Sistema Básico de Recuperación de Información

Prácticas de Sistemas de Recuperación de Información

Alejandro Arroyo Loaisa

Prácticas de SRI

Universidad de Jaén - 2024

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Introducción

Se ha solicitado la programación de un **Sistema de Recuperación de Información** básico, para la asignatura con el mismo nombre. El objetivo es, inicialmente, que este cumpla su función de la manera más eficiente posible y, después en posteriores prácticas, implementar mejoras sobre el sistema.

Qué es un Sistema de Recuperación de Información

Un **Sistema de Recuperación de Información** (o SRI para abreviar) permite la recuperación de la información, previamente almacenada, por medio de la realización de una serie de consultas a los documentos contenidos en la base de datos. La información recuperada debe coincidir con la necesidad de información del usuario, en mayor o menor medida.

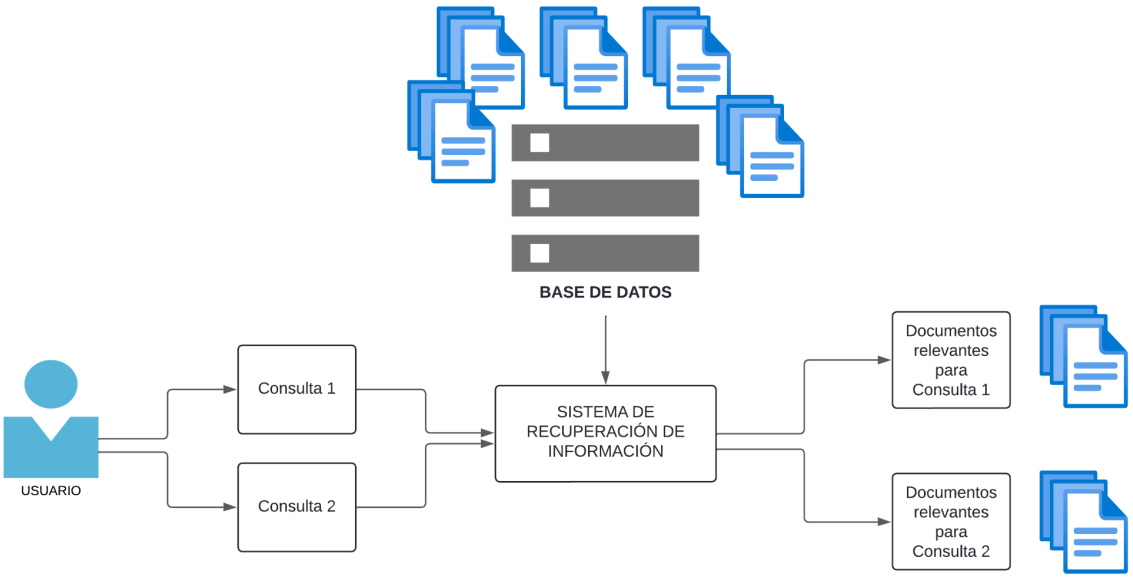

Diagrama 1 - Esquema de un Sistema de Recuperación de Información Básico

Objetivo de la Práctica

En esta Práctica se pretende programar un Sistema de Recuperación de Información basado en un **Modelo Vectorial**. Se da una colección de 1000 ficheros XML/HTML de Scielo, con textos en español e inglés de revistas científicas de ciencias de la salud y se requiere realizar consultas sobre estos datos. El proceso que se ha seguido es el siguiente:

1. Se procesa la colección XML/HTML, esto es se **EXTRAE** la información importante sin etiquetas (se ha considerado el Título, la Fecha, los Autores, el Tema y la Descripción como información importante, y en idioma Español), se **TRANSFORMA**,

es decir, se eliminan caracteres que no estén en el conjunto: *['a'...'z', 'A'...'Z', ' ', 'á', 'é', 'í', 'ó', 'ú', '_', '-', '\n','0'..'9']* y se crea una lista con todas las palabras separadas (tokenizar). Después, se **CARGA** esta información en un fichero por cada documento.

