PROYECTO DE PRÁCTICAS DE SAR

Trabajo en grupo (grupos de **4** personas).

Objetivo:

El proyecto consiste en la implementación en python de un sistema de indexación y recuperación de noticias. El alumno deberá desarrollar dos aplicaciones distintas: la primera (SAR_indexer.py) extraerá las noticias de una colección de documentos alojados en un directorio, las indexará y guardará en disco los índices creados; la segunda (SAR_search.py) leerá los índices y recuperará aquellas noticias relevantes para las consultas que se realicen.

La nota máxima será de 1,5 puntos. Las aplicaciones deberán contar con unas funcionalidades mínimas que se puntuarán en total con un máximo de 0,75 puntos. Opcionalmente, se podrán ampliar las funcionalidades para obtener mayor nota, hasta un máximo de 0,75 puntos adicionales.

Entrega: se abrirá una tarea en PoliformaT para subir el código del proyecto. Las entregas las realizará un único miembro de cada grupo. Se pueden hacer resubidas, en ese caso se evaluará la última entregada. Tanto en TODOS los ficheros fuente como al entregar la práctica se debe identificar a TODOS los miembros del grupo.

Evaluación: habrá 2 sesiones de evaluación.

Funcionalidades básicas (0,75 puntos):

Indexador (SAR_indexer.py):

Funcionalidades:

- Aceptará dos argumentos de entrada: el primero el directorio donde está la colección de noticias y el segundo el nombre del fichero donde se guardará el índice.
- Procesará los documentos y extraerá las noticias: eliminar símbolos no alfanuméricos (comillas, sostenidos, interrogantes,...), extraer los términos (consideraremos separadores de términos los espacios, los saltos de línea y los tabuladores). No se deben distinguir mayúsculas y minúsculas en la indexación.
- A cada documento se le asignará un identificador único (docid) que será un entero secuencial.
- A cada noticia se le asignará un identificador único. Se debe saber cada noticia a que documento pertenece y que posición ocupa dentro de él.
- Se deberá crear una fichero invertido accesible por término. Cada entrada contendrá una lista con las noticias en las que aparece ese término.
- Toda la información necesaria para el recuperador de noticias se guardará en un único fichero en disco.

Recomendaciones de Implementación:

- Una versión esquemática del algoritmo del indexador podría ser:

```
mientras hay_documentos:
    doc ← leer_siguiente_documento()
    docid ← asignar_identificador_al_doc()
    mientras hay_noticias_en_doc:
        noticia ← extraer_siguiente_noticia()
        newid ← asignar_identificador_a_la_noticia()
        noticia_limpia ← procesar_noticia(noticia)
        para termino en noticia_limpia:
            añadir_noticia_al_postings_list_del_termino(termino, newid)
```

- Se debería tener un diccionario de documentos además del fichero invertido. El diccionario de documentos puede ser una tabla hash accesible por docid o una lista donde el docid indique la posición que la información del documento ocupa en la lista.

- Para almacenar la información de las noticas existen dos opciones: a) una tabla hash donde a partir del newid podamos obtener el documento donde está la noticia y la posición relativa o simplemente que el identificador de la noticia sea una tupla (docid, pos).

- El fichero invertido puede ser una tabla hash implementada como un diccionario de python, indexado por término y que haga referencia a una lista con los newid asociados a ese término.

- La mejor forma de guardar los datos de los índices en disco es utilizar la librería ***pickle*** que permite guardar un objeto python en disco. Si quieres guardar más de un objeto, puedes hacer una tupla con ellos, *(obj1*, *obj2*, …, *objn)*, y guardar la tupla. Consulta la práctica del mono infinito.

Recuperador de noticias (SAR_searcher.py):

Funcionalidades:

- Aceptará uno o dos argumentos de entrada. El primero será siempre el nombre del fichero que contiene los índices. Si se le proporciona una consulta como segundo argumento resolveré la consulta y finalizará. Si sólo se le pasa un argumento, el programa entrará en un bucle de petición de consulta y devolución de las noticias relevantes hasta que la consulta esté vacía.

- La búsqueda se hará en el cuerpo de las noticias. Las noticias relevantes para una consulta serán aquellas que contengan todos los términos de la misma (búsqueda binaria).

- Se debe permitir utilizar AND, OR y NOT en las consultas. El orden de evaluación de las conectivas (orden de prelación de las operaciones) será de izquierda a derecha.

  Ejemplo: la consulta "*term1 AND NOT term2 OR term3*" deberá devolver las noticias que contienen "*term1*" pero no "*term2*" más las que contienen "*term3*".

