Web Search – An Application of Information Retrieval Theory

Term Project

Summer 2009

Introduction

The goal of the project is to produce a limited scale, but functional search engine. The search engine should be able to provide a list of relevant documents when a query is given, just like any commercial search engine would do. It is in a limited scale that it is required to collect a limited number of documents (e.g. in the order of a few hundreds to a few thousands). The more your search engine can collect, the better it is.

This is a multi-phase, team project. It will start from the beginning of the semester and last through the semester. The detailed scope of the project, the team organization, technical information, and other details will be given as the semester progresses. An overview and the first part of the project will be given here.

Components of a Search Engine

A search engine consists of a collection of software components that work together to accomplish the task of collecting, analyzing a large number of documents over the Internet and giving the user a list of relevant documents and URLs when a query is issued to the search engine.

Major components of a typical search engine include the following:

- a **user interface** which takes the user input, passes the query to the back-end engine, displays the results sent back by the engine;
- a **crawler** which visits the Web and collects information about all the documents it encountered from the Web;
- an **indexer** which indexes each of the pages collected from the Web by the crawler and establishes links between keywords and documents that contain these keywords;
- a **ranker/retriever** which ranks the documents for a given query according to certain measures and retrieves the top relevant documents for the user;
- a **back-end engine** which takes care of network and file operations.

Figure 1 indicates the relation among different components in a typical search engine.

A search engine consists of two major parts, somewhat independent of each other, as can be seen from the figure. One is on the left side of the *document collection*, which answers user's queries. The other is

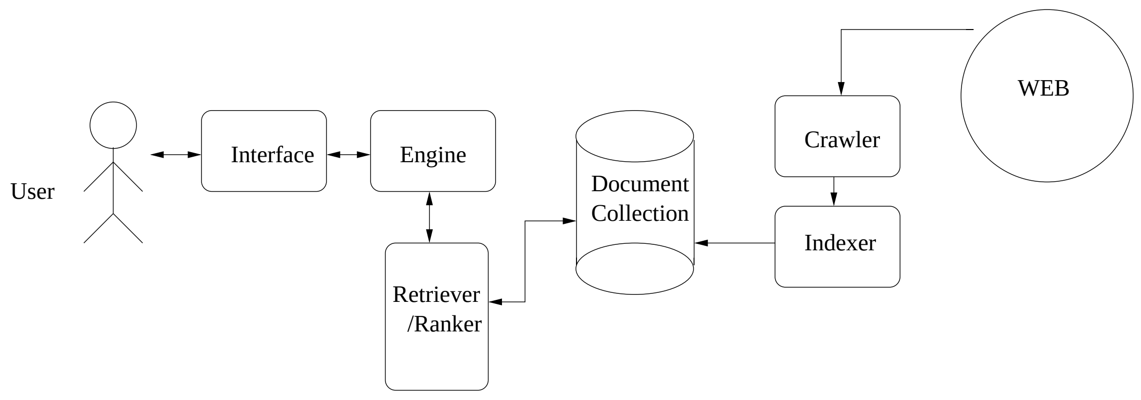

Figure 1: Components of a Typical Search Engine

on the right side of the *document collection*, which collects information from the Web so the URLs related to the user queries can be retrieved. A *crawler* goes around the Web to read Web pages and to extract information about each Web page it reads. The information is then sent to an *indexer*. The *indexer* takes this information and creates the links between keywords and the documents that contain these keywords. The result is typically saved into a file or a collection of files. When a user issues a query the document list is searched and a collection of relevant documents is generated. The *ranker* is responsible to rank these documents according to certain algorithms and measures. The top ranked documents are returned to the user for review. It is possible for the user at this point to review the documents and send feedbacks to the search engine. The *ranker* may take these feedback into account and re-select or re-rank the documents for the user to view.

Project Team

The project will be carried out in teams. The details of the team work are given in a separate handout.

Your Work in Phase One and Some Technical Details

Your phase one work is to implement a basic version of the interface and the back-end engine. This is just a framework. As the project progresses, some of these components will need to be enhanced. The interface part is responsible for the following main tasks.

- Display a main search engine page (you should design this page carefully, it will be the gateway of your search engine to the world).
- Read the user inputs from the browser.
- Display the results generated by the search engine.

The back-end engine takes care of the operations of files and network. It does the following.

- Maintain the communication between the browser and the search engine.
- Read and parse the inputs from the browser.

- Retrieve and send the results of the query to the browser.

There are some code examples in the directory of <http://localhost:9999/webpages/code/javaServer/> for this part of the project. Students who use C++ may find a similar example in C in the directory of <http://localhost:9999/webpages/code/cServer/>. You should before you design and develop your own program. Do the following.