2. Se normalizan los ficheros almacenados en el punto anterior, esto es aplicar un proceso STOPPER y STEMMER:
    - **STOPPER**: consiste en reducir el número de términos de los documentos eliminando aquellas palabras que tienen poca carga semántica (palabras vacías). Se descarga una lista de *Stopwords* y para cada fichero de la colección se eliminan las palabras que aparezcan en dicha lista.
    - **STEMMER**: consiste en reducir el número de términos de los documentos normalizando diversas formas de un término a una raíz común. Para la extracción de raíces existen diversos algoritmos populares y eficientes, como el que se ha usado con la librería *Snowball.*
3. Se creará un **ÍNDICE INVERTIDO**. Esto es, se creará un índice de las palabras que aparecen en la colección completa de documentos y se anotará en los documentos que aparecen dichas palabras y el número de veces.
4. Se **CALCULAN LOS PESOS Y FRECUENCIAS INVERSAS** de los ficheros de la colección. Esto es, para cada palabra del índice inverso se calculará el peso de esa palabra en cada documento que aparece. Posteriormente se normalizarán y se almacenarán como un nuevo índice invertido.
5. Se crea el buscador, es decir, se **PROCESAN LAS CONSULTAS**. Se leen las consultas desde un fichero y se les aplica el mismo proceso que a los documentos de la colección. Una vez terminado este proceso, se calcula la similitud y se devuelven los *n* documentos en orden descendente que más similares sean.

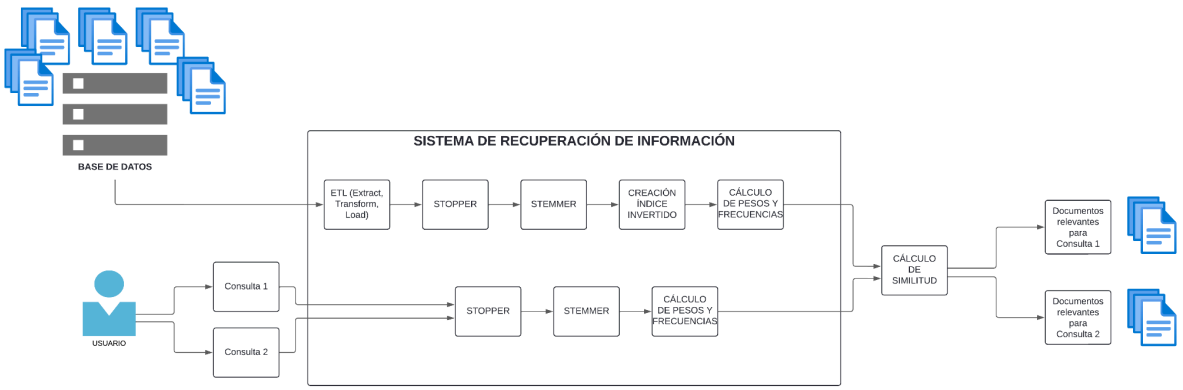

Diagrama 2 - Esquema del Sistema de Recuperación de Información implementado

Desarrollo de la Práctica

El proceso anteriormente descrito ha sido desarrollado a lo largo de varias sesiones de prácticas en forma de módulos. Es decir, para cada práctica:

1. Se **leen datos** de ficheros
2. Se **realizan operaciones** sobre estos datos
3. Se **obtienen unos resultados** y se almacenan (menos en la última sesión, que solo se devuelven los resultados)

Es una práctica incremental, es decir, para cada sesión se ha desarrollado un módulo que forma parte del cómputo total de un Sistema de Recuperación de Información. De manera que, al finalizar estas sesiones, se tenga en funcionamiento el sistema y cumpla su función.

Cada módulo equivale a un fichero de código en Python, y cada uno de ellos lee o almacena su información en un fichero diferente, en un directorio con un nombre orientativo. Realmente no es necesario dividir este sistema en módulos y que lea y guarde información por cada uno de ellos, se podría construir como un único sistema sólido que realizase todo este proceso ya que, si no es así, se pierde eficiencia. Pero se ha realizado de esta manera con fines educativos.

Detalles de la Implementación

Se van a detallar algunas decisiones de diseño que se han llevado a cabo a lo largo de las sesiones de prácticas:

- Procesar Colección XML/HTML

Para este módulo se ha utilizado la librería creada precisamente para ello, *BeautifulSoup.* Esta librería, indicando el conjunto de datos y que están en formato XML, es capaz de buscar y aislar información que esté etiquetada de la manera que se le indique, con la función *find('nombre_etiqueta')*. En este caso, para aumentar el contexto de los documentos se ha querido recuperar de estos:

Título	Fecha	Autores	Tema	Descripción

La información recuperada se ha almacenado en un fichero por cada documento en formato *.json*.

