Informe del proyecto de SAR

Alumnos del grupo del proyecto:

- **Luis Alberto Álvarez Zavaleta**
- **Diego Ramos Nebot**
- **Vicent Descals Carbonell**
- **David Arnal García**

1. Funcionalidades extra implementadas en el proyecto.

Las funcionalidades extra que se han decidido implementar en el proyecto han sido:

- *Stemming*
- *Multifield*
- Búsquedas posicionales
- *Permuterm*
- Paréntesis
- Funcionamiento conjunto de las funcionalidades extra

2. Descripción y justificación de las decisiones de implementación realizadas.

En este apartado, se describirán aspectos relevantes sobre la implementación de algunos métodos destacados dentro del código del proyecto.

Index_dir*:** En este método se comprueba con el argumento ****args qué funcionalidades extra son activadas y, dependiendo de las que sean, se realizan las llamadas correspondientes a cada método para crear sus respectivos índices.

***Index_file*:** El objetivo de este método es indexar el contenido de un fichero. El primer paso es obtener el contenido de cada *field* de la noticia para, a continuación, *tokenizar* cualquier field excepto el *field* de *date*. Lo que se hace a continuación es la indexación a través de *posting lists*. En este momento, se pueden distinguir dos casos: el primero, en el que se hace uso de la función extra de búsquedas posicionales, y el segundo, en el que se hacen unos pasos similares, pero sin tener esta función adicional activada. En ambos casos, se recorre el contenido del campo correspondiente a sus *tokens*. Si se encuentra el *token* en el contenido, para cada *token,* se comprueba si se

encuentra dentro de alguna noticia y en el diccionario de *tokens* de *field* y, si no es así, se actualizan los índices. Si no se encuentra el *token* en el contenido, en el primer caso, se guarda la posición en la noticia donde ha sido encontrado y, en el segundo caso, se guarda su número de ocurrencias en dicha noticia.

***Make_stemming*:** Para cada campo que esté indexado en *self.index*, y para cada *token*, se crea su *stem*, para luego hacer su indexación en un índice diferente llamado *self.sindex*.

***Make_permuterm*:** Para cada campo que esté indexado en *self.index*, y para cada token, se crea su lista *permuterm* asociada, para luego hacer la indexación de cada elemento de la lista en un índice diferente llamado *self.ptindex*.

***Show_stats*:** En este método se muestran por pantalla todas las estadísticas referentes a la indexación. Dependiendo de las funcionalidades extra utilizadas, la información visualizada será más o menos amplia.

***Solve_query*:** Para la implementación de este método, se ha decidido dividirlo en dos partes, el *solve_query* original y otro método nuevo llamado *mini_query*. En el primero, se realiza el tratamiento de la consulta, es decir, se va reduciendo la consulta original a consultas más pequeñas de manera recursiva (en el caso de contener paréntesis) hasta quedar una consulta básica que se le pueda pasar a *mini_query* para que la resuelva y devuelva la *posting_list* resultante. En este método se ha tenido en cuenta que las consultas pueden incluir paréntesis o búsquedas posicionales. En el segundo (*mini_query*) método, se resuelve cualquier consulta básica con todas las posibles combinaciones de operadores lógicos.

Mini_query: Se ha decidido crear este método como recomendación del profesor para poder resolver las consultas básicas de forma eficiente,

controlando todas las casuísticas básicas que se han podido encontrar en una consulta. Este método puede recibir cómo parámetros tanto *posting lists* cómo terminos, ya que al inicio se convierten todos a sus *posting lists* asociadas.

***Get_posting*:** Aquí se comprueba si el término que se pasa contiene algún comodín, de ser así, llamamos a *get_permuterm* para que lo resuelva. Por otra parte, se comprueba si se le pasan diversos términos, donde se llamaría a *get_positionals*. En cualquier otro caso, se recupera la *posting_list* asociada a ese término del *self.index*.

***Get_positionals*:** Para una lista de términos se comprueba que el primero se encuentra dentro del índice y campo correspondiente y, si es así, se obtiene el conjunto de noticias en las que aparece y sus posiciones. A continuación, se comprueba que el siguiente término también esté indexado, que aparece en la misma noticia que el primer término y que aparece en la posición consecutiva. Este proceso se repite para todas las posiciones de todas las noticias en las que aparece el primer término. Finalmente, se eliminan los repetidos y se ordena.

***Get_stemming*:** Obtiene el *stem* del término y, para cada *token* del *stem,* busca su *posting_list* asociada sin incluir repetidos.

***Get_permuterm*:** Para cada elemento del índice permuterm (*self.ptindex)*, se comprueba que esté formado por el término original más el comodín y, si es así, para cada *token,* se obtiene su *posting_list* sin incluir repetidos.

Operadores lógicos: Para todos ellos se han realizado las implementaciones vistas en clase, a excepción del *and_posting*, en el que se ha implementado con *skipping lists* para conseguir una mayor eficiencia.

Snippet: Para la implementación de este método nos hemos guiado por la segunda propuesta que aparece en el boletín del proyecto. Esto es, para cada término que aparece en la consulta, se busca la primera ocurrencia donde aparece en la noticia y se consigue un pequeño fragmento de texto para imprimirlo por pantalla.

3. Descripción del método de coordinación utilizado por los miembros del grupo en la realización del proyecto.

El método que se ha usado para la realización del proyecto ha sido la edición de código de forma **colaborativa** a través de ***Visual Studio Code***. Para los **métodos** que requerían **menor tiempo de implementación**, **cada uno** se **ha hecho** cargo de **la implementación del mismo**. Por otra parte, para los métodos en los que hacía falta una **implementación más laboriosa**, se ha intentado reunir a **todos los miembros del equipo** para **elaborar el código colaborativamente**, trabajando en algunas ocasiones, en dos subgrupos.

4. Descripción de la contribución al proyecto de cada miembro del grupo.

Los métodos básicos se realizaron colaborativamente por todos los miembros donde cada miembro se centró en algunos métodos específicos y, a continuación, se revisaron todos los métodos por todos los integrantes del equipo.

Por otra parte, en los métodos que requerían de un mayor tiempo de implementación, se ha intentado trabajar el grupo entero para hacer el código colaborativamente, y cuando no se ha podido, se ha dividido en grupos de dos personas, donde cada uno se ha encargado de implementar las funcionalidades extras en el *indexer* y en el *searcher*, y dichos subgrupos del

equipo trabajaron en ellos. Estos subgrupos, cuando estaban operativos, estaban compuestos por los miembros David y Luis, y Vicent y Diego.

Por último, todos los miembros contribuyeron en la realización de la memoria y en la revisión definitiva del código a entregar.

5. Opiniones y dificultades del proyecto.

En la parte referente a la programación, la mayor dificultad que se ha tenido ha sido en la implementación opcional de combinar todas las funcionalidades extra en el método de *solve_query*, concretamente la combinación de las búsquedas posicionales con las demás.

Por otra parte, otro de los problemas que hemos encontrado para realizar el proyecto ha girado en torno a su coordinación y, es que, aunque para los métodos de menor calibre no ha habido gran problema debido a que se han podido realizar colaborativamente, cada uno encargándose de algunos en específico y revisando todos juntos todos los métodos una vez terminados, para la realización de los métodos de mayor tamaño, aunque gran parte de ellos se ha intentado realizar colaborativamente, en ciertos momentos, cada uno intentó resolver parte de ellos por su cuenta, lo que dificulta la coordinación, por lo que, para resolver esto, se decidió dividir el equipo en subgrupos para resolver este tipo de problemas.