Prácticas Recuperación de Información.
Grado en Ingeniería Informática.

P1. Ejercicio 1

Indexador para un sistema de desktop search

Estudie y pruebe el código
http://lucene.apache.org/core/9_4_2/demo/src-html/org/apache/lucene/demo/IndexFiles.html

P1. Ejercicio 2

Buscador para un sistema de desktop search

Estudie y pruebe el código
http://lucene.apache.org/core/9_4_2/demo/src-html/org/apache/lucene/demo/SearchFiles.html

En los dos ejercicios anteriores no es necesario que estudie y pruebe la indexación con embeddings (-knn_dict) y la búsqueda semántica (-knn_vector), es decir ejecute esos programas sin esas opciones y omítalas en el estudio y en la adaptació de IndexFiles que se pide en P1.

P1. Ejercicio 3.

Se debe adaptar **IndexFiles** para que sea multihilo y con otras funcionalidades que se detallan. IndexFiles es el núcleo de un sistema de Desktop Search, por tanto el documento, o retrieval unit, es un archivo. Observe que esto no es apropiado en otros sistemas de RI, donde una archivo puede contener muchos documentos que es necesario extraer con un parser. **Se proporciona** también un **ejemplo de un pool de threads** que se puede adaptar para esta práctica.

-Se indexarán lo campos ya indexados por el código original (**path, modified, contents**) de la misma manera que en el código original.
Además se indexarán los siguientes campos que además deben ser todos *stored* para poder ver su contenido al navegar en la pestaña de documentos de Luke: **hostname** y **thread** que identifican el host y thread que se encargaron de la indexación de este documento (pueden obtenerse de estas u otras formas: InetAddress.getLocalHost().getHostName(), Thread.currentThread().getName()); **type** (regular file, directory, symbolic link, otro); **sizeKb** con el tamaño en kilobytes del fichero; **creationTime**, **lastAccessTime** y **lastModifiedTime** con la conversión a string de los objetos FileTime de Java correspondientes; **creationTimeLucene**, **lastAccessTimeLucene** y **lastModifiedTimeLucene** con los strings de los objetos FileTime correspondientes pero en el formato Lucene. Para esto último primero hay que convertir el objeto FileTime a un objeto Date de Java. A continuación con el método dateToString de la clase DateTools de Lucene se convierte el objeto Date de Java en un string para almacenar en el campo correspondiente del índice. Todo este proceso es necesario para que posteriormente las fechas puedan ser reconocidas por Lucene, en particular en queries sobre rangos de fechas.
http://lucene.apache.org/core/9_4_2/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description

-Para la clase principal **IndexFiles** Los argumentos -index, -create y -update serán los mismos que en el código original IndexFiles y con el mismo comportamiento, otros argumentos de la línea comando son:

-El argumento -numThreads int, para indicar el número de hilos. Si no se indica esta opción, se usarán por defecto tantos hilos como el número de cores que se puede obtener con Runtime.getRuntime().availableProcessors().

-La opción -docs del código original de IndexFiles se usa también para indicar donde residen los archivos a indexar. Se indexarán los archivos de las carpetas que cuelgan de la carpeta docs, ignorando los archivos que residen en la carpeta docs directamente. Es decir, se puede suponer que de la carpeta docs cuelgan carpetas y que en estas carpetas cuelgan archivos y otras subcarpetas, y todo esto es lo que hay que indexar. Se crearán índices parciales para indicar el resultado de indexar cada carpeta de primer nivel que cuelga de docs y todo lo que cuelga de esa carpeta, y que se deben finalmente fusionar en el índice indicado en la opción -index. Lo más cómodo es indexar cada subcarpeta de primer nivel con un hilo pero si algún estudiante quiere hacerlo de otra manera debe conseguir los mismos requirimientos funcionales y no funcionales. Las carpetas para los índices parciales se almacenarán en el mismo nivel que la carpeta para el índice final pero con sufijos que las distingan. Lo mejor sería que el sufijo indicase el nombre de la carpeta de primer nivel asociado pero no se obliga.
También es necesario que cada hilo informe por pantalla inmediatamente antes e inmediatamente después de procesar las entradas de una carpeta con mensajes del tipo «Soy el hilo xxx y voy a indexar las entradas de la carpeta yyy» «Soy el hilo xxx y he acabado de indexar las entradas de la carpeta yyy». Si hay dudas sobre como el código de IndexFiles procesa las carpetas, véase éste u otro tutorial sobre el tema:
https://docs.oracle.com/javase/tutorial/essential/io/walk.html

-La opción -depth n indica la profundidad hasta la que se indexa. Si n=0 no se indexa nada. Si n=1 se indexan los archivos que cuelgan de las carpetas de primer nivel que cuelgan de la carpeta docs y nada mas, y así sucesivamente. Si la opción no se indica, no se aplica, es decir, no se aplica límite de profundidad en la indexación.