- STOPPER

Para este módulo, se ha descargado la lista de *Stopwords* del español del siguiente enlace:

http://members.unine.ch/jacques.savoy/clef/spanishSmart.txt

Posteriormente, en el programa, se han leído del fichero las palabras y se han cargado en una Lista para su consulta. Después y, documento a documento, se leerá la información

contenida y se eliminarán TODAS LAS PALABRAS QUE ESTÉN EN LA LISTA. Esto desde Python es bastante sencillo con el operador IN.

Los datos para filtrar se recogieron de ficheros *.json*, se almacenaron también en Listas y se guardaron, al finalizar, en ficheros *.json*, de nuevo.

Además se ha utilizado la librería *Counter* para el conteo de palabras para estadísticas, ya que esta sirve para contar datos en conjuntos, de manera eficiente y con funciones para ello.

- STEMMER

Para este módulo, se ha utilizado la librería ya existente para ello *SnowballStemmer*, para el proceso de Stemming. Indicándole un idioma en su creación a un objeto de esta clase (en este caso, Español) se le puede pedir mediante la función *stem('palabra')* que realice automáticamente el Stemming de la palabra indicada.

Los datos para filtrar se recogieron de ficheros *.json*, se almacenaron también en Listas, se le realizó la extracción de raíces y se guardaron, al finalizar, en ficheros *.json*, de nuevo.

Además se ha utilizado la librería *Counter* para el conteo de palabras para estadísticas, ya que esta sirve para contar datos en conjuntos, de manera eficiente y con funciones para ello.

- Creación del Índice Invertido

Para este módulo, se han aprovechado algunas funcionalidades internas de Python interesantes en el manejo de estructuras de datos, debido a que estas están creadas para lograr la eficiencia más alta posible. Se utilizarán diccionarios, que funcionan de la siguiente manera:

> En Python, 'índice[palabra] = {}' es una instrucción que crea un nuevo diccionario vacío y lo asigna a la clave 'palabra' en el diccionario 'índice'.
>
> 1. 'índice' es un diccionario existente.
> 2. 'palabra' es una clave en el diccionario 'índice'.
> 3. '{}' crea un nuevo diccionario vacío.
> 4. 'indice[palabra] = {}' asigna el nuevo diccionario vacío a la clave 'palabra' en el diccionario 'índice'.
>
> Por lo tanto, después de ejecutar esta instrucción, si intentas acceder a 'índice[palabra]', obtendrás un diccionario vacío '{}'.

Los diccionarios permiten el acceso a los valores a través de sus claves de manera muy rápida y eficiente. Esto se debe a que están implementados como tablas hash, lo que

permite que las operaciones de búsqueda, inserción y eliminación se realicen en tiempo *O(1)*.

Cabe mencionar que, a diferencia de las listas, los diccionarios no están ordenados y se accede a los elementos mediante su clave en lugar de su posición. Esto significa que no importa cuán grande sea el diccionario, el tiempo que se tarda en buscar una clave específica no aumenta con el tamaño del diccionario, lo que es perfecto para aplicarlo en nuestro problema.

Se ha aplicado de la siguiente manera:

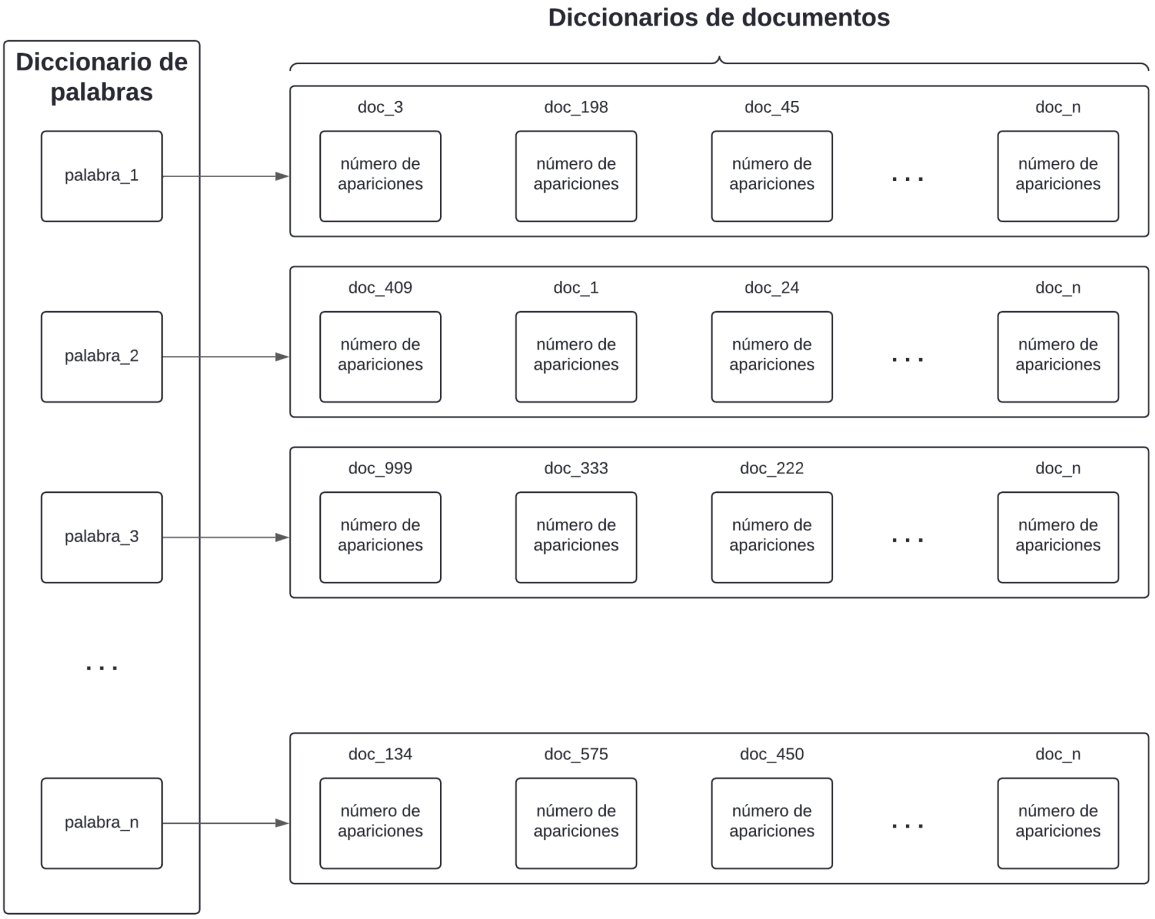

Diagrama 3 - Esquema del Índice Invertido implementado

Es un diccionario de palabras que han sido indexadas utilizando la propia palabra como clave. Y cada palabra, a su vez, almacena un diccionario con los documentos en los que aparece y cuántas veces apareció, siendo estos indexados por su nombre de documento.

De esta manera, nos ahorramos crear las cuatro estructuras diferentes e independientes que se sugirieron: *term2id, id2term, doc2id* e *id2doc,* agrupándolas todas en una. Estas estructuras sirven para consultar los identificadores asignados a palabras y documentos y

son matrices, por lo tanto la búsqueda de la información en estas es más lenta (aunque se pueden implementar también con diccionarios y aumentar la eficiencia).

- Cálculo de Pesos y Frecuencias

Para este módulo, hay que realizar una serie de cálculos (especificados en el material de prácticas) para conseguir los pesos y frecuencias de las palabras en los documentos.

Uno de estos cálculos, es saber cuál es la frecuencia máxima de las palabras que contiene un documento. Para ello, hay que recorrer todos los documentos de todas las palabras y almacenar la frecuencia más alta para cada documento. Es algo ineficiente *O(n*m)*, pero no queda más remedio que consultar toda la información. Las frecuencias han sido almacenadas en un diccionario, indexadas con el nombre del documento como clave.

Una vez conozcamos la frecuencia máxima de cada fichero, normalizamos estas frecuencias dividiendo cada frecuencia de documento entre su frecuencia máxima, almacenada en el diccionario, de acceso *O(1)*.

