MSCI 541/720 - HW1
Assigned: Tuesday, January 16, 2018
Due: 11:00am, Tuesday, January 23, 2018.

Introduction

As discussed in lecture, a search engine can be divided several components: lexicon, ranker, snippet generator, metadata store, and a query front end. These components all depend on the output of the indexer, which consumes documents as its input. The indexer obtains the documents from the crawler.

In this class, we will not build a crawler. We provide you the document collection. Once you download the document collection to your computer, you will have completed the crawling of the documents!

We are not going to build the indexer in one fell swoop. Our first step is to become proficient at processing documents and start building the metadata store and document store.

Data

We will make available via secure download the LA Times portion of the TREC volumes 4 and 5 collection to Learn/D2L under "Content". You must have signed a TREC agreement to have access to this data. The data is gzip'd. The web contains lots of information on how to read gzip'd files from a computer program without ever uncompressing them on disk using your programming language of choice. For example, this post explains the Java way:
https://stackoverflow.com/questions/1080381/gzipinputstream-reading-line-by-line
In C#, we need these includes at the top of the program file:

```
using System.IO;
using System.IO.Compression;
```

And then we can get a StreamReader for the file as follows.

```
string filename = "latimes.gz" ;
FileStream instream = new FileStream( filename, FileMode.Open );
GZipStream gzStream = new GZipStream( instream, CompressionMode.Decompress );
StreamReader sr = new StreamReader( gzStream );
```

The data format is as described in class and as the uploaded READMEs also explain.

We have provided a sample of the larger file to enable to see what some of the data looks like before you begin work. The large file is so large, that it is a **bad idea** to try and view the large file in an editor / word processor. If you have a unix toolset available, you can do the following to view the large file: `cat latimes.gz | gunzip -c | less`

Problems

Complete the problems below. At the end of the assignment are instructions on how to submit your answers and work. As with other courses, it is important to always show your work for all problems where applicable.

1. a) Explain what is meant by "precision enhancing" and "recall enhancing" in the context of search engines.

b) In the case of an alphabetical, subject card catalog, describe and explain separately both a precision enhancing and recall enhancing technique that an indexer could employ.

2. Explain how a relational database and a search engine are both similar but different.

3. What are the advantages and disadvantages to downcasing all terms in a search engine?

4. Write two programs in the language of your choice such that the first program will read the latimes.gz file and be able to store separately each document and its associated metadata such that the second program can efficiently retrieve a document and its metadata. You must also demonstrate that your programs work correctly and meet all requirements. In effect, you must create a test plan with test cases and then show that your program passes your tests and thus meets the requirements. Your first program must meet the following requirements.

A. The program accepts two command line arguments[1]: a path to the latimes.gz file and a path to a directory where the documents and metadata will be stored. For example, if your program was named IndexEngine, you would run it from the command prompt / terminal / shell as:

   ```
   IndexEngine /home/smucker/latimes.gz /home/smucker/latimes-index
   ```

   If it is impossible for you to figure out how to run your program from the command line, you may prompt the user for these two arguments, but I would like you to eventually learn how to work from the command line.
B. If the directory where the documents and metadata are to be stored already exists, then the program should exit with an error message complaining that the directory already exists. If the directory does not exist, the program should create the directory.
C. If no arguments are supplied to the program, the program should exit with a help message explaining the usage of the program. If you are prompting for inputs, you should display instructions by default.
D. The program must be able to read the gzip'd latimes.gz file. Do NOT write the program to read an uncompressed version of the file.
E. Your program should not require the installation of extra software such as a relational database. For this course, our data needs are modest enough that we can use the file system and some simple files for our store.
F. What the program actually does, is determined by the needs of your second program, which is detailed next.

The second program must meet the following requirements:

A. The program accepts three command line arguments: a path to the location of the documents and metadata store created by the first program, either the string "id" or the string "docno", and either the internal integer id of a document or a DOCNO. For example, if your program was named GetDoc, you would run it from the command prompt / terminal / shell as:

   ```
   GetDoc /home/smucker/latimes-index docno LA010290-0030
   ```

   or

   ```
   GetDoc /home/smucker/latimes-index id 6832
   ```

   Again, if it is impossible for you to figure out the command line, you may have your program prompt the user for these inputs.
B. If no arguments are supplied to the program, the program should exit with a help message explaining the usage of the program. If you are prompting for inputs, you should display instructions by default.
C. Lookup of a document and fetching it should be a constant time operation. To clarify, you will not be marked on your time complexity, but your program must not scan through all of the full document texts.

[1] Many students don't know what the "command line" is. The command line is when you run a shell, e.g. the CMD shell in Windows or the Terminal (which actually runs a shell such as bash) on a Mac, and run your program from the prompt of the shell.

D. The textual content of the documents is largely contained inside the TEXT, GRAPHIC, and HEADLINE tags, and is the content that we will be using to search for documents. The metadata that we have for each document consists of:
    a. The internal integer id we are using to represent the document in our system. This is not the DOCID. It is the unique integer id that you have assigned to the document.
    b. The DOCNO. DOCNOs are found in the document, for example:

       <DOCNO> LA010389-0090 </DOCNO>

       and consist of only the non-whitespace characters.
       For the above example: "LA010389-0090" is the DOCNO, not " LA010389-0090 ".
    c. HEADLINE if it exists. If HEADLINE does not exist, should be an empty string or something else to allow you to know that it does not exist for the document.
    d. The date of the document. The date of the document is encoded in the DOCNO as LAMMDDYY-NNNN where MM is the month (01-12), DD is the day (01-31), and YY is the year, 19YY (ha!, year 2000 problem in the flesh for you here). Do not use the DATE tag. There are no 18YY nor any 20YY dates.

