MEMORIA
PROYECTO PRÁCTICAS

Sergi Albiach Caro
Manel Angresola Navarro
Stéphane Díaz-Alejo León
Antonio Martínez Leal

ÍNDICE

1.-INTRODUCCIÓN

Para la realización de este proyecto se nos requería completar una serie de métodos en el archivo SAR_lib.py para ofrecer las funcionalidades esperadas al indexador y recuperador de noticias.

Además de este funcionamiento básico, se comenta la posible implementación de determinados métodos para añadir funcionalidades extra al sistema. Aunque en principio se requieren solamente cuatro, nuestro equipo ha decidido realizar las siguientes cinco:

- Stemming.
- Multifield.
- Permuterm.
- Paréntesis.
- Ranking.

Durante el proceso de implementación se ha tenido en cuenta la eficiencia de los métodos y se han codificado para que así sea, añadiendo las librerías necesarias cuando ha sido pertinente.

Cada método está comentado en castellano o valenciano para su mejor comprensión a la hora de revisar el código y entender el sentido de las variables utilizadas.

Cada miembro del equipo ha trabajado individualmente y se ha encargado del correcto funcionamiento de sus propios métodos. Asimismo, el resto de miembros del equipo han verificado su exactitud en las reuniones semanales y han realizado los comentarios pertinentes sobre la correspondiente implementación y sus posibles mejoras.

2.-DECISIONES DE IMPLEMENTACIÓN

2.1.-PARTE OBLIGATORIA

En esta parte se distinguen dos funciones principales: la indexación de documentos y la recuperación de documentos mediante una query. Ambas están distribuidas a lo largo del archivo SAR_lib.py en diferentes métodos bien definidos.

En primer lugar, la parte de indexación corresponde a index_file y index_dir, métodos donde se crea el indexador y las diferentes posting lists de los términos del vocabulario.

Por otra parte, los métodos restantes se encargan de solventar las queries mediante el uso de las posting lists creadas anteriormente. Para conseguirlo se aplican diferentes operaciones de conjuntos (OR, AND, MINUS) que posteriormente permite presentar estadísticas de los resultados.

Para la correcta y eficiente realización de esta parte se han tomado las siguientes decisiones:

minus_posting()
Se optó por completar el método opcional de minus_posting. Este método facilita enormemente la implementación de reverse_posting y proporciona una estructura clara.

index_file()
En este método se indexan todos los archivos, identificando a cada documento y notícia de forma única partiendo del número 0, incrementándose en una unidad cada vez que se visite un documento o una notícia de forma respectiva. Para cada noticia se obtiene el campo artículo y se crea su posting list, ya que de haberse realizado posteriormente en métodos como get_posting, esto hubiera repercutido negativamente en los tiempos resultantes y hubiera seguido una estructura más caótica.

solve_query()
Para resolver la query se ha decidido implementar este método mediante una llamada a un método auxiliar recursivo meu_solve(), añadiendo un “OR” con una lista vacía en la primera llamada al método auxiliar para que no surjan errores en la recursión. Desde este momento se procesa la query normalizada con query.lower(). Se tomó la decisión de implementarlo de forma recursiva porque nos resultó más intuitivo y más fácil de hacer.

solve_and_show()
Se ha decidido implementar dos formas de mostrar la información dependiendo de si hay que mostrar los snippets o no, tal y como se muestra en las consultas de ejemplo. Sin la opción -N se muestra un resumen compacto de la noticia, pero si se añade -N se desglosa la información por líneas para tener una vista más clara de cada apartado.

Para el cálculo de los snippets se ha tenido en cuenta la primera aparición de cada término, y se ha puesto en contexto mediante la concatenación de las 3 palabras anteriores y las 3 posteriores. Si dos términos seguidos están a una distancia menor de 3 palabras, se ha solapado y juntado en un solo fragmento.

2.2.-PARTE OPCIONAL

En cuanto a la parte opcional se distinguen en nuestro proyecto cinco ampliaciones diferentes. No se realizó la implementación de posicionales ya que conlleva una gran cantidad de modificaciones que se extienden por todo el proyecto. Ante tales dificultades, consideramos más rentable realizar las demás propuestas y hacer solo una opción más extra para intentar asegurar la puntuación máxima.

Así pues, a continuación se describen las decisiones tomadas respecto a las ampliaciones opcionales:

Multifield
Para la ampliación de multifield se ha modificado la estructura de self.index, añadiendo las claves de la lista de campos a indexar, haciendo que en el caso de no usar multiterm, se añade solamente la clave de artículo. De esta forma, cuando se necesite acceder a la posting list de un campo, accederemos a la clave de ese campo en self.index.

