

# Implementación de un Motor de Búsqueda de palabras en textos grandes

Daniel Palomino Paucar, Erick Tornero Tenorio,  
Felipe Moreno Vera, Luis Colchado Soncco

Universidad Católica San Pablo  
Arequipa, Perú

31 de agosto de 2018

## Resumen

El área de estudios de estructura de datos tiene múltiples aplicaciones, ya sea desde almacenamiento de elementos multidimensionales, objetos, multimedia, etc. También tiene otras aplicaciones en organización de información (así como los hashes o listas) y preprocesado de información (como en bases de datos).

En el presente proyecto se llevó a cabo el desarrollo de una interfaz web que mediante una adecuada comunicación con un programa núcleo en C++ permite emular un search-engine similar a google, el cual busca palabras o cadenas de texto en un conjunto de documentos de textos provistos por Wikipedia, la búsqueda se realiza mediante una estructura de datos óptima, la cual se le conoce como Inverted Index.

## 1. Descripción

El presente trabajo muestra la manera de implementar un Motor de Búsqueda de palabras en textos grandes utilizando algoritmos de indexación y de consultas para mejorar el almacenado de texto y las consultas a dichos textos, ya sea palabras o frases.

El proyecto consta de 3 partes fundamentales, donde en cada una se ha buscado la manera más óptima de implementación, para reducir el tiempo de consultas y búsqueda.

## 2. Arquitectura del software

La arquitectura del software se divide en 3 partes:

1. Motor de Búsqueda.
2. Consulta por consola.
3. Servicio web.

Tal como se muestra en la Figura 1:

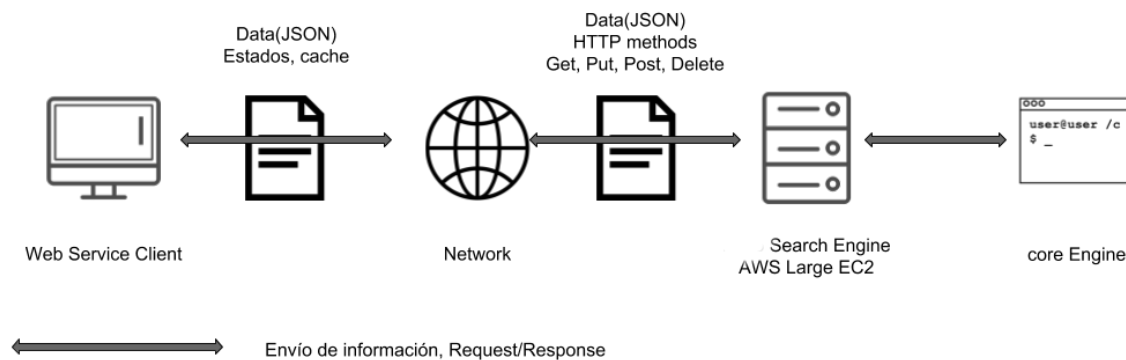


Figura 1: Arquitectura del search engine

## 2.1. Moto de Búsqueda

El Core del Motor de Búsqueda está basado en la estructura Inverted Index [1].

Antes de usar dicha técnica de indexado, se pasó a hacer un preprocesado del texto transformando todo a minúsculas y agregando una similitud entre letras con tildes y sin tildes.

Una vez ya estandarizado los textos, se procede a indexar los tokens (palabras) de los textos en una estructura similar a una tabla hash, con la modificación que se contabilizan las frases/palabras en una tabla de frecuencias.

### 2.1.1. Pre-procesamiento de texto

Existen 3 etapas de procesamiento de texto:

1. La primera etapa es la extracción de los documentos contenidos en los archivos, clasificándolos en la estructura *RetrievalData* que permite diferenciar el identificar, el título y contenido de cada documento.
2. La segunda parte es tokenizar el contenido de cada documento para eliminar los caracteres especiales ('@', '.', etc), y diferenciar las palabras que pueden contener información.
3. La tercera etapa consiste en estandarizar todos los caracteres que serán introducidos al *Inverted index* con el fin de minimizar el tiempo de consulta. Es decir considerar solo los caracteres que se les puede aplicar el *LowerCase* o *UpperCase*, esto es los caracteres del alfabeto. Además las vocales que llevan tildes son cambiadas a su equivalente sin tilde.

### 2.1.2. Inverted-Index y Ranqueo de Documentos

Dada la colección de documentos, asumimos que cada documento posee un identificador único al que hemos llamado *db\_index*. Durante la construcción del índice simplemente relacionamos cada documento con su identificador único.