-La opción -contentsStored, indicará que el campo **contents** además de tokenizado e indexado tal y como está en el código original, debe ser almacenado (stored).

-La opción -contentsTermVectors, indicará que el campo **contents** debe almacenar Term Vectors.

Habrá un archivo config.properties que residirá en la carpeta src/main/resources del proyecto y debe ser cargado y la lista de propiedades leída por los métodos de la clase Properties de Java (métodos tipo load, get, set). Esas propiedades serán:

-La propiedad onlyFiles indica que se indexen sólo los archivos con las extensiones indicadas en la variable onlyFiles del archivo de configuración.

-La propiedad notFiles indica que no se indexen los archivos con las extensiones indicadas en la variable notFiles del archivo de configuración. Si no se indica esta opción, por defecto se indexan todos los archivos.

Si no se indican las opciones onlyFiles notFiles, por defecto se indexan todos los archivos. Las propiedades onlyFiles y notFiles son exclusivas, y de aparecer ambas en config.properties se aplica

la primera que aparece.

-La propiedad onlyLines m, indica que para el campo contents se considerarán solo las líneas 1 a m del archivo.

Ejemplo de config.properties

```
notFiles= .out .mp4 .mov .wmv .avi
onlyLines=5
```

Además de la clase principal IndexFiles, debe haber otras clases principales. La clase principal **TopTermsInDocs** llevará un argumento -index path, donde se le indica una carpeta con un índice Lucene, un argumento -docID int1-int2, donde se le indica que para los documentos con docID en ese rango debe obtener los términos de ese documentso, su tf y su idf, y presentar el top n que se le indica en un argumento -top n, ordenados por (raw tf) x idflog10. El resultado mostrado por pantalla y en un archivo que se indicará en el argumento -outfile path, mostrará para cada documento su docId y los top n terms con su tf, df y tf x idflog10.

La clase principal **RemoveDuplicates** -index path, crea otro índice eliminando los duplicados del índice contenido en la carpeta path. Se entiende que los duplicados son documentos Lucene con el mismo contenido del campo contents. Debe proponerse una solución tanto para el caso de que el campo contents fue almacenado como si no lo fue, en este caso podrá añadir a la indexación realizada por IndexFiles lo que crea conveniente. Se visualizará la ruta del índice original con el número de documentos y número de términos que contiene. La clase llevará un argumento -out path, donde se le indica la ruta para almacenar el índice sin duplicados y se visualizará también el número de documentos y número de términos que contiene el índice sin duplicados.

Para todas las funcionalidades de la práctica se valorará la calidad y eficiencia de las soluciones. Todas las funciones deben ser probadas exhaustivamente antes de su defensa y funcionar correctamente. Puede entregarse la práctica sin alguna funcionalidad indicándolo en la entrega. La calificación dependerá del número de funcionalidades correctas y de la calidad de la implementación y eficiencia de las mismas.

Entrega P1

-LAS PRÁCTICAS SON EN EQUIPO. LOS EQUIPOS TIENEN QUE CONFORMARSE CON ALUMNOS DEL MISMO GRUPO DE PRÁCTICAS.
-EN EL CASO DE COPIA DE PRÁCTICAS, TODOS LOS ALUMNOS IMPLICADOS PERDERÁN LA NOTA TOTAL DE PRÁCTICAS.

Tenéis que crear un proyecto Maven **`mri-indexer`** y entregaréis por GitHub Classroom el archivo **pom.xml** y la carpeta **src** sin incluir archivos no necesarios para la entrega en la carpeta src.

En el proyecto deben aparecer tres clases con método main() que se corresponden con las partes de la práctica y con nombres **IndexFiles , TopTermsInDocs , RemoveDuplicates,** de forma que se puedan construir un *runnable jar* con cada parte.
-En las defensas parciales y final, el comportamiento de la práctica tiene que ser correcto y debe ser

eficiente y se pedirán cambios que deberán implementarse en el aula y horario de prácticas. Cualquier miembro del equipo de prácticas debe responder a cualquier aspecto de la defensa, y también se pedirá que cada uno implemente una variante distinta de la práctica. Por tanto aunque el trabajo es en equipo cada miembro debe conocer al detalle lo realizado por el compañero. Si esto no es así, debe advertirse antes de la defensa para no perjudicar al equipo.
-Si se detecta **copia en prácticas** se aplicará lo establecido en las normas de la asignatura.
-Al principio de la defensa final o parcial también se comprobará que se defiende la práctica entregada, importando el proyecto desde Github y lanzándolo.