Apache Lucene

Javier Parapar, Daniel Valcarce, Alfonso Landin, Alvaro Barreiro

Information Retrieval Lab
Departamento de Computación
Universidad de A Coruña

Apache Lucene 9.0.0 Working Notes
Documentación http://lucene.apache.org/core/9_0_0/
Distribución http://archive.apache.org/dist/lucene/java/9.0.0/

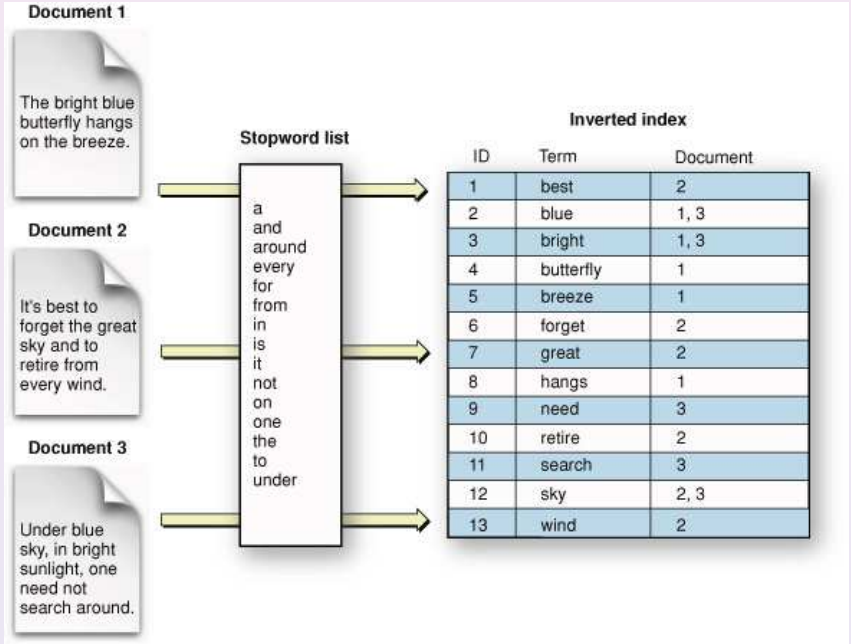
¿Qué es IR?

Information Retrieval (IR) a.k.a. Recuperación de Información

- *"El problema central en Recuperación de Información (IR) es encontrar documentos, generalmente texto, que satisfagan una necesidad de información por parte de un usuario en grandes colecciones de datos (repositorios, sistemas archivos, Web, etc.)"*
- Es un área de las ciencias de computación que se encarga de desarrollar técnicas y modelos para facilitar ante una necesidad de un usuario la búsqueda en una colección de muy gran tamaño de los documentos en los que está interesado.

Índices

- Dada la necesidad de proporcionar un método eficiente de responder a estas necesidades de información surgieron los "índices invertidos"



Búsqueda sobre Índices

- Una vez construidos los índices invertidos nos permiten obtener todos aquellos documentos que contienen los términos de la consulta del usuario simplemente operando sobre las "posting lists".
- Dependiendo del modelo de recuperación y esquemas de pesado que querramos usar a los documentos se les asignará una puntuación determinada por la fórmula de *scoring* del modelo: sim(Q,D).
- Los documentos serán posteriormente ordenados decrecientemente por dicha puntuación y la lista de documentos, ya en forma de ranking, es devuelta al usuario como respuesta a su consulta

Apache Lucene

- *"Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. Apache Lucene is an open source project available for free download."*
- Es decir, Lucene es un API que nos permite la construcción de índices invertidos y su consulta en procesos de búsqueda, i.e., Lucene es un sistema de recuperación de información.
- Lucene implementa varios modelos de recuperación de información: una versión del VSM (Vector Space Model), TFIDFSimilarity; el modelo probabilístico BM25 (el modelo por defecto en Lucene 6.x), BM25Similarity; modelos de lenguaje con suavización de Dirichlet y Jelinek-Mercer, LMDirichletSimilarity, LMJelinekMercerSimilarity.

Lucene, un poco de historia

- El proyecto fue iniciado por Doug Cutting en 1999.
- En 2001 se hizo cargo de él la Apache Software Foundation.
- Posteriormente se dividió en un proyecto principal, Lucene Java y varios subproyectos.
- Los ejemplos de estas notas están probados para la versión oficial 9.0.0.
  `http://lucene.apache.org/core/9_0_0/`
- Lucene posee *ports* a múltiples lenguajes y es usado en muchas aplicaciones y sitios web:
  `https://wiki.apache.org/lucene-java/LuceneImplementatic`
  `https://wiki.apache.org/lucene-java/PoweredBy`

Objetivos

Objetivos de las prácticas

- Saber analizar documentos para su indexación.
- Saber indexar documentos con multiples campos.
- Saber postprocesar índices con borrado, modificación y añadido de documentos.
- Saber analizar consultas del usuario sobre multiples campos.
- Saber realizar consultas sobre los índices invertidos.
- Saber interpretar las lista de resultados y su información.
- Saber alterar los pesos de documentos, términos y campos.
- Estar capacitado para de forma autónomo abordar proyectos avanzados como la modificación del modelo de recuperación de información, la integración con un crawler (Nutch), o trabajar con proyectos relacionados (Apache Solr, Apache Tika, Elastic Search, etc.).

Antes de empezar

Necesitaremos:

- Java 11 (requerido por Lucene 9.x)
- Eclipse (con soporte para Java 11)
- Colecciones de documentos.
- Binarios de Lucene (9.0.0)
- Fuentes de Lucene (en principio no)
- Maven (integrado en versiones actuales de Eclipse)

Getting started

- Bajar los binarios de Lucene (Apache Arquive, http://archive.apache.org/dist/lucene/java/9.0.0/, los binarios están dentro de los *.tgz o los *.zip)
- Crear con Eclipse un proyecto Java TestLucene9.0.0
- Crear carpeta /lib e importar a la carpeta los jar de lucene desde la carpeta del sistema de ficheros en la que se guardaron Al menos lucene-core-9.0.0.jar, lucene-analysis-common-9.0.0.jar, lucene-queryparser-9.0.0.jar, lucene-demo-9.0.0.jar
- Añadir al build path del proyecto los jar de lucene (Project Properties, Java Build Path, Libraries, Add Jars)

