CIS 612 Project Proposal:

Project Gutenberg Inverted Index

By: Christen Ford

Description

The plan for this project is to at least create an inverted index over a chunk of the Gutenberg Project eBook library. The Gutenberg Project started back in the 1970’s as a free online repository of public domain text documents (today it is mostly eBook versions of famous works such as the works of Homer, Plato, and various other classical philosophers and writers). The Gutenberg Project makes their data set publicly available (as it always has been) however, the large issue with the library is acquiring the dataset (to be discussed later). If time permits, I plan on using the inverted index to implement a search engine over my personal eBook library.

Data Set

After batting things around in my head for a while, I settled on using the Gutenberg Project eBook library as my data source. This decision was two-fold:

1. The data set is easy to retrieve. I do not have to write any scripts to gather it as Project Gutenberg provides approved access methods to download the entire library directly on their website.
2. As of June 2018, the Gutenberg Project reported they had reached the 57,000-document mark in their collection. Because these are eBooks and not, say academic papers, they are far longer in length and provide more tangible data than other types e-documents would. Additionally, the library maintains supporting archives such as image files (for eBooks that contain images) and even MP3 files for audio books.

Each eBook maintains the same format (in this order):

1. The eBook header
2. The contents of the eBook
3. The eBook trailer

The header contains the following information on the eBook: the title the author, the release date on the Gutenberg Project, the language, and the character set encoding. The Gutenberg Project has eBooks in a wide variety of languages and character set encodings, however I have limited the eBooks I will be working with to English, ASCII encoded eBooks (which is a good chunk of the library).

The trailer does not contain any useful information for our purposes and is just standard Gutenberg Project boilerplate. I do not plan on including the header or trailer files in the indexing operation. I only want to index the contents of the eBook itself.

Data Collection Methods

As mentioned earlier, the Gutenberg Project makes their dataset publicly available through several methods on the Gutenberg Project website. This is great for me as I do not have to write any custom software to pull data from the site (not that I would, that would get my home IP address blacklisted). Instead, there are several options available for me to gather the data:

1. As an RDF/XML or MARC catalog (only contains the links to the eBooks, not the books themselves, which is a waste of bandwidth since the catalog is ~780MB)
2. Using the wget program (this may be the best method as it allows for configuring it with parameters that limit the set of books I pull down)
3. Becoming a mirror by using the rsync program to pull down the entire library (a good middle ground if I want the entire library, but I only want the English, ASCII encoded eBooks)

The Gutenberg Project also makes a RDF/XML and MARC catalog of the library available for download over FTP or HTTP that contains all of the metadata on each document in the collection, including a link for each physical file. This is impractical though. First, downloading the catalog is a waste of bandwidth; I still have to download the eBooks pointed to by the catalog. Second, RDF and MARC are file formats that are not really in use anymore, the catalog exists more as a historical record then as a usable data source.

Becoming a mirror is recommended by the Gutenberg Project if you are going to use the project in any official capacity (to not overwhelm the projects servers). However, I only want the eBooks that match the criteria I specified earlier, and the wget command allows me to do that. The only restriction is that I can only pull down a single eBook every two seconds, as any faster and the server will disconnect (meaning I will have to start over from the beginning). Additionally, it seems that the Gutenberg Project will stores the same eBook in multiple character set encodings for the same language. For instance, an English eBook may have a separate file for the ASCII, ISO-8859-1, and UTF-8 character sets. Even then, eBooks may state they are encoded in ASCII, but are in fact encoded in a different format.

Processing and Analytics Plan

I plan on executing this project in the following phases:

1. Data collection
2. Data extraction
3. Inverted index construction
4. Search engine construction (stretch goal)

Since, I discussed my plan for phase one in the previous section I will not discuss it here.

Once I have enough eBooks downloaded from the library, I can begin work on phase two, which is where the first hurdle I expect will occur. To reiterate, each eBook has a header, content, and a trailer. The header and trailer will not be indexed, however the metadata for each eBook can be found in the header. Fortunately, the header information is nicely arranged inn key value pairs for me, however some of the values split across multiple lines, which complicates processing the file. The start of the eBook and the end of the eBook is always delineated by a triple asterisk line that respectively look like:

“\*\*\* START OF THIS PROJECT GUTENBERG EBOOK [Book Title] \*\*\*”

“\*\*\* END OF THIS PROJECT GUTENBERG EBOOK [Book Title] \*\*\*”

Phase three should not be that difficult. I plan on using MongoDB to hold the extracted eBook content gathered in the previous phase. As for effectively processing the data? For that, I intend to use MongoDBs built in MapReduce operation. That said, the extraction of terms from the eBook content will not be done in MapReduce (although it should). This is because I will be using a stemming program along with stop word removal in the previous phase. Instead of storing the content of the eBook in the database, I will be storing the filepath to it alongside the stemmed words from the eBook (easily used as input in phase three). The physical files for each eBook will exist separately from MongoDB.

In phase four, I will be constructing a basic search engine that uses the inverted index generated in phase three alongside a cosine similarity index and vector ranking to return relevant documents to a user’s query. This will be implemented in NodeJS using Express and Pug as well as the Mongoose MongoDB connector. I am still working on the finer details of this phase, and will get to them when I actually get there.