  Se deben implementar los algoritmos de merge de postings list vistos en teoría.

- La presentación de los resultados se realizará en función del número de resultados obtenidos:

  - Si sólo hay una o dos noticias relevantes. Se mostrará la fecha, el titular, las keywords y todo el cuerpo la o las noticias.

  - Si hay entre 3 y 5 noticias relevantes. Se mostrará de cada noticia la fecha, el titular, las keywords y un *snippet* del cuerpo de la noticia que contenga los términos buscados. Si no se hace búsqueda por el cuerpo de la noticia se mostrarán las primeras 100 palabras.

  - Si hay más de 5 noticias relevantes. Se mostrará la fecha, el titular y las keywords de las 10 primeras en una única línea por noticia.

En todos los casos se mostrará el nombre de los ficheros que contienen las noticias y se informará al usuario del número total de noticias recuperadas como último resultado mostrado.

Recomendaciones de Implementación:

- Un *snippet* de un termino es una subcadena del documento que contiene el termino y un contexto por la izquierda y derecha. Prueba diferentes tamaños de contexto.

Funcionalidades ampliadas (hasta 0,75 puntos):

Para obtener la máxima puntuación, además de las funcionalidades básicas, se deberán implementar correctamente al menos cuatro de las siguientes funcionalidades extra:

- Permitir el uso de paréntesis, "(" y ")", para modificar la prelación de los operadores lógicos.

  Ejemplo: la consulta "*term1 AND NOT (term2 OR term3)*" deberá devolver las noticias que contienen "*term1*" pero no "*term2*" ni "*term3*".

- Permitir *opcionalmente* la realización de stemming de las noticias y las consultas. Se necesitará añadir un parámetro adicional al recuperador de noticias. La funcionalidad de stemming se activará mediante del argumento "-s" en la llamada al searcher.

- Añadir índices adicionales para el titular de la noticia, la categoría y la fecha. En las consultas se podrán utilizar los prefijos "title:", "article:", "summary:", "keywords:" y "date:" junto a un término para indicar que ese término se debe buscar en el índice de los titulares, el cuerpo, el resumen, las keywords o la fecha. Si no se indica nada el término se buscará en el índice del cuerpo de la noticia (article). Se debe permitir en la misma consulta mezclar busquedas sobre diferentes índices.

  Ejemplo: "title:messi valencia" debería recuperar las noticias donde aparezca "messi" en el titular y "valencia" en el cuerpo de la noticia.

- Permitir la búsqueda de varios términos consecutivos utilizando las dobles comillas. Esto hace necesario el uso de postings list posicionales.

  Ejemplo: buscar *"fin de semana"* encontraría sólo los documentos donde los tres términos aparecen de forma consecutiva, mientras que buscar *fin de semana* encontraría todos los documentos en los que aparecen los tres términos sin importar la posición.

- Devolver los documentos ordenados en función de su relevancia utilizando para ello una distancia entre el documento y la consulta. La puntuación de ordenación de los documentos deberá aparecer en los resultados. El criterio de ordenación (la distancia) debe estar clara y tener sentido.

- Permitir la búsqueda con comodines (wildcard queries).

  Ejemplo: buscar *"S*dney"* encontraría las noticias que contenga términos que comiencen por "S" y terminen en "dney". Sólo se permite un comodín por palabra.

  Se necesita implementar índices **permuterm**. De todas formas no os compliquéis mucho con la estructura/eficiencia del índice: una lista ordenada por permuterm pude dar buenos resultados.

Todas las funcionalidad extra implementadas deben funcionar simultáneamente, teniendo en cuenta que:

- Aunque esté activada esa opción de stemming, esta no se aplica a las búsquedas posicionales. Con la opción de stemming activada, la consulta 'día AND "semana"' realizará stemming en día pero no en semana.
- En las búsquedas posicionales no se podrán utilizar comodines. La consulta "fin de sem*a" no está permitida.

Algunas dudas (FAQ):

- ¿Cómo será la evaluación?

Consistirá en utilizar los mismos programas python subidos a la tarea para realizar consultas sobre documentos parecidos a los que se proporcionan para hacer los programa (por ejemplo: noticias de otros meses/años pero con el mismo formato), ver si funciona y qué ampliaciones están soportadas. También preguntaremos cuestiones sobre el funcionamiento para valorar la comprensión y la autoría del código entregado a cualquier miembro del equipo.

- ¿Hay que procesar los documentos con alguna biblioteca?

Los documentos serán ficheros json. Cada fichero contendrá las noticias correspondientes a un día o una semana.

