University of Toronto
csc228S— **File Structures and Data Management, Spring 2000**

Assignment 2 — Web Search

Deadline A: at the beginning of tutorial, Friday 11 February.
Deadline B: at the beginning of tutorial, Friday 03 March.
Total weight: 15% of your course grade. Part A is worth a small fraction of this total weight.

Your company is working on a new web search engine. The job of a web search engine is to take strings supplied by a user and return a list of web pages that contain those strings. Typically, the list is extremely large. Your company plans to make millions by building an engine that returns a smaller list that better matches what the user is looking for.

Trolling the web

The search engine needs a way to associate strings with web pages. Your company has a tool that regularly trolls the internet following links from web page to web page. For each web page that it visits, it identifies "keystrings" that it thinks reflect the content of the page. Identifying these strings is a very difficult task, and your company has hired world-renowned specialists in Computational Linguistics to help design clever techniques. This is one of the important ways that your company hopes its search engine will be superior to those of its competitors.

The trolling program logs its results as it goes in a **troll file**. A new, huge, troll file is produced each day. The troll file consists of variable-length records, each of which contains the following, in order:

- A vertical bar character. This makes it easy to identify the beginning of each record.
- The URL ("Uniform Resource Locator") for a web page, such as `http://www.cs.toronto.edu/~dianeh/pubs.html`. This is a fixed-width field, and the URLs are left-justified and padded with blanks if necessary.[1]
- A set of keystrings associated with that page, in no particular order. The number of keystrings logged varies by web page. The keystrings are separated from each other by a newline character. Each individual keystring is a variable-length field, but you may assume that it does not exceed thirty characters.

As far as your program is concerned, the URL is just a string, as are the keystrings; you will never have to pull them apart.

A small sample troll file appears in figure 1.

Organizing the data for efficient search

The troll file contains all the information needed by the search engine, but in a very inconvenient form. We need to be able to search quickly for all URLs associated with a given keystring. Since the same keystring can appear any number of times in the file, associated with any number of URLs, that would involve a

[1] Using a fixed-width for the URL field is unrealistic. We have designed it this way purely to make the assignment easier. As you work on your program, think about what complications would be introduced if one were to to remove this simplification.

```
|http://www.cs.uofa.edu/~fred
discourse
laser printer
|http://www.cs.uofa.edu/~toni
resolution
proof theory
discourse
|http://www.cs.uofa.edu/~tim
architecture
systems
proof theory
|http://www.cs.uofa.edu/~jan
discourse
laser printer
proof theory
|http://www.cs.uofa.edu/~marsha
discourse
resolution
systems
```

Figure 1: Sample troll file.

linear search of the entire huge file. Your job on this assignment is to write a program that transforms the information in the troll file into a form that makes such searches much more efficient. Specifically, your company wants two files to be produced from a given troll file:

- A **URL file** that contains a set of URLs for each keystring in the troll file. (This is the opposite of the troll file, which contains a set of keystrings for each URL.) The URLs for a given keystring need not be in any particular order. Each set of URLs is stored in the file as a linked list whose nodes have room for many URLs. Realistically, we would make the node size equal to the number of bytes read in one disk access, but in your program, make it a small (and easy to change) constant. Notice that the URL file does not store any keystrings, only URLs. It is not necessary to store keystrings here because they are stored in the index file.
- An **index file** that is an index on the URL file. The index is in the form of a binary search tree[2]. Each node in the BST stores a keystring and a pointer to the first node of the linked list for that string in the URL file. The index is kept in a file; you are to assume that it will not fit into memory.

A small example

If URL and index files were created from scratch using the troll file in figure 1, the result might be as shown in figure 2. Some comments about the diagram:

- The diagram shows the *logical* structure of the files. Of course their physical layout is linear. So for example, the nodes of the tree would be laid out one after the other in the file. (The order in which they occur in the file would depend on the order in which they were created.) It is the pointers from parent to child that make them into a tree.
- Notice that some of the nodes are not full. But they all have capacity for the same number of URLs.
- Because our sample troll file was so small, the linked lists are very small. In practise, of course, the linked lists would contain many nodes, and each would have the capacity to hold many URLs.

[2] A B-tree, would be more suitable, but we're using a BST in order to make the assignment simpler.

