

# Project 3: Search Engine

CSC 1351, Spring 2006

Due: 25 March 2006

Design and implement a simple search engine. The basics for reading the contents of a web page are in the Project 15.1 description on p. 573 in the textbook.

Write a class `Main.Main` that takes four arguments on the command line, the maximal link length, the maximal number of pages, a search string, and the URL of the start page. E.g., if your program is run as

```
java main.Main 5 100 Baumgartner http://www.csc.lsu.edu/
```

it should find my home page.

Your search engine should start collecting pages at the URL provided on the command line. Any links found on this page will be traversed and the pages pointed to by `<a href="...">` tags will be read as well, and so on. Your search engine will keep track of the number of references to a page and will then sort the pages in decreasing reference count order. It prints the first URL in this sorted order whose `<title>` contains the search string.

The arguments provided on the command line are used as follows:

**Maximal link length** The maximal number of links to traverse from the start page to any page included in your search.

**Maximal number of pages** The maximal number of pages to collect.

**Search string** The string to be searched in the `<title>` fields of the web pages in decreasing reference count order.

**Start page URL** The URL for the page on which to start the search.

## Data Collection

Your program should traverse the web pages in *breadth-first order*. You will maintain an array of pages to traverse. You start out by adding the start page to this array. You would then first visit (scan) the start page, extract the `<title>` and record it, and then you add all the URLs found in `<a href="...">` tags to your list of pages. When you finished reading the current page, you go on to visit the next URL in your array that hasn't been visited yet.

For keeping track of the order in which the pages should be visited, you can maintain an array of URLs (Strings) together with two indices into the array, the index of the web page to visit next and the index where to add the next URL found on the current page. This array together with the two indices is an implementation of a *queue* data structure.

Each time a web page is visited, your program should print the URL to `Logger.global` (see the material on logging in Chapter 10).

## Sorting and Searching

Your program will maintain a second array of web page descriptor objects. For each web page, you will record the URL (as a String), the title (from the `<title>` tag), and how often this web page was referenced by other web pages.

While you are collecting data, keep this array sorted alphabetically by the URL. When you encounter the `<title>` of a page, you search this array for the descriptor of the page and store the title. When you encounter an `<a href="...">` link, you search for the URL in this array. If you have seen it before, you increment the *reference count*. If you have not seen it yet, you insert a descriptor for this page into the sorted array.

For searching this array for a given URL, it would be most efficient to use *binary search*, but you can also use *linear search* (i.e., a simple for-loop).

When you finished collecting the data, resort the array *lexicographically*. Sort it by decreasing reference count. For two pages with the same reference count, sort them by increasing URL. Then using linear search find the first web page whose title contains the search string. Print the URL of the page you found or print a message saying that you did not find any page with the given search string in the title.

## Google

Our search engine works similar to the Google search engine. One of the innovations of Google was to sort the search result by reference count so that the most frequently referenced pages are at the top of the list of search results.

Of course, what makes the Google search engine much more complex is that it searches the entire page, as well as PDF files and other documents, instead of just the title and that it can have more complicated search expressions. Also, making the search efficient for the huge volume of data that Google collects requires much more complicated data structures.

## Administrative Stuff

Put your files in the directory `~/prog3` in your `cs1351xx` account on `byte` and submit it using

```
~cs1351c/bin/r_copy 3
```

Your main program should be in class `main.Main`. Split your program into two or more packages. Also submit a `README` file in which you describe how you designed your code, what other logging information you print, or anything else you want the grader to know.