La entrada de la indexación es una lista de tokens normalizada por cada documento, estructurada de tal forma que se relacione el token con el documento al que pertenece y la cantidad de ocurrencias que posee.

Cuando ocurre una búsqueda se extrae todas las relaciones encontradas de las palabras que componen la consulta y se ordena por orden de relevancia de acuerdo al siguiente criterio:

1. Los documentos que tienen más repeticiones de todas las palabras tiene más relevancia.
2. En caso de empate en el criterio anterior, se considera más relevante a aquellos documentos que posean más repeticiones de palabras individuales.
3. En caso de no encontrar todas las palabras juntas en el mismo documento, se considera más relevante a los que contengan la mayor cantidad de palabras de la búsqueda en el mismo documento.
4. En caso de empate en el criterio anterior, se considera más relevante a aquellos documentos que posean más repeticiones de palabras individuales.

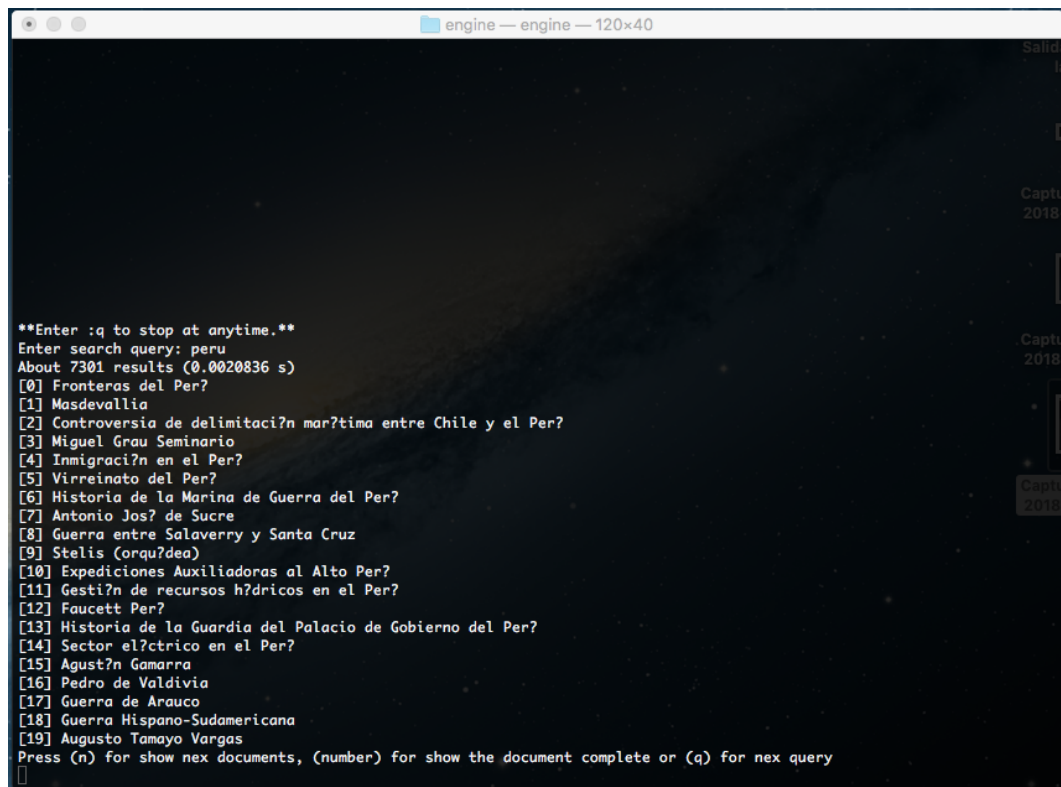
Es decir si se busca la frase *Comida Peruana*, primero aparecerán todos los documentos que contengan más repeticiones de todas las palabras *Comida* y *Peruana*. Luego se mostrarán los documentos que contengan más repeticiones por palabra individual *Comida* o *Peruana*

En otro caso de búsqueda *Guerra Curitiva*, por ejemplo, de no encontrarse ningún documento que contenga a ambas palabras juntas en un mismo documento, se mostrarán los documentos que contengan más repeticiones por palabras individuales.

Finalmente, si se buscara *GuerraXsksjdh*, dado que la palabra *Xsksjdh* no existe en ningún documento, se mostrarán como relevante a todos los documentos que sólo contengan la palabra *Guerra* ordenados según la cantidad de ocurrencias en cada documento.

