Homework: Google-lite *(check duedate on lcm.liacs.nl !)*

Submit a zip file on the LML Course Manager containing
- Journal.pdf
- your source code and documentation - *it is important to show screenshots of your program working for problem (2)*

Feel free to ask me questions by email. Class notes on the inverted/reverse index are up on the LML Course Manager.

GOAL: So far we have been developing a *basic* webspider. Here we create Google-lite - a functional web search engine based loosely on the original Google. Grades are based on difficulty level and quality of code. *Note that you are allowed to rewrite the code to your preference using C or C++ (html/php are used for the browser interface.* It must compile using *make* and have the functionality described below. At this point I assume you know what you are doing so be careful regarding downloading from the Web.

(1) Duplicate Detection (low to medium difficulty)

The main goal of this problem is to eliminate duplicates in your webcrawler. You have several options of varying difficulty level
- low difficulty - using string functions to check for duplicates. Only append a link if it does not exist in q.
- low-medium difficulty - build a binary tree to check for duplicates
- medium difficulty - convert the webcrawler to using linked-lists and use a binary tree to check for duplicates

In your answer, please mention which choice you made and whether you think your implementation was efficient.

(2) Inverted Index Web Searching

(students can choose for medium or high difficulty)
Recall that Google created several inverted indices. We will also create them. There are two different challenge levels one can take with this problem determined by either **medium or high difficulty**

Medium Difficulty

For each link that we added to q, parse it for the constituent words and create an inverted index under the directory "**webindex**" where each word is a file and the contents of the file are the weblinks that contain the word. [please see the class notes for more details]

i.e. http://www.liacs.nl/masters-education should be parsed into the following words

www liacs nl masters education

Note that a handy function for appending text to a file is fprintf:
FILE *fp; if(fp = fopen(myword,"a") != 0);fprintf(fp,"%s\n",myurl);fclose(fp);
where myword and myurl are strings (must end with '\0') and
"a" means append to end of file. The file is created if it does not exist.

Create a command line program called **webquery** that takes as input a single keyword query and prints a list of URLS which are relevant to the query based on the inverted indices from webindex.

Use the web language (php, perl, etc.) of your choice and create a working web search page (like Google) which shows a list of relevant URLs to the query. Note that for the purposes of this exercise, it is fine to simply have your web language (e.g. php) call the webquery function and return the output. (An example is given in file: **webinterface.mongoose.zip**)

Web Indices

Create the following:

(note: you should use docIDs for at least the LinkIndex. It is your choice if you use them for all of the URLs in indices such as Titleindex, Pageindex, etc.)
WebIndex: For each link that we added to q, parse it for the constituent words and create an inverted index under the directory "**webindex**" where each word is a file and the contents of the file are the weblinks that contain the word. [please see the class notes for more details]

i.e. http://www.liacs.nl/masters-education should be parsed into the following words

www liacs nl masters education

TitleIndex: Extract the title information (a simple example is given in the zip file: **title.zip**). For each link that we added to q, parse the title for the constituent words and create an inverted index under the directory "**titleindex**" where each word is a file and the contents of the file are the weblinks that contain the word.

LinkIndex: For each webpage we download, we create an inverted index under the directory "**linkindex**" where each URL in the webpage becomes a file and the contents of the file are the weblinks that point to webpages which contain the URL. Due to file creation problems, the preferred option is to use a docID (convert URL to MD5, CRC64, etc.) for the filename. To be precise, each file contains the weblinks whose webpage contains links that point to the URL corresponding to the filename. (ask me for help if this is confusing)

PageIndex: For each html webpage parse it for the constituent words (not html tags) and create an inverted index under the directory "**pageindex**" where each word is a file and the contents of the file are the weblinks that point to the webpage that contain the word.

Repository: Save the webpages in the directory "**repository**"

Create a command line program called **webquery** that takes as input a single keyword query and prints a list of URLS which are relevant to the query based on the inverted indices as described below in the section **Relevance Ranking**:

Google assumes that webpages which have query terms in the title are the most relevant followed by query terms in the URL followed by query terms in the webpage.

Relevance Ranking: Assume that we have the complete list of URLs, **L**, which are referred to through at least one of the inverted indices. For each URL in **L**, initialize its relevance score to 1. If the URL also came from titleindex then increase the score by 16. If the URL also came from webindex then increase the score by 4. Note: There is always a subjective aspect to the weight scores because relevance always is determined by human subjective evaluation. You should feel free to change the weights of the scores. For each link from linkindex which points to the URL, increase the score by 1. Sort the list by relevance score.

Use the web language (php, perl, etc.) of your choice and create a working web search page (like Google). Note that for the purposes of this exercise, you should have your web language (e.g. php) call the webquery function and return the output. (An example is given in file: **webinterface.mongoose.zip**) Each entry in the results should look roughly like Google:

Leiden University
www.leiden.edu
On 10 May 1946 Leiden University awarded Winston Churchill an Honorary Doctorate in Law. The entire city of Leiden came out onto the streets to welcome the ...

Images: Medium Difficulty

Also, implement text queries for images (like Google images). Place the image links in **ImageIndex**. Use at least the title of the webpage and the "alt" text for indexing. Show the results in a grid and link the image results to the original image.

[optional]: High Difficulty

Implement a search by similar color by adding a link to each image result underneath the image that says "similar images by color". This means that you will need to download each image from your index, calculate the color histogram, and place the color histogram in an index.

Feel free to widen the scope of the webcrawler beyond leidenuniv.nl but please be careful. Run the webcrawler to download at least 2,000 html pages. In Journal.pdf, mention what parts you were able to get working and include a few screenshots of your web search engine page showing results of several queries. Also remember to mention exactly what you wrote vs. using other libraries.