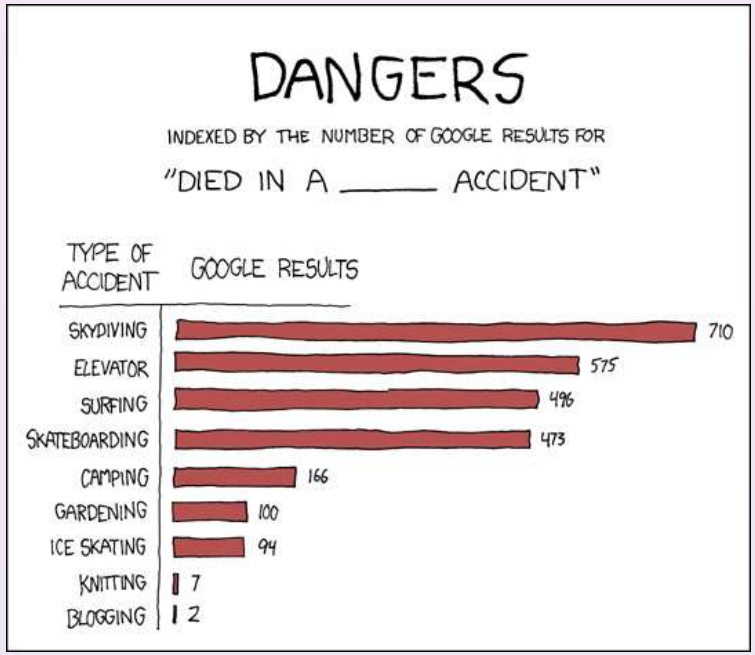
Getting started

- los API Javadocs están en la documentación: http://lucene.apache.org/core/9_0_0/
- lucene-core-9.0.0-javadoc.jar, lucene-analysis-common-9.0.0-javadoc.jar, lucene-queryparser-9.0.0-javadoc.jar, lucene-demo-9.0.0-javadoc.jar
- se pueden bajar de https://repo1.maven.org/maven2/org/apache/lucene/lucene-core/9.0.0/
- Properties, Java Build Path, Libraries, expandir el nodo de la librería asociada en cuestión y seleccionar Java Doc Location, introducir la localización donde se haya guardado el archivo javadoc correspondiente a esa librería
- Alternativa más cómoda para la gestión de proyectos Java: Maven (software project management tool) para Eclipse permite crear proyectos Maven que gestionan las dependecias declaradas en el pom.xml file

En este curso

Aprenderemos

- Indexar documentos.
- Manejo de índices.
- Búsqueda en índices.e
- Construcción de *queries*.
- Explotación de resultados.
- ... pero primero un ejemplo sencillo para tener visión global:
- SimpleDemo1.java, SimpleDemo2.java (los ejemplos se comparten por moodle)



Creando índices

- Entendemos por indexación el proceso de construir los índices invertidos (y otras estructuras con información adicional) a partir de la colección de documentos sobre la que queremos habilitar la búsqueda.
- El proceso general siempre es el siguiente:
  1. Partimos de la colección de documentos.
  2. Convertimos los documentos a formato textual.
  3. Creamos los "documentos Lucene" con los campos necesarios.
  4. Delegamos en Lucene la construcción del índice a partir de dichos documentos.

Clases clave

- A la hora de construir un índice necesitaremos tratar al menos con las siguientes clases
  - org.apache.lucene.analysis.Analyzer
  - org.apache.lucene.document.Field
  - org.apache.lucene.document.Document
  - org.apache.lucene.store.Directory
  - org.apache.lucene.index.IndexWriter
- La clase de Análisis (Analyzer) implementa una política de extracción de *index terms* a partir del texto. En principio no entraremos en detalle y trataremos con el analizador que se conoce como StandardAnalyzer (incluye un sencillo tokenizador y stopwords) org.apache.lucene.analysis.standard.StandardAnalyzer

Built-in analyzers

- WhitespaceAnalyzer: Splits tokens at whitespace.
- SimpleAnalyzer: Divides text at nonletter characters and lowercases.
- StopAnalyzer: Divides text at nonletter characters, lowercases, and removes stop words.
- KeywordAnalyzer: Treats entire text as a single token, i.e. **No tokenizado**
- StandardAnalyzer: Tokenizes based on a sophisticated grammar that recognizes email addresses, acronyms, Chinese-Japanese-Korean characters, alphanumerics, and more. It also lowercases and removes stop words.
- EnglishAnalyzer: Basically the PorterStemmer for English added to the Standard Analyzer.
- FrenchAnalyzer, SpanishAnalyzer, GermanAnalyzer, ItalianAnalyzer and more.

Definiciones (I)

- The fundamental concepts in Lucene are index, document, field and term.
  An index contains a sequence of documents.
  A document is a sequence of fields.
  A field is a named sequence of terms.
  A term is a sequence of bytes.
- The same sequence of bytes in two different fields is considered a different term. Thus terms are represented as a pair: the string naming the field, and the bytes within the field.

Definiciones (II)

- Inverted Indexing. The index stores statistics about terms in order to make term-based search more efficient. Lucene's index falls into the family of indexes known as an inverted index. This is because it can list, for a term, the documents that contain it. This is the inverse of the natural relationship, in which documents list terms.
- Types of Fields. In Lucene, fields may be stored, in which case their text is stored in the index literally, in a non-inverted manner. Fields that are inverted are called indexed. A field may be both stored and indexed.
  The text of a field may be tokenized into terms to be indexed, or the text of a field may be used literally as a term to be indexed. Most fields are tokenized, but sometimes it is useful for certain identifier fields to be indexed literally.
  See the Field java docs for more information on Fields.

org.apache.lucene.document.Field (I)

- Cada una de las secciones de un documento (e.g. título, contenido, url, etc.). Un field consta de: nombre, valor y tipo. El valor puede ser texto o numérico. A partir de Lucene 4.x los detalles de indexación del field se movieron al tipo.
- Hay ciertos tipos predefinidos por su uso común:
  - StringField, para indexar un string como un token único (i.e. lo indexa pero no lo analiza). This field turns off norms and indexes only doc IDS (does not index term frequency nor positions). This field does not store its value, but exposes TYPE_STORED as well.
  - TextField, para indexar y tokenizar un string (i.e. lo indexa y analiza). This field does not store its value, but exposes TYPE_STORED as well. **Indexa sin Term Vectors**.
  - StoredField, campo que almacena un valor (binario, int, float, string).
  - Campos Point y DocValues (ver documentación), básicamente se pueden usar para scoring de documentos, búsqueda por rangos, filtrado, ordenación de resultados y faceting.

org.apache.lucene.document.Field (II)

A field is a section of a Document. Each field has three parts: name, type and value. Values may be text (String, Reader or pre-analyzed TokenStream), binary (byte[]), or numeric (a Number). Fields are optionally stored in the index, so that they may be returned with hits on the document.
Subclasses of Field:
TextField: Reader or String indexed for full-text search
StringField: String indexed verbatim as a single token
IntPoint: int indexed for exact/range queries.
LongPoint: long indexed for exact/range queries.
FloatPoint: float indexed for exact/range queries.
DoublePoint: double indexed for exact/range queries.

org.apache.lucene.document.Field (III)

SortedDocValuesField: byte[] indexed column-wise for sorting/faceting
SortedSetDocValuesField: SortedSet<byte[]> indexed column-wise for sorting/faceting
NumericDocValuesField: long indexed column-wise for sorting/faceting
SortedNumericDocValuesField: SortedSet<long> indexed column-wise for sorting/faceting
Estos campos almacenan los valores por documento, y son apropiados para hacer sorting y faceting de forma muy rápida porque almacenan un mapping documento-valor en tiempo de indexación.

org.apache.lucene.document.Field (IV)

