PROYECTO DE PRÁCTICAS DE SAR

Trabajo en grupo (grupos de 2 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 1 punto. Opcionalmente, se podrán ampliar las funcionalidades para obtener mayor nota, hasta un máximo de 0,5 puntos adicionales.

Entrega: hasta el 24 de Mayo a las 17.00 horas utilizando la Tarea correspondiente en PoliformaT. 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: Se establecerán sesiones de evaluación en las que los alumnos deberán probar el proyecto con una colección de documentos diferente.

Funcionalidades básicas (1 punto):

Indexador (SAR_indexer.py):

Funcionalidades:

- Requiere dos parámetros de entrada: el primero es el directorio donde está la colección de noticias y el segundo es el nombre del fichero donde se guardará el índice.
- Procesa los documentos y extrae 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 asigna un identificador único (docid) que será un entero secuencial.
- A cada noticia se le asignará un identificador único. Se debe saber cada noticia a qué documento pertenece y que posición ocupa dentro de él.
- Se debe crear un fichero invertido accesible por término. Cada entrada contendrá una lista con las noticias en las que aparece ese término.
- Todas las estructuras de datos necesarias para el recuperador de noticias se debe guardar 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:

- Acepta un único parámetro de entrada (el fichero que contiene los índices) y entra 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).

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

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

    - Si hay entre 3 y 5 noticias relevantes. Se mostrará el titular de cada noticia y un *snippet* del cuerpo de la noticia que contenga los términos buscados.

    - Si hay más de 5 noticias relevantes. Se mostrará el titular de las 10 primeras.

  En todos se debe informar al usuario del número total de noticias recuperadas como último resultado mostrado.

Recomendaciones de Implementación:

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

Funcionalidades ampliadas (hasta 0, 5 puntos):

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

- 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.

- Añadir índices adicionales para el titular de la noticia, la categoría y la fecha. En las consultas se podrán utilizar los prefijos "headline:", "text:", "category:" y "date:" junto a un término para indicar que ese término se debe buscar en el índice de los titulares, el cuerpo, la categoría o la fecha. Si no se indica nada el término se buscará en el índice del cuerpo de la noticia (text). Una consulta puede contener peticiones de términos sobre diferentes índices.

  Ejemplo: "headline: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.

- Permitir la búsqueda con tolerancia. Se necesita implementar índices **permuterm**.

Ejemplo: buscar *"S*dney"* encontraría las noticias que contenga términos que comiencen por "S" y terminen en "dney".

Todas las funcionalidades extra implementadas deben funcionar simultáneamente.

FAQ:

- ¿Cómo será la evaluación?

Consistirá en utilizar los mismos programas python subidos a la tarea para realizar consultas sobre documentos parecidos a enero y a mini_enero (por ejemplo: noticias de otros meses/años), ver si funciona y qué ampliaciones están soportadas. También preguntaremos cuestiones sobre la implementación para valorar la comprensión y la autoría del código entregado.

- ¿Hay que procesar los documentos con alguna biblioteca de xml o similar?

Se podría pero no hace falta. Un simple split por <DOC> (o por </DOC>) ya rompe los documentos en noticias y posteriormente se pueden extraer los distintos campos con el método index de string.

- ¿Las ampliaciones han de ser acumulativas?

Para obtener la máxima puntuación sí, las ampliaciones deben ser acumulativas. En el fichero de ayuda aparecen algunas combinaciones 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.

- ¿Qué pasa si el número de noticias para enero o mini_enero no coincide con el del pdf de ayuda?

En 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".

- ¿Hay que tratar los errores de formato?

Puedes suponer que los documentos son ficheros con el formato sgml de los ejemplos de enero y mini_enero.

Las consultas con AND, OR, NOT puedes suponer que están correctamente escritas. Igual ha de ocurrir con los textos entre comillas.

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

En el enunciado 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