Web Search – An Application of Information Retrieval Theory
Phase Two: Basic Text Processing And Indexing

Term Project

Summer 2009

Introduction

In this phase of the project you are to build the indexing component of the search engine. Refer to the description of the first phase of the project for the relation between the indexing component and the rest of the system.

The indexer takes a sequence of file names as input. For each of the files specified by the file name, the indexer processes the file in the following steps.

1. Lexical analysis (tokenizing)
2. Stopwords removal
3. Stemming
4. Selection of indexing terms among the word collection
5. Updating the indexing system for the newly processed file

You may restrict the types of files to process to a sub-set of all files. As a minimum your program should be able to handle plain text files and HTML files. Your program may ignore other types of files for now.

The result of the processing should be a inverted indexing system that all index terms are associated with a set of files.

In later stages of the project, a crawler will provide a sequence of URLs (file names) to the indexer. The result of the indexing system can be used by the retriever to select relevant URLs based on the search key words.

Some Details

Here we describe some of the details in each steps involved in the indexing.

Lexical Analysis

In general you can consider the input file as a long text string. The task of lexical analysis is to parse this long text string into a collection of words based on the specified delimiter(s). In this part of the processing you should also be able to extract the URLs contained in the file if they are part of the anchors. That is, they are within the

```
<a href = "http://www.somewhere.com/somepath/">anchor text</a>
```

Although we do not use these URLs in this stage of the process, it will be used in the crawling part of the project.

To ease the processing, it would be a good idea to pre-process the text such that

1. Multiple white spaces (i.e. new-line, tab and space chars) are squeezed into one and all new line characters, tabs are converted into the space character,
2. All characters are converted into lower case.

By squeezing multiple white spaces into one and converting all white space, search the pattern for URL becomes very easy. All you have to do is to search for the pattern of "`<a href`". Once you locate this pattern, extracting the URL after it is easy. Converting all characters into lower cases also makes the pattern search easier, except that it presents one problem: how to preserve a URL's path part? In the following URL

```
http://nlp.stanford.edu/IR-book/pdf/irbook.pdf
```

We want to preserve the cases of the path part of the URL. We don't want to get the URL as

```
http://nlp.stanford.edu/ir-book/pdf/irbook.pdf
```

since this is not the same URL as the original one. While there are many different ways of handling this problem, one possible solution would be to keep two copies of the input text. One copy keeps the original case, the other copy contains the lower case version of the original. We search the pattern of URL `<a href` in the lower case text while extracting the URL in the original one. Note that both copies should have been processed by squeezing the white spaces.

Alternatively if you'd like to keep the original HTML documents as they are, you may find useful the existing java classes such as the `HTMLLinkExtractor.java` available at the course web site

```
http://localhost:9999/code/misc/HTMLLinkExtractor.java
```

Stopwords Removal

Once the text is tokenized, we should remove the stopwords that we don't want to include in the index collection. Here you may come up a list of stopwords of your own, or you may search the Internet to find a source that has a list of common stopwords. These lists won't be the same among themselves. That is fine. You need to go through the list of words you just extracted from the text in the first step to remove the ones that are in the stopwords list. The key here is to use the same list of stopwords in indexing and later on in searching.

Stemming

You may use Porter's algorithm for stemming. I found a few implementations of the algorithm from the Web. You may experiment them a little bit and use the one you feel comfortable with. You can certainly implement your own version of the algorithm.

Click this link <http://locahost:9999/webpages/code/Porters/> to see a list of implementations of Porter's algorithm.

Selecting Index Terms

For simplicity we use all individual words as our indexing term for now. If you run into the problem of not having enough memory, you may have to select some terms from the whole collection. This can be implementation dependent.

Building and Updating the Indexing System

The most important part of this phase of the project is to build the indexing system. We will use an *inverted index* system, that is, indexing term points to a list of documents that contain the term.

Figure 1 shows how a inverted indexing system may look like. The basic index system has a list of index

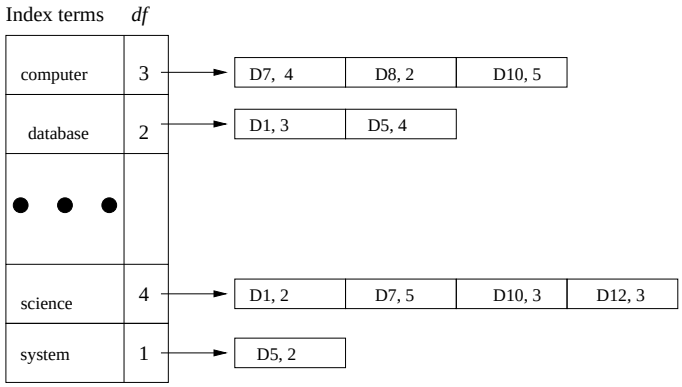

Figure 1: An Example of Inverted Indexing System

terms across the whole document set. Each index has a list of nodes each of which contains the information about that term and that document. In the simplest form a pair as document ID and term frequency is kept there. You may keep other information such as the location of the term within the document and the importance of the term as perceived within the text (whether or not a heading, bold faced ...). The index terms should be sorted alphabetically for easy search. Or you can use other data structures such as hash tables for the list or balanced binary search trees. Using hash table may reduce the number of terms you can index since hash table typically is less efficient in space.

What data structure to use in the indexing system is very critical to the search system. The main consideration should be the space efficiency. The more efficient your indexing system is, the more URLs you can index, thus the more you can support the search. Your team should choose the data structure. One easy way to save some space is not to store document name (URLs) everywhere. Rather, you should assign a document ID to each document. Use this ID when processing and storing information about the document. Keep just one copy of the document name and its mapping to the ID. So a mapping from

document name to a unique number is needed. The following is a general description of the algorithm to create an inverted index system.

```
Create an empty index term list I;
For each document D in the document set {
  For each token T in D  {
    If (T is not already in I)
       insert T into I;
    Find the location for T in I;
    If ((T, D) is in the posting list for T)
       increase its term frequency for T;
    Else {
       create (T, D);
       add it to the posting list for T;
    }
  }
}
```

Test Files

You may select any test files to work with. Keep in mind that your program should be able to work with at least a few hundreds of terms and a few thousands of documents. The document set should contain plain text and HTML documents. If you don't have other sources, you may try to index my `public_html` tree at

`~xmeng/public_html`

on the Sun file system, or

`~xmeng/sunhome/public_html`

on the Linux file system.

What to Hand In

Your team needs to hand in a hard copy of the following in the order given. Please staple all pages.

1. A team report for phase two 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) A description of the roles of each team member and the contributions of each member;
   (b) 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.
   (c) A description of the choices your team made for various steps in the text processing phase of the project.

2. Source code for the programs.

3. Snapshots of sample runs.

Email the instructor a copy of the complete source code in `zip` or `tar` format.