StoredField: Stored-only value for retrieving in summary results
FeaturedField: Field that can be used to store static scoring factors into documents.
Feature values are internally encoded as term frequencies. Putting feature queries as BooleanClause.Occur.SHOULD clauses of a BooleanQuery allows to combine query-dependent scores (eg. BM25) with query-independent scores using a linear combination. The fact that feature values are stored as frequencies will allow search logic to efficiently skip documents that can't be competitive when total hit counts are not requested in the future. This makes it a compelling option compared to storing such factors eg. in a doc-value field.
This field may only store factors that are positively correlated with the final score, like pagerank. In case of factors that are inversely correlated with the score like url length, the inverse of the scoring factor should be stored, ie. 1/urlLength.

org.apache.lucene.document.Field (IV)

- new StringField("field", "value", Field.Store.NO)
  No tokenizado, indexado. Si además se quiere almacenar:
- new StringField("field", "value", Field.Store.YES), o
  new Field("field", "value", StringField.TYPE_STORED)
- new TextField("field", value, Field.Store.NO)
  Tokenizado, indexado. Si además se quiere almacenar:
  new TextField("field", value, Field.Store.YES)

org.apache.lucene.document.Field (V)

- You can also create your own FieldType from scratch:
  FieldType t = new FieldType();
  t.setTokenized(true);
  t.setStored(true);
  t.setOmitNorms(true);
  True if normalization values should be omitted for the field. This saves memory, but at the expense of scoring quality (length normalization will be disabled), and if you omit norms, you cannot use index-time boosts. The default is false.
  t.setStoreTermVectors(true);
  t.setStoreTermVectorsPositions(true);
  t.setStoreTermVectorsOffsets(true);
  Almacena una representación del campo como vector términos pudiendo incluir en esta representación character offsets y posiciones de las palabras. El IndexReader tiene un método getTermVector para recuperar estas representaciones que necesita que se haya definido así el campo

org.apache.lucene.document.Field (VI)

t.setIndexOptions(IndexOptions value); for example:
t.setIndexOptions(IndexOptions.DOCS_AND_FREQS_
AND_POSITIONS_AND_OFFSETS);
t.freeze(); prevent further changes for this field
new Field("field", "value", t)
ver detalles de los métodos en la documentación de FieldType

org.apache.lucene.document.Field (VI)

- Index Options, describing what should be recorded into the inverted index.
  DOCS_AND_FREQS: Only documents and term frequencies are indexed: positions are omitted.
  DOCS_AND_FREQS_AND_POSITIONS: Indexes documents, frequencies and positions.
  DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS: Indexes documents, frequencies, positions and offsets.
  DOCS: Only documents are indexed: term frequencies and positions are omitted.
  NONE: Not indexed (no se puede buscar sobre ese campo).
  (Las posiciones ordinales de un token en el texto son necesarios por ejemplo para resolver phrase queries; el character offset inicial y final de un token son necesarios por ejemplos para resaltarlos en el texto -keywords in context-).

org.apache.lucene.document.Field (VII)

- Ejemplos de uso de las opciones de Field:
- Indexado, no analizado (no tokenizado), almacenado, no term vector: IDs, teléfonos, URLs, fechas, etc.
- Indexado, analizado, almacenado, term vector con posiciones y offsets: título, abstract
- Indexado, Analizado, no almacenado, term vector con posiciones y offsets: cuerpo del documento (si realmente sólo queremos hacer búsqueda y no recuperar el texto del cuerpo del documento, ni hacer resúmenes o query expansion sobre el cuerpo del documento)
- No indexado, almacenado, no term vector: campos como claves u otros sobre los que no queremos ofrecer búsqueda.
- Indexado, no analizado, no almacenado, no term vector: campos como keywords ocultas sobre las que se puede buscar pero no se almacenan en el índice directo.

Campos numéricos

En sus primeras versiones Lucene sólo indexaba texto. Posteriormente introdujo campos no numéricos para dar respuesta a necesidades de los proyectos, y algunos campos nméricos especializados como los geoespaciales. Estos campos tienen sus propias estrategias de indexación en los índices de Lucene.

org.apache.lucene.index.PointValues (I)

public abstract class PointValues extends Object
Access to indexed numeric values.
Points represent numeric values and are indexed differently than ordinary text. Instead of an inverted index, points are indexed with datastructures such as KD-trees (k-dimensional trees). These structures are optimized for operations such as range, distance, nearest-neighbor, and point-in-polygon queries. Basic Point Types

Java type	Lucene class
int	IntPoint
long	LongPoint
float	FloatPoint
double	DoublePoint
byte[]	BinaryPoint

Geospatial Point Types: Lucene has also specialized classes for location data. These classes are optimized for location data: they are more space-efficient and support special operations such as distance and polygon queries (more info in Lucene docs)

org.apache.lucene.index.PointValues (II)

Basic Lucene point types behave like their java peers: for example IntPoint represents a signed 32-bit Integer, supporting values ranging from Integer.MIN_VALUE to Integer.MAX_VALUE, ordered consistent with Integer.compareTo(Integer).
In addition to indexing support, point classes also contain static methods (such as IntPoint.newRangeQuery(String, int, int)) for creating common queries. For example:

```
//add year 1970 to document
document.add(new IntPoint("year", 1970));
// index document
writer.addDocument(document);
...
// issue range query of 1960-1980
Query query = IntPoint.newRangeQuery("year", 1960, 1980);
TopDocs docs = searcher.search(query, ...);
```

org.apache.lucene.document.Document

- Documents are the unit of indexing and search. A Document is a set of fields. Each field has a name and a textual value. A field may be stored with the document, in which case it is returned with search hits on the document. Thus each document should typically contain one or more stored fields which uniquely identify it.
  Note that fields which are not stored are not available in documents retrieved from the index, e.g. with ScoreDoc.doc or IndexReader.document(int).
- Pueden contener 1 ó n campos (Field)
- Operaciones más usadas:
  - add(IndexableField field): Adds a field to a document.
  - removeField(String name) Removes field with the specified name from the document.
  - get(String name): Returns the string value of the field with the given name if any exist in this document, or null.

org.apache.lucene.search.similarities (I)

- Vale la pena avanzar algo sobre los modelos de retrieval de Lucene
- Class TFIDFSimilarity : Implementa el modelo de espacio vectorial en Lucene. Ver los detalles en la documentación de Lucene.
  org.apache.lucene.search.similarities.TFIDFSimilarity
- A partir de Lucene 4.x, contiene implementaciones de otros modelos, en particular los muy usados BM25 (por defecto en Lucene 6.x) y Language Models: clases BM25Similarity, LMDirichletSimilarity, LMJelinekMercerSimilarity

org.apache.lucene.search.similarities (II)

- Changing Similarity
  Chances are the available Similarities are sufficient for all your searching needs. However, in some applications it may be necessary to customize your Similarity implementation. For instance, some applications do not need to distinguish between shorter and longer documents (see a "fair"similarity).

org.apache.lucene.search.similarities (III)

- To change Similarity, one must do so for both indexing and searching, and the changes must happen before either of these actions take place. Although in theory there is nothing stopping you from changing mid-stream, it just isn't well-defined what is going to happen. (Esto es así porque por ejemplo LMs necesita la Collection Frequency de los términos que Vector Space Model no usa, y esa info hay que recogerla en tiempo de indexación).
  To make this change, implement your own Similarity (likely you'll want to simply subclass an existing method, be it DefaultSimilarity or a descendant of SimilarityBase), and then register the new class by calling IndexWriterConfig.setSimilarity(Similarity) before indexing and IndexSearcher.setSimilarity(Similarity) before searching.