## 2.2. Modo consulta por consola

Se desarrolló una aplicación en entorno de consola, en este caso se tiene la opción de realizar la búsqueda por palabra, y ver el tiempo de la consulta en segundos y los resultados de 20 en 20, se puede también de poder ingresar al contenido de los resultados, seleccionando su indicador de contador en la pantalla. Además se tiene la opción de cancelar una búsqueda actual para poder realizar una nueva búsqueda de otra palabra.



```
engine — engine — 120x40

**Enter :q to stop at anytime.**
Enter search query: peru
About 7301 results (0.0020836 s)
[0] Fronteras del Per?
[1] Masdevallia
[2] Controversia de delimitaci?n mar?tima entre Chile y el Per?
[3] Miguel Grau Seminario
[4] Inmigraci?n en el Per?
[5] Virreinato del Per?
[6] Historia de la Marina de Guerra del Per?
[7] Antonio Jos? de Sucre
[8] Guerra entre Salaverry y Santa Cruz
[9] Stelis (orqu?dea)
[10] Expediciones Auxiliadoras al Alto Per?
[11] Gest?n de recursos h?dricos en el Per?
[12] Faucett Per?
[13] Historia de la Guardia del Palacio de Gobierno del Per?
[14] Sector el?ctrico en el Per?
[15] Agust?n Gamarra
[16] Pedro de Valdivia
[17] Guerra de Arauco
[18] Guerra Hispano-Sudamericana
[19] Augusto Tamayo Vargas
Press (n) for show nex documents, (number) for show the document complete or (q) for nex query

```

Figura 2: Consulta por consola

## 2.3. Implementación de la conexión entre el programa Search-engine y el servicio web

En esta sección se presenta la interconexión usando HTTP entre el Search-Engine desarrollado en C++ y la interface web hecha en js y html con un servicio en golang.

### 2.3.1. Implementación de la interfaz web

Para la implementación de la interfaz web, se utilizó el webapp Sign, version 3.1, para el diseño de la interfaz.

#### Templates

Los templates, son ficheros con extensión *.tmpl* los cuales el lenguaje Golang interpreta y realiza un parse (se puede embeber información en dichos templates) dicha técnica es utilizada por todos los frameworks web como Django, ExpressJS, etc.

Los templates, permiten también manejar lenguaje de marcado html de forma nativa (sin cambiar la sintaxis) por lo cual hace más fácil la implementación y diseño de los componentes web.

#### Golang

Se utilizó el lenguaje de programación Golang para realizar el web service que se comunica con el servicio REST en cpp que se comunica con las consultas de la interfaz.

Se utilizó el protocolo HTTP por defecto, a diferencia de MQTT, HTTP se maneja de manera nativa en web services.

Además, la interfaz web se implementó utilizando el estilo application min y fonts de google para el tipo de letra 3.

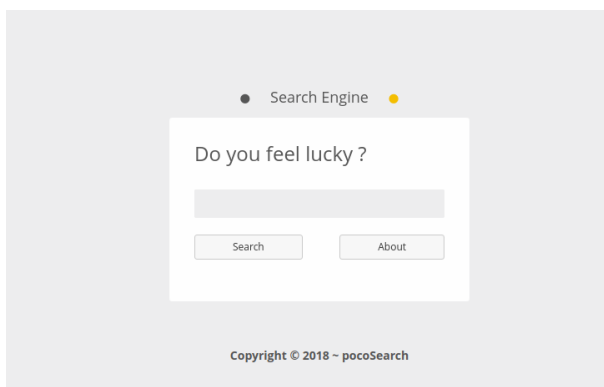


Figura 3: Vista principal del search engine

Al momento de realizar una consulta, nos retorna una lista de páginas de las cuales se identifican por Id de página (pero en la vista se presenta como título y contenido mínimo) y además el número de páginas en la parte inferior del documento de texto 4.



Figura 4: Resultados de una consulta en el search engine

### 2.3.2. Web Server

En lado del servidor se implemento un Web Server con el lenguaje de programación C++ y con la librería Boost. Esto nos permitió en específico crear endpoints que con los métodos del protocolo HTTP logamos realizar la comunicación entre el core y el cliente en entorno Web.

#### HTTP Protocol

Los métodos HTTP tales como Get, Post, Delete y Update se utilizaron para realizar las consultas entre ambas partes del core search engine, el REST service y el web service.

Para la consulta desde clientes web usamos métodos GET, entre las consultas del web client y el core search engine usamos métodos post.