- Download these program examples and save them into a directory of your own, e.g. `WebServer`.
- Compile the programs by

  ```
  javac EasyWebServer.java
  ```

- Run the program by

  ```
  java EasyWebServer port#
  ```

  where the `port#` is an integer of your choice. There are two restrictions for this number. It has to be greater than 1024. Port numbers below 1024 are reserved for system applications (for example, email server at port 25, telnet server at 23 and ftp server at 21). The second restriction is that the port number has to be unique. There cannot be two applications running at the same port on the same machine.

- Now that your simple Web server is running, point your favorite Web browser at `http://server:port/` to see the results generated by the server. The *server* is the name of the computer where your *EasyWebServer* is running, e.g. `localhost` and the *port* is the port number you chose to run the server in the previous step. For example, if the Web server is running on `localhost` at port 9999, then you should point your Web browser at

  ```
  http://localhost:9999/
  ```

- Try a few different paths on the Web server such as

  ```
  http://localhost:9999/sample.html
  http://localhost:9999/search
  ```

  Then read the programs and make sure you understand what the program is doing.

Let's discuss some technical details as we walk through an example. Assume the search engine is running at host `localhost` at port `9999`. We have issued the URL in our browser as

```
http://localhost:9999/search
```

An HTML form will be displayed in the browser as the result. We typed "123" in the first input box and "abc" in the second input box and then we clicked on the button "Submit". Let's now exam what happens between the browser and the server.

- When a Web browser contacts a Web server, it sends, among other things, a command of the form to the server

```
get /path http/1.1 \r\n\r\n
```

This means the browser is requesting (*get*) a page specified by */path* and the protocol that the browser is using is *HTTP 1.1*. It is required to have two consecutive new lines to end the command, each of which consists of a `new-line` character and a `carriage-return` character.

In our example, the command sent to the server from the browser is

```
get /search http/1.1 \r\n\r\n
```

This resulted in the form to be displayed to the browser's screen.

- The server has to parse this command to understand what the browser wants to do. A *get* command indicates that the browser is requesting some Web page(s). A *post* command indicates that the browser is sending some information to the server, for example, a search query.

- The browser also sends the length of the input to the server when it is requesting to post a form. This length tells the server how many bytes the browser is sending to the server as contents. The server is then expecting to read the number of bytes specified by the content length.

  In our example, when we fill in the form and click the "Submit" button, the browser is sending a "post" request to the search engine. The content of the form is sent to the server as the actual content after the header information.

- The following is a sample string the browser sends to the server when it is requesting a regular Web page. When you run the sample program, you will see this echoed on your server screen.

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [en] (X11; U; SunOS 5.8 sun4u)
Host: polaris:9999
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

- The following is a sample string the browser sends to the server when it is posting a form to the server. When you run the sample program, you will see this echoed on your server screen.

```
POST /form HTTP/1.0
Referrer: http://polaris:9999/search
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [en] (X11; U; SunOS 5.8 sun4u)
Host: polaris:9999
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 44
```

Note that the last line indicates that the content length is 44 characters. In this case, after the header part (in the above display) the browser is sending a form containing data to the server in the following format.

```
FirstInput=123&SecondInput=abc&Submit=Submit
```

which is exactly 44 characters. This string represents the form that is sent from the browser. The content length (44) and the content string depends on your input to the form. In our example, we typed "123" in the first input box and "abc" in the second input box. The content string is formed by the browser in the format of a sequence of name/value pairs. The name and value in a pair is separated by an equal sign "=" and the pairs are separated by an ampersand sign "&". In our example, *FirstINput, SecondInput, Submit* are the names of the form entries (read `form.html` to see where they are specified) and *123, abc, Submit* are the values corresponding to these names.

- Once the server parses the input form string, it captures all the input values. The server can then act accordingly. If this is a search engine, the server can pass this value as the query string to the *ranker/retriever* to retrieve relevant documents. In this phase of the project, you may just echo the query term to indicate that the server understands the query. The actual retrieval operation will be implemented in a later phase.

What to Hand In

Your team needs to hand in the following in the order given. Please staple all pages.

1. A team report for phase one of the project with a cover sheet. The report shouldn't be too long, maybe two to three pages. The report should include the following as a minimum.
   (a) The name of your team (should be same as your search engine);
   (b) A description of the roles of each team member and the contributions of each member;
   (c) A summary of the working process, e.g. what the team started with, what the team has accomplished, any problems encountered, how the team solved them, any thoughts on the project, among others.
   (d) Team meeting minutes during this phase of the work.
2. Source code for the programs and HTML pages.
3. Snapshots of sample runs from a Web browser.
4. Email the instructor a copy of the complete source code in `zip` or `tar` format.