ...
    Child Link1
    Child Link2 ... ...
    ———————————— (The separator line should be a line of hyphens, i.e. -)
  - Page title
    URL
    Last modification date, size of page
    Keyword1 freq1; Keyword2 freq2; Keyword3 freq3; ... ...
    Child Link1
    Child Link2 ... ...
    ... ...
    ... ...
  - The list of keywords/child links display up to 10, and there is no requirement for the order.

- You should submit:

  1. a document containing the design of the jdbm database scheme of the indexer. All supporting databases should be defined, for example, forward and inverted indexes, mapping tables for URL <=> page ID and word <=> word ID conversion. The jdbm database schema depends on the functions implemented. You should include an explanation on your design.
  2. the source codes of the spider and the test program
  3. a readme.txt file containing the instructions to build the spider and the test program, and how to execute them.
  4. the db file(s) which contain the indexed 30 pages starting from `https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm` or `https://comp4321-hkust.github.io/testpages/testpage.htm` (backup website)
  5. spider_result.txt which is the output of test program

- Zip the files and submit via Canvas. The assignment name is Phase1.

2.2 Final Submission

- It worths 30% of the course marks.
- You need to:
  - implement the search engine and the web interface. The user inputs queries the search engine through the web interface. The search engine compares the query terms against the indexed terms in jdbm and returns the top documents to the user through the web interface.
  - index 300 pages starting from `https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm` or `https://comp4321-hkust.github.io/testpages/testpage.htm` (backup website)
  - provide materials to TA to replicate your results
  - Make a video to demonstrate your project. The TAs may ask you to demonstrate your project through Zoom if the submitted files cannot be run.
- You need to submit:
  1. the source codes of the spider, indexer, search engine and the web interface (DO NOT submit the db files)
  2. a document (around 8-10 pages long) containing
     - Overall design of the system
     - The file structures used in the index database
     - Algorithms used (including the mechanism for favoring title matches, BFS, etc.)
     - Installation procedure (it could be as simple as "Type make in the project directory")

- Highlight of features beyond the required specification
- Testing of the functions implemented; include screenshots if applicable in the report
- Conclusion: What are the strengths and weaknesses of your systems; what you would have done differently if you could re-implement the whole system; what would be the interesting features to add to your system, etc.
- Contribution: The contribution that each member of the team made to the project, e.g. Andy: 33.3% (mainly responsible for crawler building, etc.), Bob: 33.3% (mainly responsible for search engine building, etc.), Cindy: 33.3% (mainly responsible for report writing, etc.).

3. a readme.txt file containing the instructions to run your codes and replicate your results.

- Zip the files and submit via Canvas. The assignment name is FinalPhase.

3 Marking Scheme

	Phase 1 (10)	Final Submission (20)
Correctness and completeness	70%	60%
Program design and programming style	10%	10%
Documentation	20%	10%
Basic user interface design	0%	5%
Demonstration (to TA)	0%	15%
Bonus	0%	10%
TOTAL	**100%**	**110%**

4 Bonus

You can enhance your system beyond what is required in the project description to get a maximum of 10% bonus. However, the final decision on whether your additional work is worth any bonus is completely up to the TA and my decision. The following is a couple of examples that may be considered for bonus. Whether you will get any bonus, and if so, how much, depends on how much effort is required to do the extra work and how well is the implementation.

1. Implementation of the relevance feedback feature "get similar pages" (cf. Google); clicking the "get similar pages" button will extract the top 5 most frequent keywords (excluding stop words) from the page; rewrite the original query and submit the revised query for a new search.
2. Allow the user to see a list of all (stemmed) keywords indexed in your database, browse through them, and select the keywords he/she is interested in, and then submit them as a vector-space query to your search engine.
3. A user-friendly interface for all of the functions (e.g., making use of DHTML, AJAX, application-based interface)
4. Keep track of query history (need application-based interface or cookies, etc.) and allow users to view the result of a previous query (or operate on the results, e.g., merging the results of two previous queries or search within the result of a previous query).
5. Many other features that you can observe from existing commercial search engines.
6. Consider links in result ranking (e.g., PageRank).
7. Exceedingly good speed by using special implementation techniques.

5 Some Suggestions And Requirements

1. You are free to choose any programming language you like to do your project, but you should ask the TAs for approval if you want to include any APIs/packages/libraries. In addition, you should bear your own risk if you choose another language, and you should not expect us to help you with debugging. You should also comment on the code snippets that realize the algorithms described

in the report for grading. Besides, it is recommended to use Java with version 17 or higher and Tomcat with version 10.1.18 as we are going to test your code using those versions. And if you use another programming language, please state in the README file the versions of the softwares used to build the project, e.g. node 14.15, npm 10.4.0 for JavaScript, python 3.10 for Python, cmake 3.23, cpp standard 17 for Cpp, etc., and the precise commands to run it.

2. You are allowed to use other packages about database, parser, tokenizer, etc, to help you implement your project. Just be sure to include them clearly in the readme.txt file so that TAs are able to replicate your results by following your instructions. However, you MUST implement the key components in your project by yourself, WITHOUT using any external packages. For example, using a PageRank algorithm from a library is strictly prohibited. Please consult TAs if you are not sure any package is allowed. Your final mark will be deducted if you are found to use packages inappropriately.

3. You are suggested to use the stopword list (stopwords.txt) in lab 3.

4. Speed is not essential but must be reasonable in that you program (spider or search) should run at the average speed (i.e., if your program takes 5 times more than your friend's, then there must be something wrong with your design, or your friend's). If you had implemented special techniques that can speed up indexing and search (e.g., by using a special index design and/or intelligent caching), you can highlight it in your report and to the TA for qualifying bonus points. If your system is judged to be slower than normal, you should be able to give an explanation to avoid penalty.

5. If you are so unfortunate that you cannot finish the phases before deadlines, you should still submit the parts that you have done so that the TA can give partial marks to you. I won't be harsh in grading as I realize that you have lots of things to do. However, don't use this as an excuse for copying from other or from the old projects. If you are so unlucky that I discover you copy from other, you and the one who gives you the source would get ZERO MARK!