Stemming
En cuanto a la ampliación de stemming, se ha decidido realizar un diccionario con los stems por clave y directamente las posting lists con las noticias por valor. Para que mejorara la eficiencia temporal, se ha optado por añadir las noticias a este diccionario a la vez que se indexa, siempre que la opción de stemming esté activada y ciertas condiciones, que reducen las llamadas al costoso método self.stemmer.stem(), se cumplan. Con esto se amplía el coste espacial, pero la reducción de tiempo se puede percibir claramente.

Por otro lado, a la hora de hacer una query, simplemente se genera el stem del término, se busca en el diccionario y se devuelve la posting list asociada.

Paréntesis
Para la ampliación de paréntesis, se ha implementado un método betweenParenthesis() para obtener una consulta que se encuentre entre dos paréntesis y resolverla posteriormente.

Permuterm
Para permuterm se procede a obtener el comodín utilizado para más tarde realizar diferentes acciones según el carácter empleado. Su implementación también sirve como base para el desarrollo de pterms, un método utilizado por rankings. De igual manera, se mejora la eficiencia temporal haciendo uso de la misma práctica explicada en stemming.

Ranking

En la ampliación del ranking se ha decido utilizar un pesado de noticias basado en la multiplicación de una función local y una función global de pesado. En nuestro caso, un pesado log y idf respectivamente. El pesado log se ha implementado teniendo en cuenta la frecuencia del término en la noticia, pero en su respectivo campo, ya que pensamos que casa mejor con la implementación multifield. Por otro lado, la función idf se ha implementado para quitar peso a aquellos términos que aparecen en más noticias. Para poder realizar correctamente el pesado en conjunto con las otras ampliaciones se han tenido que generar métodos y atributos específicos para saber a qué términos hacen referencia la query, como podría ser con stemming o permuterm.

3.-MÉTODO DE COORDINACIÓN

Para coordinar el código implementado hemos usado la herramienta Git, almacenando nuestro código en un repositorio privado de GitHub. Hemos usado las funcionalidades de *push* y *pull* para sincronizar nuestro código, usando la herramienta *merge* en los casos en que hayan ocurrido conflictos.

Hemos encontrado una variedad de uso de herramientas a la hora de usar Git, como por ejemplo la interfaz GitKraken, GitHub Desktop, Visual Studio y desde la terminal.

Se realizó una primera reunión para dividir entre los miembros del equipo los métodos a implementar, dejando por escrito esta división del código en una carpeta de Google Drive, que también se utilizó para redactar la presente memoria.

Una vez repartido el código, se realizaron reuniones semanales utilizando la herramienta Discord para hacer un seguimiento del avance del proyecto y para resolver las dudas que pudieran haber surgido durante la implementación de los métodos asignados, así como para poner en común los correos intercambiados con los profesores.

4.-CONTRIBUCIÓN AL PROYECTO

4.1.-PARTE OBLIGATORIA

La parte obligatoria se ha dividido de la siguiente forma:

- Sergi Albiach Caro:
    - solve_query(self, query, prev={}).
    - solve_and_show(self, query).
- Manel Angresola Navarro:
    - index_dir(self, root, **args).
    - index_file(self, filename).
    - show_stats(self).
- Antonio Martínez Leal:
    - get_posting(self, term, field='article').
    - reverse_posting(self, p).
- Stéphane Díaz-Alejo León:
    - and_posting(self, p1, p2).
    - or_posting(self, p1, p2).
    - minus_posting(self, p1, p2).

4.2.-PARTE OPCIONAL

La parte opcional ha sido dividida por ampliaciones de la siguiente forma:

- Sergi Albiach Caro: Paréntesis.
- Manel Angresola Navarro: Multifield.
- Antonio Martínez Leal: Permuterm.
- Stéphane Díaz-Alejo León: Stemming y Ranking.

5.-VALORACIÓN

En general consideramos que este proyecto nos ha ayudado considerablemente a asentar los conceptos estudiados en la parte teórica de la asignatura. También nos ha ayudado a ejercitar nuestro cerebro y mejorar nuestras habilidades de Python.

No obstante, hemos echado en falta un poco más de detalle en la guía respecto a cómo funcionan varios métodos conjuntamente. Por ejemplo, en un primer momento pensamos que permuterm y stemming se podían aplicar simultáneamente y, hasta la realización de los tests, no nos dimos cuenta de que no era así.

Por otro lado, la sección de preguntas frecuentes nos ha parecido realmente útil y se agradece que se facilitase una semana más de tiempo y haber tenido a disposición a nuestro profesor vía email, ya que nos ha resolvido muchas dudas.