- ¿Las ampliaciones han de ser acumulativas?

Para obtener la máxima puntuación sí, las ampliaciones deben ser acumulativas. En el fichero de ayuda aparecerán algunas ejemplos de combinaciones. Sobre todo para que podáis comprobar si funciona bien.

- ¿Cómo se generan los snippets?

La descripción da libertad para ello. Algunas propuestas serían :

A)

1) Trabajar a nivel de palabras (el cuerpo de la noticia como lista de palabras).
2) Sacar para cada término su primera ocurrencia (lo que devuelve el método index)
3) Quedarse con las posiciones mínimo y máximo de los índices anteriores
4) Sacar un fragmento de texto (método join) entre las posiciones anteriores +- un pequeño valor (2 o 3, por ejemplo) para incluir algo de contexto.

Esto puede dar snippets muy largos si hay términos al inicio y al fin de la noticia. Otra opción sería:

B) Sacar de cada término (su primera ocurrencia en el documento, para simplificar) un snippet poniendo dicho término con un contexto antes y después. Opcionalmente se pueden unir segmentos que se solapen. Esta opción es "ligeramente" más compleja que A).

Existen otras opciones... en todo caso no hace falta respetar ni las mayúsculas ni los saltos de línea del documento original. Se puede trabajar con los textos normalizados.

- ¿Se pueden usar los conjuntos python en lugar de los algoritmos de unión de posting lists?

NO, para unir los posting lists se deben utilizar los algoritmos vistos en teoría, el AND y el OR de dos posting list. De esta forma se consigue un coste linial y estás practicando lo visto en la asignatura.

- ¿Se puede tener un diccionario inverso para la versión de stemming?

Para la ampliación de stemming se permiten estas 2 opciones, si te has planteado alguna otra no dudes en consultarlo:

* Tener un diccionario que va de un stem a la lista de palabras que han dado dicho stem. Este diccionario se puede generar al final a partir de las claves o keys del diccionario inverso. Por ejemplo, si solamente estas tres palabras dan el stem "quij": quijote, quijada, quijotesco, el diccionario auxiliar tendrías una entrada:

clave "quij" valor lista ["quijada", "quijote", "quijotesco"]

La forma de usar este diccionario auxiliar es:

+ obtener el stem de los términos de la query

+ para cada stem, unir todas las noticias de los términos asociados a ella (en el diccionario auxiliar)

+ usar esas uniones como las posting list en la versión sin stemming (el and implícito de todas ellas)

* Generar otro diccionario inverso donde los términos son stems. Esto ocupa más memoria que la opción anterior.

- Qué pasa si el número de noticias no coincide con el del pdf de ayuda?

La mayoría de casos esto es debido a que en la normalización se reemplazan los símbolos no alfanuméricos por la cadena vacía. Hay queries donde aparece "Valencia" y en una noticia se

habla del partido "Valencia-Sevilla", si se sustituye "-" por "" quedaría "ValenciaSevilla" y luego no lo encuentra por "Valencia".

Si no es ese el fallo ponte en contacto con el profesor de prácticas.

- ¿Cuál es la mejor forma de hacer los cálculos del AND OR NOT en la versión obligatoria?

En el enunciado se explica que una consulta con AND, OR, NOT se realiza "como si" se procesara de izquierda a derecha. Por ejemplo, si tenemos:

term1 AND term2 AND NOT term3 OR NOT term4 AND term5

debe dar lo mismo que si tuviese esta precedencia (todo la misma) y asociatividad (izquierda a derecha):

(((term1 AND term2) AND NOT term3) OR NOT term4) AND term5

pero claro, AND y OR son conmutativas, así que nada os impide sacar primero term5 y luego un and con el resto... es un ejemplo con el que mostrar que hay formas de optimizar el coste, si bien a este nivel tampoco es vital que os esforcéis en tener algo supereficiente.

- ¿Cómo combinar consultas posicionales y stopwords?

En este proyecto no se eliminan stopwords

- Además de los archivos “SAR_indexer.py” y “SAR_searcher.py”, ¿se pueden tener más ficheros que contengan funciones o clases compartidas por los 2 ficheros?

Se puede tener (si lo necesitas) un único fichero llamado “SAR_library.py” como biblioteca de funciones.

- Si implementamos funcionalidades extra, ¿tienen que funcionar solo si se pasan ciertos argumentos o tienen que funcionar siempre?

Para que funcione el stemming se le debe indicar con el parámetro "-s" en el searcher. Las demás funciones extra son "implicitas".