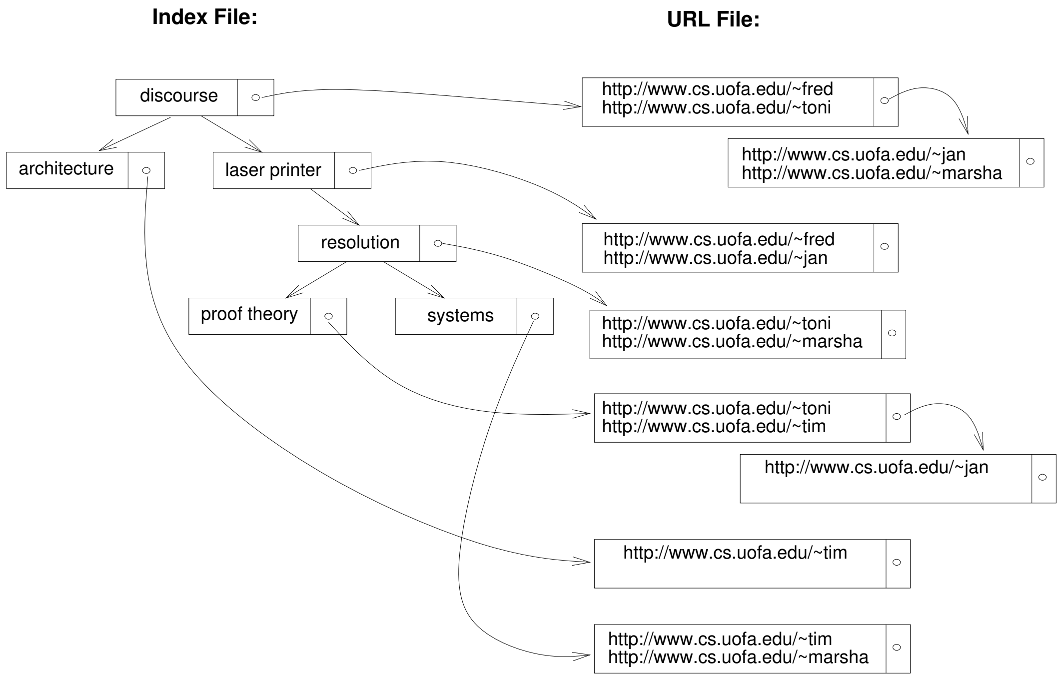

Figure 2: Logical structure of the URL file and index file that might result from processing the troll file shown in figure 1. For this example, nodes in the URL file can hold two URLs.

Your task

First, design file structure layouts for the URL file and index file. It may seem that everything has been decided, but the specifications above only describe the *logical* structure of the files. There are a number of design decisions left, including the following:

- For "pointers" into a file, will you use a byte offset or relative record number?
- When pointing from the index to a linked list in the URL file, will you point to just the front of the list, or to both the front and the back of the list?

Then, write a program that takes a troll file and produces a URL file and index file from it. Your program must also be able to take a troll file along with an *existing* URL file and its associated index file, and *revise* the URL and index files.

Notice that your program can only make the URL and index files bigger. Since web pages are constantly disappearing, we ultimately will need a facility for removing a URL; however someone else at your company will deal with that.

You will need to write a small program to test that the URL and index files were created or updated properly. This program should allow you to search the index by keystring and print out the set of URLs associated with it.

Ultimately, your company plans to build a retrieval program that can store a set of URLs in memory and perform binary operations on pairs of such sets. For example, it would allow one to intersect the set of URLs associated with "algorithms" and the set associated with "binary search". This is another way in which your company plans to do better than its competitors, but it is someone else's job to implement this.

Some advice about the code

Take time to design the code structure before you start writing. This means much more than a rough idea of what the classes are. For each class, you should design its interface precisely, giving careful specifications of what each public method does, including preconditions.

Also take time to plan the *process* of developing your code. In what order will you do it? What sort of testing can you do along the way? Plan to write and test the code incrementally, and to keep a record of such testing.

What to hand in for deadline A

For deadline A, you should have some code running. Exactly what is up to you, but it should be something non-trivial that is along the way to completing the assignment. Don't hand in a report with part A. Just hand a printout of your code (including any driver programs), and some test runs that demonstrate what works. Explain your test results with brief, hand-written comments on your printout. Try to make your submission as brief as possible.

What to hand in for deadline B

For deadline B, hand in your assignment in the form of a report, as described in the 228 Course Guide. Your report must include all of your code and all relevant input and output test files. Remember to justify the decisions you made in designing your program. This is a very important part of your report.

Your report should be thorough, well-written, concise, and honest. It should be no more than about five pages long, not including code and tests. Make sure that you run your report through a spelling checker.

You will also be required to hand in your assignment electronically. Please check the website for instructions.