A continuación, calculamos el IDF y multiplicamos las frecuencias de cada palabra por su IDF. Este IDF lo almacenaremos en el Índice Invertido, para su uso posterior, de la siguiente manera:

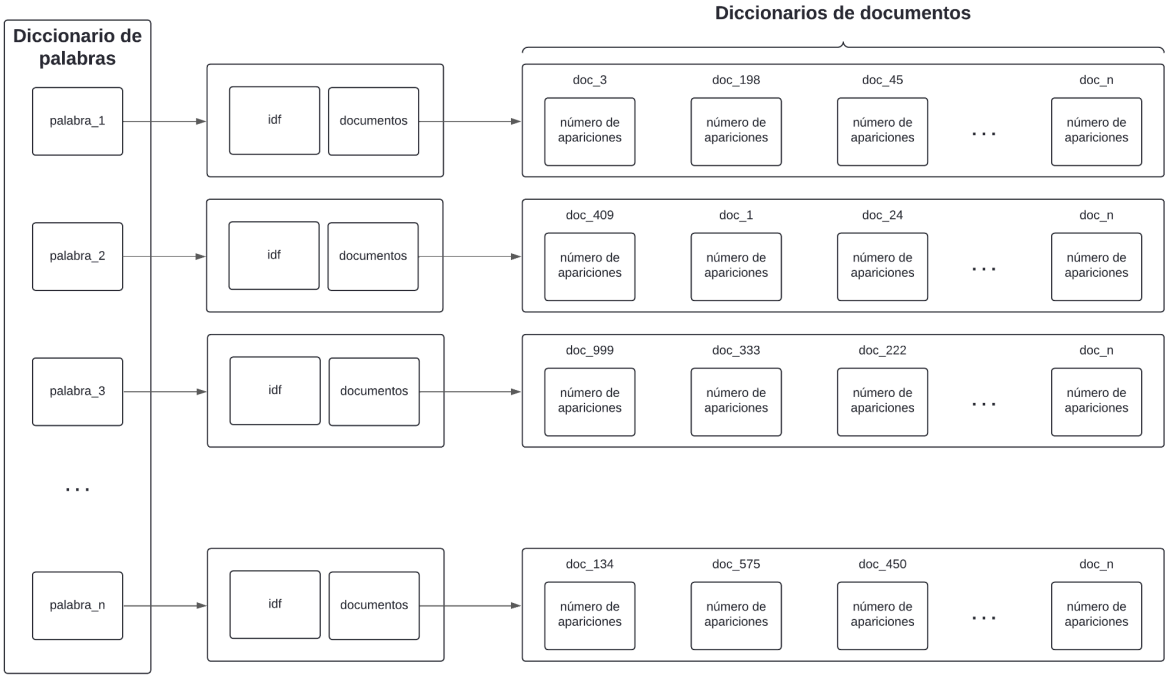

Diagrama 4 - Esquema del Índice Invertido con IDF almacenado

Esto modificará la forma de acceso a la información, ligeramente. Por último, normalizamos estos pesos dividiéndolos entre el módulo del vector de dicho documento, almacenándolo después en el índice y guardándolo en memoria.

- Procesamiento de la Consulta

Para este módulo, se han reutilizado partes de los anteriores debido a que hay que procesar las consultas de la misma manera que los documentos.

Antes de nada se carga en el sistema el fichero de configuración del programa, que incluye información de las direcciones de los índices, consultas, listado de stopwords… como parámetros de ejecución como el número de documentos relevantes máximo que devolver.

Las consultas están escritas en un fichero de texto, en el directorio del programa. Son leídas y almacenadas en una lista, tras aplicarle el proceso de transformación, stopper y stemmer.

Posteriormente, y para cada consulta se calcula su peso de la misma manera que con los documentos, pero reutilizando el IDF almacenado en el índice invertido. Tras esto, se calcula la similitud multiplicando los documentos por la consulta, almacenando los resultados para cada documento en una lista de documentos, ordenada de manera decreciente por su similitud respecto de la consulta.

Esta información se le devuelve al usuario, limitando el número de documentos que se muestran por consulta con el valor leído del fichero de configuración.

Ejecución de la Aplicación

La ejecución de cada módulo se ha realizado de manera independiente y a la finalización del desarrollo del mismo. Esto significa que los tiempos de ejecución se verán afectados por la lectura y escritura de los datos en ficheros durante cada módulo.