org.apache.lucene.search.similarities (IV)

- Ejemplo. El modelo de RI por defecto en Lucene 6.x es BM25. Para cambiar a Language Models con suavización de Jelinek-Mercer basta usar la clase LMJelinekMercerSimilarity, estableciéndola con el método setSimilarity del IndexWriterConfig al crear el índice y poniendo el valor del párametro de suavización lambda. Antes de la lanzar la búsqueda, con el IndexSearcher hay que hacer lo mismo con el método setSimilarity del IndexSearcher. Aquí se puede cambiar el valor de lambda si se quieren hacer pruebas para ver los resultados de la búsqueda con distintos valores de lambda sin necesdad de crear el índice otra vez.

org.apache.lucene.store.Directory (I)

- Representa una carpeta de un sistema de ficheros.
- No se usa el API estándar de Java para así poder crear distintas clases de directorios :
- Sobre el sistema de ficheros nativo
  - org.apache.lucene.store.FSDirectory
    Base class for Directory implementations that store index files in the file system. Para crear un directorio donde almacenar un índice:
    static FSDirectory open(Path path)
    Creates an FSDirectory instance, trying to pick the best implementation given the current environment.
    Path is a java object to locate a file in the system, can be obtained for example with Paths.get(String path)
    void close () Cierra la carpeta

org.apache.lucene.store.Directory(II)

- En memoria
  - org.apache.lucene.store.MMapDirectory
    File-based Directory implementation that uses mmap for reading, and FSDirectory.FSIndexOutput for writing
    NOTE: memory mapping uses up a portion of the virtual memory address space in your process equal to the size of the file being mapped. Before using this class, be sure your have plenty of virtual address space, ...

org.apache.lucene.index.IndexWriter (I)

- El "indexador" de Lucene.
- A él se le pasarán las unidades de indexación (Document) e irá añadiendo sus campos a los índices según lo establecido en la construcción de dichos campos (Field).
- public IndexWriter(Directory d, IndexWriterConfig conf)
  Constructs a new IndexWriter per the settings given in conf.
- public final class IndexWriterConfig
  Holds all the configuration of IndexWriter.
  IndexWriterConfig(Analyzer analyzer)
  Creates a new config with the specified Analyzer.

org.apache.lucene.index.IndexWriter (&II)

- Los métodos más usados:
  - addDocument(Document doc): Adds a document to this index.
  - addIndexes(IndexReader[] readers): Merges the provided indexes into this index.[1]
  - commit(): Commits all pending updates (added & deleted documents) to the index, and syncs all referenced index files, such that a reader will see the changes and the index updates will survive an OS or machine crash or power loss.
  - close(): Commits all changes to an index and closes all associated files.

[1] ya veremos que es un IndexReader

org.apache.lucene.index.IndexWriter (&III)

- deleteAll(): Delete all documents in the index.
- deleteDocuments(Query... queries): Deletes the document(s) matching any of the provided queries.
- deleteDocuments(Query query): Deletes the document(s) matching the provided query.
- deleteDocuments(Term... terms): Deletes the document(s) containing any of the terms.
- deleteDocuments(Term term): Deletes the document(s) containing term.

org.apache.lucene.index.IndexWriter (&IV)

- int maxDoc(): Devuelve el número de documentos no borrados más los que tienen una marca de borrado lógico. Al forzar una compactación de segmentos del índice con writer.forceMergeDeletes(), los documentos marcados como borrados se borran físicamente por lo que ya no cuentan en las estadísticas de maxDoc() ni en las de hasDeletions() o numDeletedDocs().
- int numDocs(): Devuelve el número de documentos no borrados (ni lógica, ni físicamente), es decir, todos aquellos que se pueden devolver como resultado de una búsqueda y es el que usa Lucene para calcular el docFreq o IDF de un término (TFIDFSimilariy.idf()). Por tanto maxDoc() aún cuenta los borrados lógicamento y numDocs() no.
- void updateDocument(Term term, doc): Updates a document(s) by first deleting the document containing the term and then adding the new document (it is really a delete followed by an add)

org.apache.lucene.search.CollectionStatistics(I)

This class Contains statistics for a filed in a collection.
Constructor: CollectionStatistics(String field, long maxDoc, long docCount, long sumTotalTermFreq, long sumDocFreq)
IndexSearcher.collectionStatistics(String field) Returns CollectionStatistics for a field, or null if the field does not exist (has no indexed terms)
Methods:

- long docCount() returns the total number of documents that have at least one term for this field.
- String field() returns the field name
- long maxDoc() returns the total number of documents, regardless of whether they all contain values for this field.

org.apache.lucene.search.CollectionStatistics (II)

- long sumDocFreq() returns the total number of postings for this field
- long sumTotalTermFreq() returns the total number of tokens for this field

Assume that a collection contains 100 documents, and 50 of them have "keywords" field. In this example, maxDocs is 100 while docCount is 50 for the "keywords" field. The total number of tokens for "keywords" field is divided by docCount to obtain avdl. Therefore, docCount which is the total number of documents that have at least one term for the field, is a more precise metric for optional fields

org.apache.lucene.index.IndexWriterConfig

- los métodos maxDoc(), numDocs(), hasDeletions() o numDeletedDocs(), se pueden usar en un objeto de las clases IndexReader o IndexWriter
- Métodos más usados
  - IndexWriterConfig.OpenMode getOpenMode(): Returns the IndexWriterConfig.OpenMode set by setOpenMode(OpenMode).
  - IndexWriterConfig setOpenMode(IndexWriterConfig.OpenMode openMode): Specifies IndexWriterConfig.OpenMode of the index:
    APPEND: Opens an existing index.
    CREATE: Creates a new index or overwrites an existing one.
    CREATE_OR_APPEND: Creates a new index if one does not exist, otherwise it opens the index and documents will be appended.

Segmentos y formatos de índices (I)

- Lucene indexes may be composed of multiple sub-indexes, or segments. Each segment is a fully independent index, which could be searched separately. Indexes evolve by:
  -Creating new segments for newly added documents.
  -Merging existing segments.
  Searches may involve multiple segments and/or multiple indexes, each index potentially composed of a set of segments.
- En la carpeta del índice están los archivos que contienen esta información pero no es para acceder directamente, esto es a título informativo, hay que usar el API para todo. Each segment index maintains the following:
  -Segment info. This contains metadata about a segment, such as the number of documents, what files it uses.
  -Field names. This contains the set of field names used in the index.

