Computer Science II — Homework 7 — Web Crawling and Searching

Preliminaries

This assignment is due Monday March 31 at 11:59:59pm. It is worth 120 points toward your homework grade.

Please adhere to the following submission instructions: Place all of your source code files and a `readme.txt` file in a folder named hw7 (no spaces, no uppercase, not `hw7 submit`, not `H7`). Make sure none of your file names include spaces. Do not include any extra files. Zip the folder (not just the files in the folder), so that when it unzips a new folder is created, and submit it through the course web page.

Overview

This homework explores several aspects of web crawling and web searching using simplified file formats and hyperlinks. Effective use of `std::map` is important to making the project manageable. Many techniques and code from lecture and lab may be adapted for use in this assignment. You are encouraged to use whatever you wish.

Your program will start with a single file ("page") and use hyperlinks found within the file to "crawl" to additional pages, continuing through additional pages until no new pages may be reached. In processing each file, it will extract words and pairs of words as the basis for future search queries. Once the program has crawled to all reachable files, it will switch to search mode. In this mode it will read search requests from a file and use its crawling results to display rank-ordered lists of pages. Details of each of the steps are covered in what follows.

Command-Line

The command-line will be executed as follows:

```
crawl init-file search-file results
```

where `init-file` is the page from which to start crawling, `search-file` is the file containing the "web" search queries to be processed after crawling is completed, and `results` is the output file.

Examples will be available on the course web site. When you download the examples, you will notice that many files (pages) are provided. One of these files will be given on the command-line as the `init-file`. The others (except for the search file) will only be opened and read if they are reached by the crawling process.

Crawling

There are two aspects to crawling: parsing an individual page and using hyperlinks to find additional pages.

The individual pages will be text files and should be broken up into a sequence of lower-case words. Each word is an uninterrupted sequence of alphabetic characters, digits or `'-'` characters, with alphabetic characters converted to lower case. The only exception is hyperlinks. A hyperlink is specified by the sequence

```
xref = "file-name"
```

where `file-name` is the name of the file containing a web page. When you read the `file-name` do not change it in any way — do not convert to lower-case and do not remove white-space. There may or may not be a space before and after the `=` sign and the `xref` and the file name may even be on separate lines. There will not be syntax errors in the files. When reading the file, eliminate words that have three or fewer characters.

Once your program has finished reading a file it will have a vector of words and a sequence of hyperlinked files (pages). The program should repeat the process of opening and reading files on the hyperlinked pages, continuing repeatedly until no new pages are reached. For each page, the program should keep track of how many other pages link to it. You may **not** assume a hyperlinked page (the associated file) exists.

You will notice that the hyperlink search described above is very similar to the breadth-first search from HW 6. In fact it is the same algorithm, realized in a very different way here. In particular, there is no need to create and use pointers.

Searching

The second input file, to be read only after the crawling process is finished, contains a list of search queries, one query per line. Each query will be a pair of words, using the same definition of words as above. In this input there will be no "non-word" characters in the input file and, more broadly, there will be no errors in the search input file. This means that the file is trivial to parse. Searching is not case-sensitive, however, so all alphabetic characters should be converted to lower case.

In response to each query your program should output the names of up to ten rank-ordered pages. To do this, we need the notion of two words "appearing together" on a page. Two words "appear together" if they are within three word positions or each other — in other words, after removal of short words as above, the difference in the index between the two words is three or less. For example, in the word sequence

```
apple a candy bunny chocolate spring it the bunny
```

The pairs of words that "appear together" are

- `apple` and `bunny`
- `apple` and `candy`
- `apple` and `chocolate`
- `bunny` and `candy`
- `bunny` and `chocolate` (twice)
- `bunny` and `spring`
- `candy` and `chocolate`
- `candy` and `spring`
- `chocolate` and `spring`

The fact that `bunny` appears together with itself should be ignored. Hyperlinks should be removed from the sequence before deciding which words appear together.

Now we are ready to define the output in response to a search. Among the pages where the two-word queries appear together, the top ten matches should be output in rank-order (of course if there are fewer than ten only the matching pages should be output, but these should be rank-ordered). The ranking is based on priority, defined as the product of (a) the number of times the two words appear together on a page and (b) the number of incoming links to the page. Among pages that have the same priority, the ordering should be lexicographic.

Output

The output should appear in four parts. See the course web page for examples.

- Output the words and then the pairs of words that appear together on the first page. This should be done in lexicographic order, with one word (or word pair) per line of output, and no repetitions. To keep this output reasonable, the first page will be relatively short.
- For each page that is "crawled" all pages that are hyperlinked from that page should be listed in the order that they are reached on the page (in the file). If a page is hyperlinked more than once from a page it should be output only once. The overall order of output should be the order in which the pages are crawled. If your program discovers that a page does not exist when it is crawled, then the name of the paage should be output together with a message saying that it does not exist. See the example output.
- Output the first 15 and last 15 (in alphabetical order) word-pairs that appear among all pages. For each pair, output the number of pages it appears on.
- Output the responses to each search. In doing so, output the two search words on one line (in lexicographic order, which is not necessarily the order of input), and then output the name of each page (up to 10) found in priority order, with each on a separate line indented by a `'\t'`. If no pages are found, output `<none>` on a separate line.

Comments

- This program will be made substantially easier with careful use of maps.
- I suggest that you create a class to store all of the information about a `page`. Once you do this, you can create a map that associates names of pages (file names) with the pages themselves.
- When parsing a page file, dividing the input up into separate lines is not particular relevant. Thus, I suggest that you do not use `std::getline`. Instead use the `get` member function of the `std::istream` library. As an example,

```
std::ifstream in_str( argv[1] );
int count = 0;
char c;
```

```
while ( in_str.get(c) )
  if ( isspace(c) ) count ++ ;
```

counts whitespace characters in a file, including the new-line and line-feed characters. It is important to realize that the above `while` loop is not equivalent to

```
while ( in_str >> c )
  if ( isspace(c) ) count ++;
```

In this loop, the expression `( in_str >> c )` **skips whitespace**. Therefore, the resulting value of `count` would always be 0!