Parámetros necesarios para la ejecución del Sistema

Para la Ejecución del sistema se necesitará:

- Colección de documentos en formato XML/HTML
- Ficheros en formato .json generados tras la ejecución de cada módulo
- Listado de Stopwords del idioma seleccionado
- Fichero de configuración del Sistema de Consultas
- Fichero de texto con las Consultas (a consulta por línea)

El fichero de configuración del Sistema de Consultas se encontrará colgando del directorio base del programa y contendrá la siguiente información:

- Dirección del fichero *.json* que contiene el Índice Invertido con los pesos calculados
- Dirección del fichero *.txt* que contiene las Consultas para el Sistema
- El número de documentos relevantes por consulta que se devolverá
- Dirección del fichero *.txt* que contiene las Stopwords

Un ejemplo para este fichero de configuración, denominado *config.txt*, sería:

```
calcular_pesos/calcular_pesos.json
consultas/queries.txt
5
stopwords.txt
```

Información de la ejecución de cada módulo

A continuación se muestra el detalle de la ejecución de los módulos del Sistema de Recuperación de Información. Cada módulo, al finalizar, muestra algunos datos de su ejecución. Estos datos se muestran en la consola, pero también crea un fichero de texto con la forma "*statsNOMBRE_MÓDULO.txt*" en el directorio del programa y almacena esta información.

- Procesar Colección XML/HTML

```
PREPROCESADO
Tiempo de ejecución total: 2.513314962387085 segundos
        ->Se han procesado: 1000 ficheros
        ->Tokens totales: 277220
        ->Número medio de tokens: 277.22
```

- STOPPER

```
STOPPER:
Tiempo de ejecución total: 1.6949942111968994 segundos
        ->Palabras totales ANTES: 277220 | Palabras totales DESPUÉS: 151139
        ->Media palabras ANTES: 277.22 | Media palabras DESPUÉS: 151.139
        ->Mínimo de palabras ANTES: 142 | Mínimo de palabras DESPUÉS: 73
        ->Máximo de palabras ANTES: 921 | Máximo de palabras DESPUÉS: 452
Las (10) palabras más comunes antes:
[('de', 22743), ('la', 11884), ('en', 9383), ('y', 9140), ('el', 7608), ('los', 5895), ('con', 4134), ('a', 3661), ('se', 3600), ('del', 3402)]
Las (10) palabras más comunes después:
[('pacientes', 2088), ('renal', 1128), ('estudio', 979), ('resultados', 965), ('años', 952), ('tratamiento', 753), ('p', 718), ('edad', 677), ('enfermedad', 656), ('objetivo', 527)]
```

- STEMMER

```
STEMMER:
Tiempo de ejecución total: 3.2653579711914062 segundos
        ->Palabras totales ANTES: 151139 | Palabras totales DESPUÉS: 151139
        ->Media palabras ANTES: 151.139 | Media palabras DESPUÉS: 151.139
        ->Mínimo de palabras ANTES: 73 | Mínimo de palabras DESPUÉS: 73
        ->Máximo de palabras ANTES: 452 | Máximo de palabras DESPUÉS: 452

Las (10) palabras más comunes antes:
[('pacientes', 2088), ('renal', 1128), ('estudio', 979), ('resultados', 965), ('años', 952), ('tratamiento', 753), ('p', 718), ('edad', 677), ('enfermedad', 656), ('objetivo', 527)]
Las (10) palabras más comunes después:
[('pacient', 2360), ('estudi', 1574), ('renal', 1246), ('result', 1090), ('años', 965), ('tratamient', 812), ('objet', 805), ('edad', 735), ('present', 731), ('p', 718)]

        ->Palabras únicas totales ANTES: 105398 | Palabras únicas totales DESPUÉS: 97488
        ->Media palabras únicas ANTES: 105.398 | Media palabras únicas DESPUÉS: 97.488
        ->Mínimo de palabras únicas ANTES: 52 | Mínimo de palabras únicas DESPUÉS: 49
        ->Máximo de palabras únicas ANTES: 276 | Máximo de palabras únicas DESPUÉS: 232
```