Segmentos y formatos de índices (II)

-Stored Field values. This contains, for each document, a list of attribute-value pairs, where the attributes are field names. These are used to store auxiliary information about the document, such as its title, url, or an identifier to access a database. The set of stored fields are what is returned for each hit when searching. This is keyed by document number. Los campos almacenados constituyen la representación directa del índice vs. la representación invertida.
-Term dictionary. A dictionary containing all of the terms used in all of the indexed fields of all of the documents. The dictionary also contains the number of documents which contain the term, and pointers to the term's frequency and proximity data.
-Term Frequency data. For each term in the dictionary, the numbers of all the documents that contain that term, and the frequency of the term in that document, unless frequencies are omitted (IndexOptions.DOCS_ONLY)

Segmentos y formatos de índices (III)

-Term Proximity data. For each term in the dictionary, the positions that the term occurs in each document. Note that this will not exist if all fields in all documents omit position data.
-Normalization factors. For each field in each document, a value is stored that is multiplied into the score for hits on that field.
-Term Vectors. For each field in each document, the term vector (sometimes called document vector) may be stored. A term vector consists of term text and term frequency. To add Term Vectors to your index see the Field constructors
-Per-document values. Like stored values, these are also keyed by document number, but are generally intended to be loaded into main memory for fast access. Whereas stored values are generally intended for summary results from searches, per-document values are useful for things like scoring factors.

Segmentos y formatos de índices (IV)

-Live documents. An optional file indicating which documents are live. Los índices pueden tener live documents y deleted documents (documentos con marca de borrado lógico) mientras Lucene no fuerza el borrado físico.
-Point values. Optional pair of files, recording dimensionally indexed fields, to enable fast numeric range filtering and large numeric values like BigInteger and BigDecimal (1D) and geographic shape intersection (2D, 3D).
Details on each of these are provided in their linked pages.

Segmentos y formatos de índices (V)

- File Names. All files belonging to a segment have the same name with varying extensions.
- The extensions correspond to the different file formats and they are described in Apache Lucene. Documentation. Reference Documents. File Formats.
  https://lucene.apache.org/core/9_0_0/core/org/apache/lucene/codecs/lucene90/package-summary.html
- Lock File. The write lock, which is stored in the index directory by default, is named "write.lock". If the lock directory is different from the index directory then the write lock will be named"XXXX-write.lock"where XXXX is a unique prefix derived from the full path to the index directory. When this file is present, a writer is currently modifying the index (adding or removing documents). This lock file ensures that only one writer is modifying the index at a time.

Ejemplos

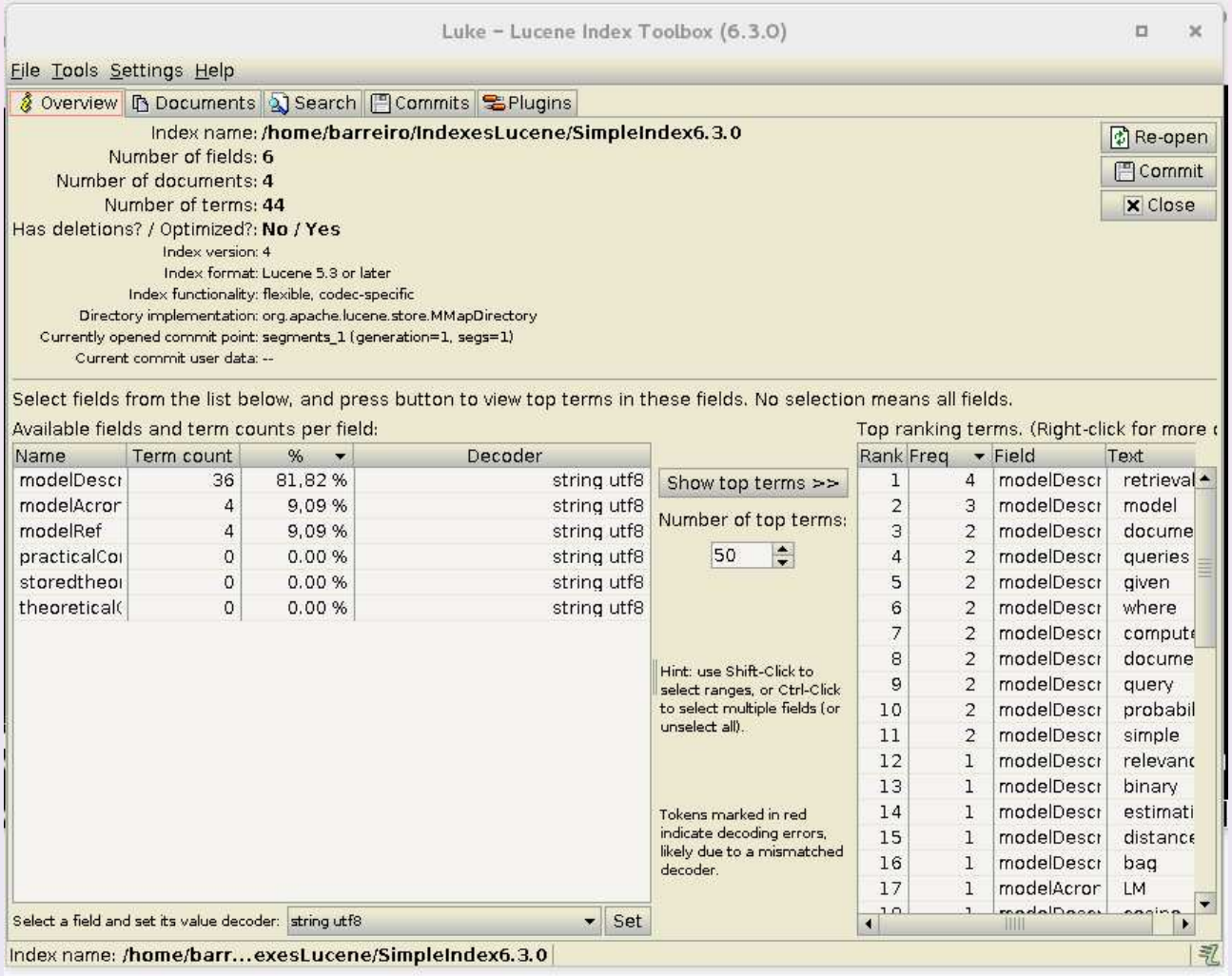
Ver ejemplos
SimpleIndexing1
SimpleIndexing2

Luke (I)

Desde Lucene 8.x Luke está integrado en Lucene. Tenéis que bajaros la distribución de Lucene correspondiente en el archivo de Apache:
https://archive.apache.org/dist/lucene/java/9.0.0/

En la carpeta luke podéis encontrar los binarios de luke que se pueden ejecutar con java -jar luceneluke-x.x.x.jar o, de forma más sencilla aún, invocar el script luke.sh

Luke (II)


