RI Practica 1 2021 Enunciado

Recuperación da Información (Universidade da Coruña)

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/8_1_1/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/8_1_1/demo/src-html/org/apache/lucene/demo/SearchFiles.html

P1. Ejercicio 3.

Se debe extender **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.

-Además de los campos ya indexados por el código original (**path, modified, contents**), se indexarán los siguientes campos: **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()); **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/8_1_1/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description

-Para la clase principal **IndexFiles** Los argumentos -index y -update serán las mismas que en el código original IndexFiles, otros argumentos de la línea comando son:

-openmode openmode (el open mode será append, create, o create_or_append) y actuarán con la semántica de Lucene para estos modos, no es necesario hacer nada más en este sentido.

-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 ignora. En su lugar habrá un archivo de configuración config.properties con una variable docs para indicar las carpetas donde residen los archivos que se deben indexar. Este fichero incluirá otras claves de configuración que se detallan más adelante.

-La propiedad -partialIndexes indica que se creen índices para el resultado de indexar cada carpeta indicada en docs y que se deben finalmente fusionar en el índice indicado en la opción -index. Si no se indica la opción -partialIndexes, no se pueden crear índices parciales, y todos los hilos deben trabajar concurrentemente para producir el índice indicado en el argumento de la opción -index por la línea comando (observad que la documentación de IndexWriter asegura que sus instancias son thread safe). Las carpetas para los índices parciales se especifican en el archivo de configuración con la variable partialIndexes.

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

-onlyTopLines m, -onlyBottomLines n. Si alguna de éstas propiedades está en el archivo de configuración, en el campo **contents** se indexarán soló las m primeras o/y n últimas líneas del archivo. Si ninguna está presente, en **contents** se indexa todo el contenido del archivo.

El archivo config.properties 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. Ejemplo de config.properties

```
docs= /home/smith/workspace /home/smith/apuntes/RI /home/smith/apuntes/DS /home/smith/papers
partialIndexes= /home/smith/WS_index  /home/smith/RI_index  /home/smith/DS_index  /home/smith/papers_index
onlyFiles= .java .c .txt .odt .doc .pdf
onlyTopLines=5
onlyBottomLines=2
```

Además de la clase principal IndexFiles, debe haber otras clases principales. La clase principal **StatsField** llevará un argumento -index y un argumento -field donde se le indicará la carpeta de un índice y el nombre de un campo y devolverá las estadísticas de ese campo que se obtienen con la clase CollectionStatistics de Lucene. Si no se le indica la opción -field devolverá las estadísticas de todos los campos del índice.

La clase principal **WriteIndex** llevará un argumento -index donde se le indicará la carpeta de un índice y volcará los campos del índice en un archivo de texto plano cuya ruta se indicará con la opción -outputfile

La clase principal **BestTerms** llevará los argumentos -index donde se le indicará la carpeta de un índice, -docID donde se le indica un número de documento Lucene, -field donde se le indica el nombre de un campo, -top donde se le indica un entero n y -order donde se le indicará de manera exclusiva tf, df o tfxidf, y devolverá los top n terms de ese campo y documento ordenados por tf, df o tf x idflog10 por pantalla o en archivo cuya ruta se indica de manera opcional en el argumento -outpufile. Cada uno de los n terms se muestra en una línea indicando el valor de tf, el valor de df y el valor de tf x idflog10.

La práctica tendrá una clase principal **SimilarTerms** con argumentos -index donde se le indique la

ruta de la carpeta de un índice construido con IndexFiles, -field campo -term term donde se le indica un par <término, campo>, -top donde se le indica un entero n y -rep donde se le indica una representación que puede ser bin, tf o tfxidf. Debe tratarse con esa representación de los términos según su ocurrencia en los documentos, y para cada término obtener los top n términos mas similares según la similaridad de coseno. Se visualizará la lista de n términos, ordenados de mayor a menor similaridad.

Finalmente habrá una clase principal **TermsClusters** con los argumentos -index, -field campo, -term term, -top n y -rep con el mismo significado que en SimilarTerms, y además otro argumento k donde se le indica un número de clusters k. En esta clase se obtienen los top n términos mas similares al que se le pasa como argumento de la misma manera que en SimilarTerms, se ordenan por similaridad y se visualizan y a continuación con el algoritmo k-means se producen k clusters con esos n términos y se visualizan. Para implementar **TermsClusters** Puede adaptarse cualquier implementación de k-means que esté disponible, dando el correspondiente crédito a los autores o utilizar una de librería.

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 POR PAREJAS. LAS PAREJAS TIENEN QUE CONFORMARSE CON ALUMNOS DEL MISMO GRUPO.
-EN EL CASO DE COPIA DE PRÁCTICAS, TODOS LOS ALUMNOS IMPLICADOS PERDERÁN LA NOTA TOTAL DE PRÁCTICAS.
-Se sube al repositorio SVN antes de la fecha límite indicada. Lo que hay que subir es el proyecto con la práctica entera en la fecha límite pero se indicarán también fechas previas en las que se irán evaluando partes de la práctica. Para estas evaluaciones parciales no es necesario haber subido nada al svn.
-Para subir la práctica al SVN debe hacerse así: se crea una carpeta **P1** y se sube a esa carpeta un proyecto Java con nombre **jri-indexer** o un proyecto Maven con nombre **mri-indexer** (artifactId). SOLAMENTE puede subirse ese proyecto y en esa carpeta. SOLO UNO DE LOS ALUMNOS de la pareja de prácticas puede subir la práctica, si la suben los dos, los scripts de bajada y procesamiento detectarán copia y se invalidará la práctica. Si la subida no se hace en las condiciones (nombre de carpeta y proyecto) establecidas, los scripts de procesamiento tampoco detectarán la práctica como correcta y **no se evaluará.** En el proyecto tiene que habar seis clases con método main() que se corresponden con las seis partes de la práctica y con nombres **IndexFiles, StatsField, WriteIndex, BestTerms, SimilarTerms, TermsClusters ,** de forma que se pueda 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 de la pareja 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 se detecta **copia en prácticas** se aplicará lo establecido en las normas de la asignatura.
-Al principio de la defensa final también se pedirá que se baje el proyecto subido al SVN en la fecha límite y que las pruebas se hagan sobre esa versión para lo que debéis probar antes los

comandos correspondientes de SVN, por ejemplo:

svn checkout https://svn.fic.udc.es/grao3/ri/20-21/login/P1 -r {'aaaa-mm-dd hh:mm'}

donde la fecha corresponde a las 00:01 del día siguiente al límite de entrega. Y después de bajar el proyecto del repositorio, con Eclipse Importar el proyecto Java o Maven.