- Creación del Índice Invertido

```
INDICE_INVERTIDO
Tiempo de ejecución total: 0.2797729969024658 segundos.
Palabras totales indexadas: 13483 palabras.
Tamaño en memoria del índice: 589920 bytes.
```

- Cálculo de Pesos y Frecuencias

```
CALCULAR_PESOS
Tiempo de ejecución total: 0.22798967361450195 segundos.
Palabras totales indexadas: 13483 palabras.
Tamaño en memoria del índice: 589920 bytes.
```

- Procesamiento de la Consulta

```
1  S1699-695X2015000300004.json   0.27979673601308724
1  S0212-97282016000300023.json   0.18885941316752464
1  S1699-695X2012000100002.json   0.18186900628894312
1  S0212-97282015000300027.json   0.17824288880318698
2  S1699-695X2014000100003.json   0.406162355133046
2  S1139-76322011000200005.json   0.40396722277687885
2  S0212-97282013000100023.json   0.3311515214236478
2  S0212-97282014000100033.json   0.2759596011863732
2  S1699-695X2011000300003.json   0.2737017465635869
3  S0211-69952015000400007.json   0.3619892435930315
3  S0211-69952012000100009.json   0.23920392254284967
3  S0211-69952012000500013.json   0.23266409351320075
3  S0211-69952014000600004.json   0.2264704834278219
3  S0211-69952012000500007.json   0.21502578262593627
4  S1139-76322010000500010.json   0.3505909043051133
4  S1139-76322013000200003.json   0.34107918397298764
4  S1139-76322009000100007.json   0.2865701746593761
4  S0212-97282014000200037.json   0.2676861876200719
4  S1139-76322014000400013.json   0.24138207374695622
5  S0211-69952017000100009.json   0.5243755392054035
5  S1139-76322014000400014.json   0.329081877193156
5  S1699-695X2011000100005.json   0.3227013688193007
5  S1139-76322010000500002.json   0.27899054506217047
5  S1699-695X2009000300004.json   0.2679914091699667
Tiempo total de ejecución:  0.15404248237609863 segundos.
```

Conclusión

En conclusión, la práctica realizada ha permitido profundizar en la programación y funcionamiento de un **Sistema de Recuperación de Información (SRI)**. A través de la recopilación y análisis de datos sobre los tiempos de ejecución y resultados obtenidos, se ha podido evaluar, en aspectos generales, el rendimiento del sistema desarrollado.

El sistema programado, según los datos obtenidos, tarda en ejecutarse una media de *8,132 segundos*. Esto es, el proceso completo de procesamiento de la colección, indexación de los datos y realización de la consulta. Se ha de tener en cuenta que la modularización del sistema ha incurrido en el detrimento de la eficiencia, debido a que, como se ha mencionado en páginas anteriores, para cada módulo se lee y se escriben datos en ficheros, por lo que se pierde eficiencia. Si esto no se tratase de una práctica modular y todo formase parte de un único sistema y una única ejecución, el SRI sería más veloz.

Cabe destacar que lo más costoso de este SRI es el procesamiento e indexación de la colección de documentos, ya que la posterior búsqueda de información en el sistema es muy rápida. Si nuestra colección de documentos permanece invariable con el tiempo, podemos mantener procesada la colección y solo realizar consultas en un tiempo muy razonable, sin repetir siempre todo el proceso.

Las consultas de ejemplo con las que se ha ejecutado el sistema, sobre una colección dada de 1000 documentos, han sido:

La diabetes en personas mayores.
Los jóvenes universitarios.
La insuficiencia renal.
El asma un importante factor para predecir virus.
Algunas enfermedades respiratorias, infecciosas e intestinales.

El Sistema de Recuperación de Información ha sido ejecutado sobre un ordenador con las siguientes características:

- Procesador: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz
- RAM instalada: 8,00 GB (7,81 GB usable)
- Sistema Operativo: Windows 10

Los resultados obtenidos han demostrado que el SRI es capaz de procesar y recuperar información correctamente. Sin embargo, también se han identificado áreas de mejora potencial que podrían optimizar aún más el rendimiento del sistema. Estas mejoras serán implementadas en la siguiente práctica.

Esta experiencia ha proporcionado una valiosa oportunidad para aplicar los conceptos teóricos aprendidos en clase a un proyecto práctico y real.