Luke – Lucene Index Toolbox (6.3.0)
File Tools Settings Help
Overview Documents Search Commits Plugins
Index name: /home/barreiro/IndexesLucene/SimpleIndex6.3.0
Number of fields: 6
Number of documents: 4
Number of terms: 44
Has deletions? / Optimized?: No / Yes
Index version: 4
Index format: Lucene 5.3 or later
Index functionality: flexible, codec-specific
Directory implementation: org.apache.lucene.store.MMapDirectory
Currently opened commit point: segments_1 (generation=1, segs=1)
Current commit user data: --
Re-open
Commit
Close
Select fields from the list below, and press button to view top terms in these fields. No selection means all fields.
Available fields and term counts per field:
Name Term count % Decoder
modelDescr 36 81,82 % string utf8
modelAcror 4 9,09 % string utf8
modelRef 4 9,09 % string utf8
practicalCo 0 0.00 % string utf8
storedtheo 0 0.00 % string utf8
theoretical(0 0.00 % string utf8
Show top terms >>
Number of top terms:
50
Hint: use Shift-Click to select ranges, or Ctrl-Click to select multiple fields (or unselect all).
Tokens marked in red indicate decoding errors, likely due to a mismatched decoder.
Top ranking terms. (Right-click for more
Rank Freq Field Text
1 4 modelDescr retrieval
2 3 modelDescr model
3 2 modelDescr docume
4 2 modelDescr queries
5 2 modelDescr given
6 2 modelDescr where
7 2 modelDescr compute
8 2 modelDescr docume
9 2 modelDescr query
10 2 modelDescr probabil
11 2 modelDescr simple
12 1 modelDescr relevanc
13 1 modelDescr binary
14 1 modelDescr estimati
15 1 modelDescr distance
16 1 modelDescr bag
17 1 modelAcror LM
Select a field and set its value decoder: string utf8 Set
Index name: /home/barr...exesLucene/SimpleIndex6.3.0

org.apache.lucene.index.Term (I)

- A Term represents a word from text. This is the unit of search. It is composed of two elements, the text of the word, as a string, and the name of the field that the text occurred in. Note that terms may represent more than words from text fields, but also things like dates, email addresses, urls, etc.
  Constructores:
  - Term(String fld): Constructs a Term with the given field and empty text.
  - Term(String fld, String txt): Constructs a Term with the given field and text.
  - Term(String fld, BytesRef bytes): Constructs a Term with the given field and bytes.
    Una vez que tenemos un objeto Term podemos usarlo para hacer búsquedas, borrar documentos que contienen el término, obtener las estadísticas del término, etc.

org.apache.lucene.index.Term (II)

- Métodos
  - BytesRef bytes(): Returns the bytes of this term.
  - int compareTo(Term other): Compares two terms, returning a negative integer if this term belongs before the argument, zero if this term is equal to the argument, and a positive integer if this term belongs after the argument. (El orden es primero por field, después por text).
  - boolean equals(Object obj)
  - String field(): Returns the field of this term.
  - String text(): Returns the text of this term.
  - String toString(): Returns a string representation of the object.
- Un índice contiene términos como resultado del proceso de indexación pero podemos construir términos programáticamente para realizar búsquedas o borrar documentos. Ver ejemplo SimpleDeleting

org.apache.lucene.index.DirectoryReader

- Con una llamado al método estático open de la clase DirectoryReader podemos obtener un index reader con el que leer un índice. Con un IndexReader podemos obtener los contenidos de los campos almacenados de los documentos de un índice. Para recorrer las listas invertidas de un índice necesitaremos un Leaf Reader (Atomic Reader en versiones anteriores de Lucene), o formas alternativas que veremos más adelante.
  - open(Directory directory): Returns a IndexReader reading the index in the given Directory
  - indexExists(Directory directory): Returns true if an index exists at the specified directory.

org.apache.lucene.index.IndexReader(I)

- Una vez construido un índice, y obtenido un index reader, podremos acceder a su contenido con esta clase. Algunos métodos:
  - close(): Closes files associated with this index.
  - maxDoc(): Returns (int) one greater than the largest possible document number. Los documentos con marca de borrados lógico se incluyen en el valor maxDoc.
  - numDoc(): Returns the number of documents in this index. Los documentos con marca de borrados lógico ya no se incluyen en el valor numDoc.
  - docFreq(Term t): Returns (int) the number of documents containing the term t. This method returns 0 if the term or field does not exists. This method does not take into account deleted documents that have not yet been merged away (es decir, usa numDoc() que es lo esperado.
  - totalTermFreq(Term term): Returns the total number of occurrences of term across all documents (the sum of the freq() for each doc that has this term). This will be -1 if the codec doesn't support this measure. Note that, like other term

org.apache.lucene.index.IndexReader(II)

- getDocCount(String field): Returns the number of documents that have at least one term for this field, or -1 if this measure isn't stored by the codec. Note that, just like other term measures, this measure does not take deleted documents into account.
- public final Document document(int n): Returns the stored fields (a Document object is a set of fields) of the nth Document in this index.
- public final Document document(int docID, Set<String> fieldsToLoad): Like document(int) but only loads the specified fields.

org.apache.lucene.index.IndexReader(III)

- Terms getTermVector(int docID, String field): Retrieve term vector for this document and field, or null if term vectors were not indexed.
- abstract Fields getTermVectors(int docID): Retrieve term vectors for this document, or null if term vectors were not indexed.
- List<LeafReaderContext> leaves(): Returns the reader's leaves, or itself if this reader is atomic.
- abstract IndexReaderContext getContext(): Expert: Returns the root IndexReaderContext for this IndexReader's sub-reader tree.

org.apache.lucene.index.Document

- Métodos de la clase Document para acceder a los fields y su contenido. (Note that fields which are not stored are not available in documents retrieved from the index. Note that IndexableField is an experimental API)
  - String get(String name): Returns the string value of the field with the given name if any exist in this document, or null
  - IndexableField getField(String name): Returns a field with the given name if any exist in this document, or null
  - List<IndexableField> getFields(): Returns a List of all the fields in a document.

Ejemplos

Ver ejemplo SimpleReader1 que ilustra la lectura de los contenidos de un índice con un IndexReader y Document
Ver ejemplo CosineSimilarity que ilustra el uso del método IndexReader.getTermVector(docID, field) para obtener el tf de los términos de un documento sin tener que recorrer todo el índice como en el ejemplo SimpleReader4

DirectoryReader devuelve la visión más abstracta de un índice. Internamente Lucene utiliza leaf (atomic) readers y multireaders. Básicamente un leaf reader se asocia a un segmento y un índice puede constar de varios segmentos. Un leaf reader proporciona métodos para acceder a los fields de un índice, a los terms de un field, y a los documentos para un term. Para acceder a los contenidos de los documentos de un índice nos basta con usar un IndexReader, pero para recorrer las listas invertidas se necesita un LeafReader: ver ejemplos SimpleReader2 y SimpleReader3. Alternativamente se pueden usar las clases FieldInfos y MultiTerms (ver ejemplo SimpleReader4), esta alternativa es más cómoda para el programador pero menos eficiente en el acceso al índice.

org.apache.lucene.index.IndexReaderContext(I)

(El contenido de las siguientes slides, hasta las slides de búsqueda, lo explicaremos por los ejemplos).
A struct like class that represents a hierarchical relationship between IndexReader instances.
Campos:

- boolean isTopLevel: true if this context struct represents the top level reader within the hierarchical context
- int ordInParent: the ord for this reader in the parent, 0 if parent is null
- CompositeReaderContext parent: The reader context for this reader's immediate parent, or null if none

org.apache.lucene.index.IndexReaderContext(II)

Métodos:

- abstract List<IndexReaderContext> children(): Returns the context's children iff this context is a composite context otherwise null.
- abstract List<LeafReaderContext> leaves(): Returns the context's leaves if this context is a top-level context.
- abstract IndexReader reader(): Returns the IndexReader, this context represents.

