**MEMORIA TRABAJO SAR**

**Componentes:** Miguel Arturo Martínez Gutiérrez, Adrián Davia García, Mykyta Grygoryev y Miguel Ángel Navarro Arenas.

Dentro de esta memoria vamos a explicar la lógica que hemos aplicado a la hora de programar nuestro proyecto, abordando y explicando los aspectos lógicos que hemos decidido usar para resolver el problema que se nos presentaba inicialmente en la asignatura. Estudiaremos todos los puntos clave, tanto del indexador, como del searcher como de la librería (SAR\_lib.py) que nosotros hemos valorado como mejores a la hora de desarrollar el proyecto.

Como referencia tomaremos el archivo en formato pdf que se nos ha facilitado para realizar el trabajo. Vemos que el trabajo se nos divide en dos funcionalidades básicas, los aspectos obligatorios y los aspectos extra. También tenemos dos funcionalidades básicas que cumplir, el indexador y el buscador. Estas funcionalidades las iremos formando a través de SAR\_lib.py, y les daremos uso en SAR\_searcher.py y en SAR\_indexer.py. En nuestro caso, hemos completado todas las funcionalidades, excepto Multifield y búsqueda posicional.

Procederemos a explicar la estrategia que hemos utilizado en cada uno de los puntos que se nos propone en el pdf:

**1.1.- Indexador**

A la hora de realizar la indexación de noticias, hemos optado por lo siguiente: en el index\_file, hemos utilizado un identificador único, tanto para noticias como para documentos. Tras abrir el filename que recibimos como entrada, vamos rellenando tanto self.docid como self.docs. El primero lo utilizamos para saber qué id le corresponde a cada uno de los documentos de la colección. El segundo, tiene la finalidad de guardar en una lista los filename de los documentos, dependiendo de si id. Posteriormente, rellenaremos self.newid con los nuevos id de los documentos. También, llenaremos self.news, self.newsKeyWords y self.newsDate, con el título, las keywords y la fecha, correspondientemente, de cada una de las noticias. Para la indexación en si, hemos tokenizado el contenido del artículo de cada una de las noticias. Para ello hemos creado también una posting list de cada término del cuerpo de la noticia. Esta posting list contiene los identificadores de las noticias donde hemos encontrado dicho término.

**1.2.- Recuperador de noticias**

En este caso hemos utilizado una versión ligeramente adaptada del algoritmo Shunting Yard para le parseo de la consulta. Una vez convertida la notica a posfija la resolvemos aplicando los algoritmos de conjunción, disjunción y negación que hemos visto en teoría.

**2.1.- Stemming**

La finalidad del Stemming es reducir términos a su raíz, para ello simplemente, en make\_stemming hemos recorrido todos los términos que teníamos guardados en el self.index para luego extraerles su raíz a través de self.stemmer.stem(termino). Después de esto es necesario crear un nuevo diccionario que guarde la raíz asociada a cada uno de los términos a los que pertenece. El get\_stemming es simplemente un buscador. Dado un término, buscamos en el diccionario que hemos creado anteriormente

**2.4.- Ranking**

Para que fuera más sencillo la implementación de las consultas, hemos decidido ‘tokenizarlas’ previamente con el método query\_tokenizer. Esta tokenización retiraba los AND, OR y NOT de las consultas, junto con los paréntesis. También convertía la consulta en una lista con los términos de esta. Con todo esto, realizamos el cálculo de la L-Norm de cada uno de los términos tanto en la consulta como en la colección de documentos que se nos introducido como result. Con esto, calculamos la longitud coseno. Por último, hacemos uso del .sort, el cual organiza todo teniendo como clave al peso de la longitud coseno.

**2.5.-Permuterm**

Para hacer un permuterm, como bien hemos visto en teoría, es necesaria crear un índice que guarde las posibles rotaciones que tiene un término. Este índice es necesario enlazarlo con el término original. Esto lo hemos conseguido en el make\_permuterm, en el cual hemos ido rotando la palabra, creando así el self.ptindex() necesario. También hemos tenido en cuenta los wildcard queries, que aparecen cuando hay un \*. La rotación termina cuando este símbolo este en el final de la palabra. En el get\_permuterm primero hemos hecho una división del término en dos, posteriormente le hemos añadido el símbolo $. Mediante una búsqueda recursiva binaria, hemos buscado dentro de self.ptindex los términos

**2.6.- Paréntesis**

Es, a groso modo, una ampliación del searcher. Utilizamos en el mismo algoritmo que utilizamos en este. La única diferencia es que, a partir de ahora, introducimos la capacidad de hacer consultas complejas utilizando paréntesis. Por tanto, debemos de tener en cuenta las subconsultas que se encuentran encerradas por los paréntesis y por que conector lógico están siendo afectadas (AND, OR, NOT).