Assignment 5 - Information Retrieval

Luiz Philippe Pereira Amaral

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

Abstract. *This project is a simplified search engine that executes queries over a collection of web pages using a vector space model to determine the document similarity to queries.*
Keywords: *C++, Chilkat, vector space model, information retrieval, search engines*

1. Implementation

The objective of this software is to provide an efficient algorithm for searching through a large collection of documents and select those that have the best match with a given query. The code is written in C++ 17 and utilizes C++'s Standard Template Library (STL) in addition to the Chilkat, RapidJSON and Gumbo parser libraries. While Gumbo parser and Niels Lohmann's JSON libraries are included in source, instructions for installing Chilkat can be found at `https://www.chilkatsoft.com/downloads_CPP.asp`. The complete source code can be found at github.com/LuizPPA/web-crawler.

2. Collections and Indexes

The format of the collections upon which the software is capable of performing searches is a list of line-break separated JSON objects, each object containing an "url"and a "html_content". E.g.:

```
        {"url": "www.example.com", "html_content": "<html>...</html>"}
```

The program also utilizes two auxiliary files: an index for the collection dictionary and an index for the collection itself. The dictionary index is a file containing on each line a term present on the collection, the number of documents featuring the term, the id of each document, the amount of times the term appears on each document and for each document, each position where the term appeared. The collection index, necessary for assembling the results with efficiency, is a file where each line contains the identifier for a document, the link to that document on the web and a brief resume of the document's content.

Suppose we have two documents in our dictionary with identifiers 1 and 2 a possible collection index would be:

```
    1 http://www.document1.com a picture is worth a thousand words
    2 http://www.document2.com a journey of a thousand miles begins with a single step
```

The dictionary index for the terms "a", "journey", "thousand"and "words"would be:

```
    a 2 1 2 1 5 2 3 1 4 9
    journey 1 2 1 2
    thousand 2 1 1 6 2 1 5
    words 1 1 1 7
```

Is important to note that the lines of both the dictionary and collection indexes must also be ordered, the former by the lexical enumeration of the line's term, and the later by the value of the document's numeric identifier. Furthermore, each line of the dictionary index must be internally ordered (that is, the first documents in a term's list must be those where the term features the most times).

3. Searching the Indexes

Since both indexes are ordered, is possible to perform a binary search and find any entry in $O(logn)$ iterations. This allows us to very efficiently find any term or document. The algorithm for the search is as follows:

Algorithm 1: Search for a line starting with a given value on a file

```
input  : The value V we want to find (either string or number)
input  : The file F in which to search
output: The line starting with V on the file

$begin \leftarrow 0, end \leftarrow$ number of characters in $F$
$last\_middle \leftarrow -1$
while $begin \neq end$ AND $middle \neq last\_middle$ do
    $middle \leftarrow floor((begin + end)/2)$
    go to position $middle$ of $F$
    read from $F$ until a line break
    $V_f \leftarrow$ next word of $F$

    if $V_f > V$ then
        $end \leftarrow middle$
    else if $V_f < V$ then
        $begin \leftarrow middle$
    else
        $line \leftarrow$ whole current line of $F$
        return $line$

return line not found
```

This way, given a term on a query, we can retrieve both the term data and the documents featuring it in $O(logn)$ time. All is left to do then is to decide which of the retrieved documents are the most similar to the given query.

4. Vector Space Model and Document Ranking

The vector space model consists of a projection of a vector for the query and a vector for each candidate document on a n-dimensional space where n is the number of terms distinct on the query. This model assumes that all terms are linearly independent, therefore, the vector space is orthogonal. To calculate how similar is a document to a given query, or how similar it is to the query, we calculate the cosine of the angle between the vector that represents the query and the vector that represents the document:

$$sim(d_j, q) = cos(\theta) = \frac{\Sigma_{i=1}^{n} w_{i,j} \times w_{i,q}}{\sqrt{\Sigma_{i=1}^{n} w_{i,j}^2} \times \sqrt{\Sigma_{i=1}^{n} w_{i,q}^2}}$$

Where $sim(d, q)$ is the similarity of the document d with the query q, θ is the angle between the vectors of the document and the query, $w_{i,j}$ is the weight of the term i on the document j and $w_{i,q}$ is the weight of the term i on the query.

4.1. TF-IDF

The metric applied for the weight calculation is the $tf - idf$ (*term frequency/inverse document frequency*) index, which is based on the term frequency on both the collection and the query or a document. The first indicator, the term frequency, represents how descriptive is a term for a document or query, that value is proportional to the number of times the term features on that document. The tf of the term i in document j is calculated with:

$$tf_{i,j} = 1 + log(f_{i,j})$$

Where $(f_{i,j}$ is the number of times the term i appears on document j. The same is valid for a query instead of a document, thus:

$$tf_{i,q} = 1 + log(f_{i,q})$$

For calculating the inverse document frequency, we are interested in how specific is a term on the collection. The most documents containing the term, the most generic it is and less likely to be a relevant keyword on a given query. The idf of term i can be interpreted as:

$$idf_i = log(N/n_i)$$

Where N is the number of documents on the collection and n_i is the number of documents featuring the term i.

With the tf and idf values, we can calculate both the weight $w_{i,j}$ of the term i on a document j and the weight $w_{i,q}$ of the term i on a query q as such:

$$w_{i,j} = tf_{i,j} \times idf_i$$

$$w_{i,q} = tf_{i,q} \times idf_i$$

5. Results

To present the result of this ranking metric, a few query results are discussed below. Those queries were performed over a collection of 1.000.068 documents and 2.738.510 distinct terms. Is important to note that since the documents on the collection are from brazillian websites, the results are much more precise for portuguese queries, therefore, this will be the language used on the searches.

5.1. Queries

When performing the query *"Bohemian Rhapsody"*, the program returns links about the band Queen (which has a song with the same name) and about the movie "Bohemian Rhapsody". For the query *"Sete maravilhas mundo moderno"* ("seven wonders of the modern word"in portuguese) we get links about the *Cristo Redentor* statue, which is expected, since it is geographically close to the location of our web pages. When searching for *"Coliseu romano"* (Rome's Coliseum) we get links about the Vatican in Rome, about history and about Italy.

An interesting result is obtained when searching for *"Pacotes viagem"* (travel packages), which responds with links advising against traveling during the pandemic and about Covid-19 tests in airports.

Notably, while our results have some (considerable) degree of correlation with our queries, they are not exact matches. This is likely due to three main factors. Our collection of nearly one million documents is not in any way as broad as the world wide web, and since our queries are about fairly specific topics, it is to expect that the search won't return a perfect match. Another reason for this imperfection is the fact that our model assumes that terms are independent, which is not the case in many scenarios, since it is very likely that someone searching for "hot dog"is more interested on the dish "hot dog"than on "hot"and "dog"terms individually. And most important, there is not any kind of page rank, therefore we have no measure as to how relevant a document is, for example, when you search for "Bohemian Rhapsody", a link to Queen's official Spotify profile would be much more relevant than a YouTube cover, but as it is, we have no way to put that into account.

Although this is not the perfect approach, it is a crucial step for efficient and precise information retrieval, as we were able to perform complex queries in a very large collection in a timely manner (often less than a few seconds). Moreover, this strategy can be further refined to improve results precision and relevance.