org.apache.lucene.index.LeafReaderContext

IndexReaderContext for LeafReader instances.
Algunos métodos para un LeafReaderContext:

- List<IndexReaderContext> children(): Returns the context's children iff this context is a composite context otherwise null.
- List<LeafReaderContext> leaves(): Returns the context's leaves if this context is a top-level context.
- LeafReader reader(): Returns the IndexReader, this context represents.
- String toString()

Ver SimpleReader2 y SimpleReader3 que ilustran el uso de LeafReaderContext y LeafReader para poder acceder a los campos de un índice y a los postings de los términos de cada campo.

org.apache.lucene.index.LeafReader(II)

Podemos usar los siguientes métodos en un LeafReader r para acceder a los campos, términos y postings de un índice. (Ver más métodos en la documentación del API de Lucene)

- Fields fields(): Returns Fields for this reader.
- Terms terms(String field): This may return null if the field does not exist.
- PostingsEnum postings(Term term) Returns PostingsEnum for the specified term with PostingsEnum.FREQS.
- PostingsEnum postings(Term term, int flags) Returns PostingsEnum for the specified term.

org.apache.lucene.index.LeafReader(III)

Podemos usar los siguientes métodos en un LeafReader r para acceder a los campos, términos y postings de un índice. (Ver más métodos en la documentación del API de Lucene)

- postings (term) equivale a PostingsEnum postings(Term term, PostingsEnum.FREQS)
- otros flags: PostingsEnum.NONE, PostingsEnum.ALL, PostingsEnum.FREQS, PostingsEnum.POSITIONS, PostingsEnum.OFFSETS, PostingsEnum.PAYLOADS

org.apache.lucene.index.Terms

- Access to the terms in a specific field. Experimental API.
- Constructor: Terms(). Algunos métodos (consultar documentación API para más):
  - TermsEnum iterator(TermsEnum reuse): Returns an iterator that will step through all terms.
  - long size(): Returns the number of terms for this field, or -1 if this measure isn't stored by the codec.
  - int getDocCount(): Returns the number of documents that have at least one term for this field, or -1 if this measure isn't stored by the codec.

org.apache.lucene.index.TermsEnum

- Constructor TermsEnum(). (Experimental API). Iterator to seek (methods seekCeil(BytesRef), seekExact(BytesRef)) or step through (BytesRefIterator.next() terms to obtain frequency information (docFreq()), PostingsEnum or PostingsEnum for the current term. Algunos métodos (consultar documentación API para más):
  - int docFreq(): Returns the number of documents containing the current term.
  - BytesRef term(): Returns current term.
  - long totalTermFreq() Returns the total number of occurrences of this term across all documents (the sum of the freq() for each doc that has this term).
  - PostingsEnum postings(PostingsEnum reuse) Get PostingsEnum for the current term.
  - abstract PostingsEnum postings(PostingsEnum reuse, int flags) Get PostingsEnum for the current term, with control over whether freqs, positions, offsets or payloads are required. Payload son arrays de bytes que se pueden usar para almacenar pesos para el término, POS tags u otras cosas.

org.apache.lucene.index.PostingsEnum(I)

- public abstract class PostingsEnum extends DocIdSetIterator: Iterates through the documents and term freqs. NOTE: you must first call DocIdSetIterator.nextDoc() before using any of the per-doc methods.
  Fields to pass as flag in TermsEnum.postings(PostingsEnum, int flag)
  - ALL, to get positions, payloads and offsets in the returned enum
  - FREQS, if you require term frequencies in the returned enum.
  - NONE, if you don't require per-document postings in the returned enum.
  - OFFSETS, if you require offsets in the returned enum.
  - PAYLOADS, if you require payloads in the returned enum.
  - POSITIONS, if you require term positions in the returned enum.

org.apache.lucene.index.PostingsEnum(II)

- Allgunos métodos:
  - int freq() Returns term frequency in the current document, or 1 if the field was indexed with FieldInfo.IndexOptions.DOCS.
  - int nextDoc():
    Advances to the next document in the set and returns the doc it is currently on, or NO_MORE_DOCS if there are no more docs in the set.
  - BytesRef getPayload() Returns the payload at this position, or null if no payload was indexed.
  - int nextPosition() Returns the next position, or -1 if positions were not indexed.
  - int startOffset() Returns start offset for the current position, or -1 if offsets were not indexed.
  - int docID()
    Returns the following:
    -1 if nextDoc() or advance(int) were not called yet.
    NO_MORE_DOCS if the iterator has exhausted.
    Otherwise it should return the doc ID it is currently on.

org.apache.lucene.index.MultiTerms(I)

A partir de un IndexReader, on los métodos de las clases FieldInfos y MultiTerms se puede acceder a los campos del índice, a los términos de un campo de un índice, o directamente a las postings de un término de un campo de un índice, sin tener que obtener las hojas. Es más cómodo para el programador porque accediendo a las hojas el programador tiene que obtener las estadísticas globales a partir de las estadísticas de las hojas, pero menos eficiente tal y como expone la documentación de Lucene. En las prácticas podéis usar los métodos de la clase MultiTerms que están en la documentación de la clase y adaptar los códigos de los ejemplos para acceder a la información de los campos, términos y postings sin acceder a las hojas. Ver ejemplo SimpleReader4

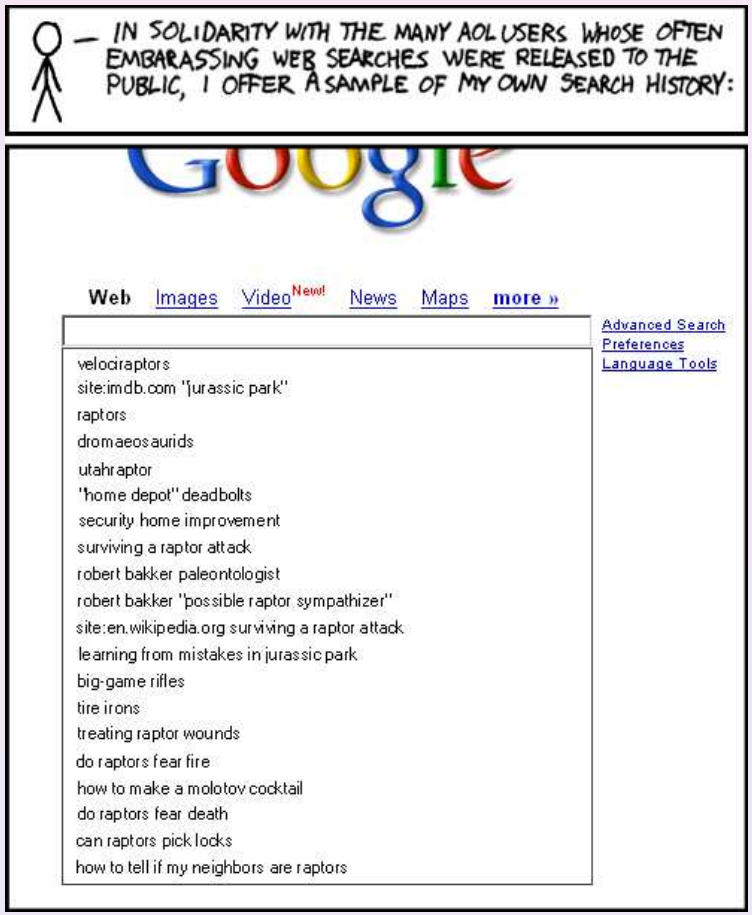
org.apache.lucene.index.MultiTerms(II)

public final class MultiTerms
extends Terms
Exposes flex API, merged from flex API of sub-segments.
WARNING: This API is experimental and might change in incompatible ways in the next release.

