Assignment 3 - Information Retrieval

Luiz Philippe Pereira Amaral

[1]Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)

Abstract. *This project is an indexer for web pages that can be used as a tool for information retrieval in large collections of documents.*
Keywords: *C++, Chilkat, web crawler, indexer, information retrieval*

1. Implementation

The code is written in C++ 17 and utilizes C++'s Standard Template Library (STL) in addition to the Chilkat and Gumbo parser libraries. While Gumbo parser is included on source, instructions for installing Chilkat can be found at `https://www.chilkatsoft.com/downloads_CPP.asp`.

1.1. Collection Reading

In order for the indexer to work, the document collection must be grouped in a folder and the documents must be named *1.html*, *2.html*, *3.html*... *n.html* (for a collection with *n* documents). Once the documents are organized properly, the indexer visits each one of them and uses Gumbo Parser to extract it's plain text content. This is done sequentially from *1* to *n*.

1.2. Indexing Terms

The indexing step is done after reading each document. Each word is extracted, sanitized, then added to the index. When adding a word to the index, the indexer checks if that word already exists in our dictionary, if it does not, it creates it's entry and proceeds. Then we check whether it has already occurred on that document, if it did, we simply increment the occurrences counter, otherwise, we create the word entry for that document. Finally, we store the new position where it appeared.

1.3. Term Sanitization

The sanitization process is done so that jalapeno will match jalapeño, which will match Jalapeño and so on. This is not a trivial task, since in the C++ char encoding, utf-8, accented letters like "á"are treated as a combination of two characters.

The solution for that problem is converting every utf-8 character to an utf-16 representation of that same character. However, the fact that utf-8 and utf-16 have different ways to represent some characters is due to the fact that they have different data sizes in memory, while utf-8 characters (char type) only takes one byte to be stored, utf-16 (wchar_t type) takes four. Since the indexer is already a very memory consuming application, we can not quadruple the size of every string by using wchar_t everywhere, so we need to apply the utf-8 to utf-16 conversion, manipulate every character, then revert the string back to utf-8 so the long chars are only used during this step of the process for each word.

1.4. Index Storing and Loading

After the index is completed, the program stores it in a file that can later be loaded back into memory (which is less expensive than re-building). This is convenient, since we do not want to leave the application running all the time between building the index and performing queries.

Each line on the index file represents a word from our dictionary, and it's format is shown bellow:

word n d_1 f_{d_1} p_{1d_1} p_{2d_1} ... $p_{f_{d_1}d_1}$ ... d_n f_{d_n} p_{1d_n} p_{2d_n} ... $p_{f_{d_n}d_n}$

where n is the number of documents where *word* appears, d_i is the i-th document where *word* occurs, f_{d_i} is the number of times *word* appeared on document i and p_{id_j} is the position where the i-th occurrence of *word* appeared on document j.

1.5. Compiling and running

You can use make to build the project by simply typing "sudo make install && make"on your terminal. The output is an executable file inside the build directory. You will need, however, to install the Chilkat library on your environment. Use "make run"to execute the crawler with a predefined seed then build the index, or use "./build/web-crawler -b [COLLECTION PATH]"replacing [COLLECTION PATH] with the path to the folder containing the collection of documents to be indexed or simply leaving it blank, it will run the indexing step only. by default the collection path will be "output/html/".

2. Results

The indexer was tested initially for a collection with one hundred thousand documents, and at document seventy thousand it was already using over 5GB of the primary memory. Then it was used to build a index for a mini collection of 5k thousand documents the statics for that process is presented bellow:

- Collection size: 844.823.393 bytes
- Index size: 42.790.795 bytes
- Size of the vocabulary: 159989 terms
- Average size of each inverted list: 13.3271 documents
- Execution time: 135063 milliseconds
- Memory consumption: 379 mega bytes (approximately)

These results show that while the crawling process is more time consuming (keep in mind that when crawling we used multiple threads, while for index we used a single process), the indexing step is much more memory-expensive. This could be optimized by not storing the whole index on primary memory at a time, but instead indexing a fraction of the documents, saving it to a file, then repeating the process until the entire collection is indexed. An additional complexity of that solution is what strategy to apply when merging the different indexes when querying the collection.