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_8_0/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_8_0/demo/src-html/org/apache/lucene/demo/SearchFiles.html

Los dos primeros ejercicios son sólo para estudiar código entregado en la distribución de Lucene y puede adaptar lo que considere necesario para la prácticas. De momento 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 de momento omítalas en el estudio. Si hay dudas sobre como el código de IndexFiles recorre las carpetas, véase éste u otro tutorial sobre el tema:
https://docs.oracle.com/javase/tutorial/essential/io/walk.html

P1. Ejercicio 3.

Se debe construir y entregar un proyecto Maven **mri-webindexer** con una clase principal **WebIndexer**. Será una aplicación mutihilo que procesa archivos de texto .url que contienen listas de URLs (sólo URLs del tipo http:// y https://), una URL por línea, y que se encuentran en src/test/resources/urls. Cada hilo se ocupa de leer y procesar un archivo .url. El procesado de cada archivo consiste en descargar las páginas, parsearlas e indexarlas. **Se proporcionó** también un **ejemplo de un pool de threads** que se puede adaptar para esta práctica.

Para descargar las páginas se puede usar el cliente HTTP de Java 11+ (java.net.http.HttpClient) y debe conseguirse una operación similar a la conseguida con wget -i xxx.url para cada URL. Sólo es necesario procesar URLs con código de respuesta 200 OK, y deben establecerse timeouts de conexión y de respuesta apropiados. **Opcionalmente** pueden procesarse redirecciones http (códigos 3xx). Las páginas descargadas se almacenarán en local, cada una en un archivo con el mismo nombre que la URL pero omitiendo el prefijo http:// o https://, y con una extensión .loc.

Usando las funcionalidades básicas de librería Jsoup pueden extraerse el título (title) y el cuerpo (body) de la página descargada y almacenar el contenido, sin etiquetas html, en un archivo .loc.notags con el mismo prefijo que tiene el .loc que contiene la página en local. El título será la primera línea de este archivo y el resto el cuerpo. Los archivos .loc.notags serán indexados por Lucene.

El índice tendrá un documento Lucene asociado a cada página web con los siguientes campos. Los campos **path,** y **contents** indexados de la misma manera y opciones que en el código de IndexFiles,

en este caso referidos a cada archivo .loc. Además se indexarán los siguientes **campos que 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()); **locKb** con el tamaño en kilobytes del fichero .loc; **notagsKb** con el tamaño en kilobytes del fichero .loc.notags; **creationTime**, **lastAccessTime** y **lastModifiedTime** con la conversión a string de los objetos FileTime de Java correspondientes y referidos al archivo .loc; **creationTimeLucene**, **lastAccessTimeLucene** y **lastModifiedTimeLucene** con los strings de los objetos FileTime correspondientes pero en el formato Lucene y referidos al archivo .loc. 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_8_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description
Los campos **title** y **body** para el título y cuerpo de la página ya parseados por la librería Jsoup, es decir, la primera línea y el resto del contenido del archivo .loc.notags.

La clase principal **WebIndexer** tendrá los siguientes argumentos.

-Con -index INDEX_PATH, se indica la carpeta donde se almacenará el índice
-Con -docs DOCS_PATH, se indica la carpeta donde se almacenan los archivos .loc y .loc.notags
-Si se usa -create, el índice se abre con CREATE, por defecto con CREATE_OR_APPEND
-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().
-Con -h, cada hilo informará del comienzo y fin de su trabajo: «Hilo xxx comienzo url yyy» «Hilo xxx fin url yyy»
-Con -p, la aplicación informará del fin de su trabajo: «Creado índice zzz en mmm msecs»
-Con -titleTermVectors, se indicará que el campo **title** debe almacenar Term Vectors.
-Con -bodyTermVectors, se indicará que el campo **body** debe almacenar Term Vectors.
-Con -analyzer Analyzer, se indicará que se use uno de los Analyzers ya proporcionados por Lucene, por defecto se usará el Standard Analyzer.

Opcionalmente 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 onlyDoms indica que se procesen sólo las URLs con los dominios indicadas en la variable onlyDoms del archivo de configuración.
-Cuantas otras propiedades considere interesantes para una mejor configuración del proyecto.

Ejemplo de config.properties

```
onlyDoms= .uk .es .com
```

Además de la clase principal WebIndexer, debe haber otras clases principales para procesar el índice creado con WebIndexer. La clase principal **TopTermsInDoc** llevará un argumento -index path, donde se le indica una carpeta con un índice, un argumento -field campo, donde se le indica un campo del índice y un argumento -docID int, donde se le indica que para el documentos con ese

docID debe obtener los términos de ese documento y campo, 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. **Opcionalmente**, en lugar del argumento -docID, puede indicar esa información con el argumento -url url, indicando la url de la página que se corresponde con ese documento. La clase principal **TopTermsInField**, llevará un argumento -index path, donde se le indica una carpeta con un índice, un argumento -field campo, donde se le indica un campo del índice y debe obtener los términos de ese campo en el índice, ordenarlos por df, y presentar el top n que se le indica en un argumento -top n, tanto por pantalla como en el archivo que se le indica con -outfile path.

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. Puede obtenerse la máxima puntuación sin implementar aquellas partes que marqué con **Opcionalmente** en este enunciado, si el resto de funcionalidades son correctas y eficientes.

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-webindexer` 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. Habrá una fecha final para la entrega y defensa pero pueden pedirse defensas parciales de lo que se informará.

En el proyecto deben aparecer tres clases con método main() que se corresponden con las partes de la práctica y con nombres **WebIndexer , TopTermsInDoc , TopTermInField,** de forma que se pudiera construirse un *runnable jar* con cada parte, aunque estos runnable jars no son parte de la entrega y en la defensa sólo se pedirá ejecutar las clases principales desde la línea comando o desde el IDE (según vuestra preferencia).
-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.