El ejemplo CosineDocumentSimilarity DoCosineOfDos ilustra otra forma obtener el tf de un término en un documento sin recorrer los postings basandose en
IndexReader.getTermVector(int docID, String field)

Búsqueda

- Ya sabemos construir un índice, leerlo y modificarlo.
- Ahora aprendermos a usarlos para búsqueda.
- Veremos cómo buscar (IndexSearcher, Query).
- Cómo interprentar los resultados (TopDocs).
- La sintaxis de las consultas.
- Los distintos tipos de consultas admitidas.



org.apache.lucene.search.IndexSearcher

- Una vez construido un índice, podremos buscar en su contenido con los métodos de la clase IndexSearcher.
  - search(Query query, Filter filter, int n) Finds the top n hits for query, applying filter if non-null.
  - search(Query query, Filter filter, int n, Sort sort) Search implementation with arbitrary sorting.
  - explain(Query query, int doc): Returns an Explanation that describes how doc scored against query.
  - setSimilarity(Similarity similarity): Expert: Set the Similarity implementation used by this Searcher.
  - getSimilarity(): Expert: Return the Similarity implementation used by this Searcher.
- Constructores: IndexSearcher(IndexReader r)
  Creates a searcher searching the provided index

Filter, Sort, Explanation, Similarity

- Filter: Abstract base class for restricting which documents may be returned during searching.
  Se pueden restringir con criterios de rangos de docID pero también de valores de campo y otros.
- Sort: Encapsulates sort criteria for returned hits.
  El orden por defecto es relevance pero se podría cambiar a otros como IndexOrder o por valores de campos.
- Explanation: Describes the score computation for document and query.
- Similarity: Similarity defines the components of Lucene scoring.

org.apache.lucene.search.Query(I)

- Da soporte a las posibilidades de consulta de Lucene. Algunas subclases:
  - TermQuery: un término.
  - BooleanQuery: expresiones lógicas (must, must not, should) must: AND, must not: NOT, should: el término debiera aparecer pero aunque no aparezca no se excluye el documento. Tened en cuenta que para el parser el operador por defecto es OR, es decir la query
    New York
    se parsea como
    new OR york (suponiendo que el analizador aplicado hace lowercase)
  - WildcardQuery: uso de comodines.
  - PhraseQuery y MultiPhraseQuery: búsqueda por frases. Ejemplo de PhraseQuery: "New York". Una MultiPhraseQuery permitiría por ejemplo construir esa query pero donde el segundo término pudiera ser York, Orleans o Zealand.

org.apache.lucene.search.Query(II)

- PrefixQuery: búsqueda por prefijos.
- FuzzyQuery: con Levenshtein (edit distance).
  Se puede indicar una distancia mínima entre los query terms y los matching terms.
  La distancia de edición es el número mínimo de operaciones de inserción, modificación, borrado, para pasar de un string a otro.
  universo universa universal: distancia edición(universo, universal) $= 2$
- SpanQuery: SpanFirstQuery, SpanNearQuery, SpanNotQuery, SpanOrQuery, SpanTermQuery, etc.
  Permite construir queries de proximidad con muchas más opciones que la simple PhraseQuery

org.apache.lucene.search.Query(III)

- RegexpQuery, queries con expresiones regulares.
- TermRangeQuery, devuelve documentos que satisfacen un rango de términos.
- PointRangeQuery, devuelve documentos que satisfacen un rango de valores en campos indexados como numéricos (PointValues)
  public abstract class PointRangeQuery extends Query
  Abstract class for range queries against single or multidimensional points such as IntPoint.
  This is for subclasses and works on the underlying binary encoding: to create range queries for lucene's standard Point types, refer to factory methods on those classes, e.g. IntPoint.newRangeQuery() for fields indexed with IntPoint, FloatPoint.newRangeQuery() for fields indexed with FloatPoint, etc.

org.apache.lucene.queryparser.classic.QueryParser (I)

- Constructor: QueryParser(String field, Analyzer a). Métodos (ver doc de los métodos en QueryParserBase que es la clase que extiende QueryParser):
  - parse(String query). Parses a query string, returning a Query.
  - setDateResolution(String fieldName, DateTools.Resolution dateResolution) Sets the date resolution used by RangeQueries for a specific field.
    In TermRangeQuerys, QueryParser tries to detect date values, e.g. date:[6/1/2005 TO 6/4/2005] produces a range query that searches for "date" fields between 2005-06-01 and 2005-06-04. Note that the format of the accepted input depends on the locale. A DateTools.Resolution has to be set, if you want to use DateTools for date conversion.

org.apache.lucene.queryparser.classic.QueryParser (II)

- org.apache.lucene.queryParser.classic.MultiFieldQueryParser.
  A QueryParser which constructs queries to search multiple fields. Constructors:
  MultiFieldQueryParser(String[] fields, Analyzer analyzer)
  MultiFieldQueryParser(String[] fields, Analyzer analyzer, Map$< \mathit{String}, \mathit{Float} >$ boosts)
  boosts: boosting de términos, i.e., aumenta la relevancia de documentos que contienen esos términos

org.apache.lucene.queryParser.classic.QueryParser (III)

- Métodos para MultiFieldQueryParser.
  - public static Query parse(Version matchVersion, String[] queries, String[] fields, Analyzer analyzer)
    Parses a query which searches on the fields specified.
    If x fields are specified, this effectively constructs:
    (field1:query1) (field2:query2) (field3:query3)...(fieldx:queryx)
  - public static Query parse(Version matchVersion, String query, String[] fields, BooleanClause.Occur[] flags, Analyzer analyzer)
    Parses a query, searching on the fields specified. Use this if you need to specify certain fields as required, and others as prohibited.
    Por defecto hace un OR de los campos pero se puede cambiar con los flats de BooleanClause.Occur[] que pueden ser MUST, MUST_NOT o SHOULD
    Ver ejemplo en la documentación
- Sintaxis de la query en

  http://lucene.apache.org/core/9_0_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html

org.apache.lucene.search.TopDocs

This class represents hits returned by IndexSearcher.search(Query,Filter,int) and IndexSearcher.search(Query,int).
Constructor: TopDocs(int totalHits, ScoreDoc[] scoreDocs, float maxScore).
Fields:

- scoreDocs Expert: The top hits for the query.
- totalHits Expert: The total number of hits for the query.
- org.apache.lucene.search.ScoreDoc
  - int doc Expert: A hit document's number.
  - float score Expert: The score of this document for the query.

Ejemplos

Ver ejemplo SimpleSearch