## 3. Resultados

Se obtuvieron los siguientes resultados en promedio:

1. Tiempo de indexación: 168 segundos
2. Cantidad de Documentos Indexados: 259599
3. Memoria utilizada: 2.8 GB
4. Tiempo promedio de consulta: 2.3 ms

Se hicieron pruebas también con 11, 22, 33, 44 y 57 files utilizando, las palabras serie1: "maestria", serie 2:"maestria en cienciasz serie3: "maestria en ciencias de la computacion". Los resultados fueron los siguientes:

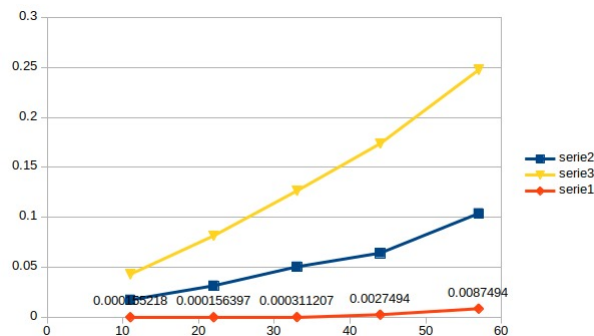


Figura 5: Resultados

## 4. Limitaciones

A pesar de los buenos resultados en cuestión de tiempo de indexación y búsqueda de frases y/o palabras, el presente trabajo posee limitaciones en cuanto a recomendaciones de otras palabras relacionadas cuando la palabra o frase original no se encuentra en ningún documento indexado.

Otra limitación a tomar en cuenta es el alto consumo de memoria para la indexación.

## 5. Trabajos futuros

El presente proyecto implementado podría incrementar su funcionalidad realizando búsquedas a páginas web para lo cual se tendría que implementar diferentes técnicas de conexión a la world wide web y extraer información, así como realizar nuevos parsers de estandarización de los miles de textos contenidos ahí. Y a su vez, se podría incrementar su eficiencia y rapidez (la cual se necesitará conforme vaya aumentando el número de información de textos que se tenga) para que las consultas se mantengan o disminuyan el tiempo de respuesta.

## 6. Conclusiones

Podemos mencionar las siguientes conclusiones importantes:

- El Inverted Index es una estructura óptima para la creación de motores de búsqueda de palabras en textos grandes debido a su fácil implementación y adecuado performance en tiempo de búsqueda.
- Siendo el consumo de memoria un factor crítico en la aplicación, el Hardware necesario para la puesta en producción debe poseer un adecuado dimensionamiento.
- El haber integrado la estructura de indexación (Inverted Index) al core del webserver permite la creación de una aplicación multi-usuario.

## 7. Distribución del trabajo

Para el desarrollo del presente proyecto, las tareas fueron divididas entre todos los colaboradores de la siguiente manera:

- **Desarrollo del Inverted Index** El desarrollo de la estructura Inverted Index, estructura la cual clasifica y asigna una determinada key a las palabras o cadenas de strings, así como también el desarrollo del cliente de consola (CLI) para realizar las búsquedas de palabras en los textos estuvo a cargo de Daniel Palomino.
- **Desarrollo del Parser** El desarrollo de la estandarización de información (textos) de tal forma que sea más fácil acceder y leer, estuvo a cargo de Erick Tornero.

- **Conexión web service - programa**

La implementación, configuración del Web Server en C++ con Boost library, el cual permite el intercambio de información por medio del protocolo HTTP, además de la configuración del servicio Web en el lenguaje Go y su configuración en la instancia EC2 en AWS, estuvo a cargo de Luis Colchado.

- **Interfaz Web**

El diseño y organización de los componentes visuales de la interfaz web, así como el desarrollo del web service de la interfaz web y la adaptación de templates en Golang y además de la funcionalidad de responsive en páginas web estuvo a cargo de Felipe Moreno.

## 8. Link repositorio Github

El link del repositorio es <https://github.com/AED-MINIGOOOGLE/minigoogle>.

## 9. Link Video

El link del video es <https://youtu.be/7ZhRS3Xr06o>

## 10. Link de la simulación web del Search Engine

El link de la página web que emula el search engine es <http://minigoogle.gescloud.io>

## Referencias

- [1] NLP Stanford, <https://nlp.stanford.edu/IR-book/html/htmledition/a-first-take-at-building-an-inverted-index-1.html>. Last visited 30/08/2018.
- [2] Erik Demaine. Session 16: Strings [Archivo de video]. Recuperado de <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-851-advanced-data-structures-spring-2012/lecture-videos/session-16-strings/>