E. When run, the program should produce a nice output of a document's metadata as well as the complete raw document including its start and end DOC tags.

F. Note: this program needs to support both retrieval by internal ID and retrieval by DOCNO. This is why you need all 3 arguments. (The reason you'll want retrieval by internal ID, is to be able to debug other data structures that will only use the internal ID. Plus, I want you to build the mapping going from internal ID to docno for future use.)

For example if I was to do:

```
GetDoc /home/smucker/latimes-index docno LA010189-0018
```

I should get output like:

```
docno: LA010189-0018
internal id: 76235
date: January 1, 1989
headline: OUTTAKES: MATERIAL MOLL
raw document:
<DOC>
<DOCNO> LA010189-0018 </DOCNO>
<DOCID> 42 </DOCID>
<DATE>
<P>
January 1, 1989, Sunday, Home Edition
</P>
</DATE>
<SECTION>
<P>
Calendar; Page 24; Calendar Desk
</P>
</SECTION>
<LENGTH>
<P>
101 words
</P>
</LENGTH>
<HEADLINE>
<P>
```

```
OUTTAKES: MATERIAL MOLL
</P>
</HEADLINE>
<BYLINE>
<P>
By Craig Modderno
</P>
</BYLINE>
<TEXT>
<P>
Madonna and Beatty?
</P>
<P>
A Madonna rep tells us -- and production sources confirm -- that the blond
bombshell will play '30s nightclub singer-gangster's moll Breathless Mahoney
opposite Warren, who has the title role in the long-delayed "Dick Tracy."
</P>
<P>
As Breathless she'll sing at least two original songs, yet unwritten. And
she'll attempt to seduce the straight-laced copper away from lady love Tess
Trueheart, who is yet to be cast.
</P>
<P>
Beatty produces/directs the $30 million-plus Touchstone film, to begin
production in February, to be filmed on the Universal lot, according to
sources. Craig Modderno
</P>
</TEXT>
<TYPE>
<P>
Column
</P>
</TYPE>
</DOC>
```

Design Guidance

The first program should have a loop in it that processes the latimes.gz file one line at a time. This loop examines the lines and starts saving them when it find a <DOC> tag, and then packages them all up as a single document when the matching </DOC> is found. You should then hand off the raw document just found to a function that proceeds to extract metadata and save the document and metadata into the store.

One option for storing the documents for fast retrieval is to store each document as a separate file. There are approximately 130,000 documents in the file. You cannot store all of these documents separately in one directory. File system directories start to have trouble at approximately 1000 files. I recommend having your program create a directory hierarchy of YY/MM/DD such that only the stores from a single day are held in a directory. It looks like the data covers 1989 and 1990, so about 700 or so days. If the stories are uniformly distributed, you should have less than 1000 documents in a file. I recommend storing the files compressed, but doing so is not a requirement.

Alternatively, another option is to create a single file containing all of the documents but to record byte offsets so that you know where each story starts and ends. This is more complex and should only be attempted by students looking for a challenge.

What to Turn In

1. You may handwrite your answers to problems 1-3, but problem 4 requires a typewritten solution.
2. For problem 4:
    a. Provide evidence that you programs work correctly and meets all requirements. If effect, you need to have a reasonable test plan with test cases and then demonstrate the results of the test plan. You do not need to include test code unless you feel that this is the best way for you to demonstrate the correctness of your programs.
    b. Provide a printout of your source code. Your source code should be commented and easy to read.
3. All pages must be letter sized pieces of paper. You must staple multiple pages together with one metal staple in the upper left hand corner. Your stapling should be neat and tidy. Mangled staples are not permitted. Do not use report binders. You are welcome to use duplex printing to save paper.
4. Hand in the paper portion of the assignment to the dropbox on the second floor between E2 and CPH.
5. There will be two dropboxes in Learn to capture electronic copies:
    a. Upload a PDF copy your solutions to the problems to the online dropbox in Learn for the **written solution** to HW1. Do NOT submit a MS-Word document.
    b. Zip up all program files necessary to build and run your programs and upload the zip to the online dropbox in Learn for the **code solution** to HW1, problem 4.

Academic Honesty and Use of Existing Code Etc.

You may program in your language of choice. You must work alone on this assignment except that you may openly discuss design choices and the homework with anyone. You may use any code that you are legally allowed to use for non-commercial purposes excluding code:
1. written by classmates, and
2. complete retrieval suites of software such as Lucene, Lemur, Indri, Terrier, etc.
and you **must carefully acknowledge all code that you have not written yourself. You may not use code written by any other 541/720 student.** Please see the course outline for general academic honesty guidelines.

Future homework assignments will build on this one. Your later programs will require the functionality you are creating for these programs. **You may not share your code from this assignment with other students without instructor permission at any point in time, for some students may be granted extensions that you are not aware of.**

Marking

Problems 1-3 are worth 10 points each. Problem 4 is worth 70 points. Problem 4 will be marked on correctness, which must be demonstrated by you. Points may be lost on any problem for lack of neatness or failure to follow good style, grammar, spelling, etc. I will not be posting solutions to the homework. I will always make myself available to answer any questions you have about the homework, but I will not provide written solutions.