

TP - Unidad 02
Búsqueda en Archivos Invertidos (Lucene)- Parte C

Ejercicio 1

- 1.1) ¿Qué tipo de índice maneja Lucene?
- 1.2) ¿Qué es para Lucene un documento?
- 1.3) ¿Se puede buscar por un campo no almacenado en el índice? ¿Se puede mostrar la información del valor de un campo almacenado en Lucene pero fuera del índice, si se tiene un resultset de documentos (lista de docid)?
- 1.4) ¿Qué opciones de almacenamiento tiene un FieldType? ¿Existe alguna combinación no admitida de almacenamiento para un FieldType?
- 1.5) ¿Qué es un TextField? ¿Por qué no me permite almacenar en Lucene si el dato proviene de archivo?
- 1.6) Se tiene 4 documentos con un FieldType llamado “content” que no almacena fuera del índice, pero sí en el índice. Los documentos tienen la siguiente información (puede provenir de archivos txt o de un String)

Docid 0

[illegible]

Docid 1

v i d e o

Docid 2

g	a	m	e
---	---	---	---

Docid 3

G	a	m	e		v	i	d	e	o	,		r	n	
r	e	v	i	e	w					g	a	m	e	.

Mostrar gráficamente que se obtiene en el archivo invertido cuando se usan las opciones:

- **IndexOptions.DOCS**
- **IndexOptions.DOCS_AND_FREQS**
- **IndexOptions.DOCS_AND_FREQS_AND_POSITIONS**
- **IndexOptions.DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS**

- 1.7) Independientemente del almacenamiento (o no) en el índice, ¿qué almacena Lucene si se le pide al campo “content” setStore(yes)? Mostrar gráficamente cómo almacena dicha información.



Ejercicio 2

Crear un proyecto maven y que maneje las siguientes dependencias:
lucene-core, lucene-queryparser y lucene-analyzers-common

Agregar el código de Campus **IndexBuilder.java** y **Utils.java**

Bajar los 4 archivos del ejemplo: a.txt, b.txt, c.txt y d.txt en el directorio para indexación.
Ejecutar IndexBuilder y verificar que se ha generado el índice en disco.

¿Cuáles son los campos que constituyen el documento? ¿Están indizados?

Ejercicio 3

Utilizando el índice creado, realizaremos consultas con el API de Lucene. Incorporar TheSearcher.java para realizar diferentes tipos de consultas.

3.1) Probar **TermQuery** sobre los campos indicados y valores pedidos. Explicar resultados.

Valor	FieldName
-	Probar con un fieldname no existente
C:\\dropbox\\2021Q1\\Unidad02\\lucene\\docs\\a.txt (usar el path del archivo a.txt)	path
game	path
Game	content
ga	content
game,,	content
game	content

¿Qué información del índice le permite a Lucene responder a las consultas por TermQuery?

3.2) Probar **PrefixQuery** sobre los campos indicados y valores pedidos. Explicar resultados.
Cambiar Query query = new TermQuery(new Term("content", "bla"));

Por

```
Query query = new PrefixQuery(new Term("content", "bla"));
```

Valor	FieldName
game	content
ga	content
Ga	content
me	content

¿Qué información del índice le permite a Lucene responder a las consultas por PrefixQuery?



3.3) Probar **TermRangeQuery** sobre los campos indicados y valores pedidos. Explicar resultados.

Permite buscar términos pero sin tener que explicitarlos uno por uno, siempre que estos pertenezcan a un mismo rango.

Cambiar:

```
Query query = new PrefixQuery(new Term("content", "bla"));
```

Por

```
Query query = new TermRangeQuery("fieldName", BytesRefIzq, BytesRefDer, Boolean,  
Boolean );
```

Donde

BytesRefIzq: es una instancia de la clase **ByteRef** que representa un **String** para el intervalo a izquierda. Esa instancia puede generarse así: **new BytesRef(String)**

BytesRefDer: es una instancia de la clase **ByteRef** que representa un **String** para el intervalo a derecha.

Los últimos **2 booleanos** indican si el intervalo es abierto/cerrado a izquierda, y abierto/cerrado a derecha. **True** significa cerrado.

Valor	FieldName
Rango ["game", "gum"]	content
Rango ["game", "game"]	content
Rango ["game", "game")	content
Rango ["gum", "gam"]	content
Rango ("game", "gum"]	content
Rango ("gaming", "gum")	content

¿Qué información del índice le permite a Lucene responder a las consultas por **TermRangeQuery**?

3.4) Probar **PhraseQuery** sobre los campos indicados y valores pedidos. Explicar resultados.

Usar:

```
Query query = new PhraseQuery("fieldName", "word1", "word2", ...);
```

Como se observa **PhraseQuery** es un API que permite parámetro variable: uno por cada término que forme parte de la frase (secuencia contigua de tokens).



Valor (cada token en ese orden)			FieldName
store	game	content	
store,,	game	content	
game	store	content	
store game		content	
store,, game		content	
game	review	content	
game	video	game	content
game	video	review	content

¿Qué información del índice le permite a Lucene responder a las consultas por PhraseQuery?

3.5) Probar **WildcardQuery** sobre los campos indicados y valores pedidos. Usar * o bien ? para expresar varios o un carácter de matching. Explicar resultados.

Usar:

Query query = new **WildcardQuery**(new Term("content", "bl*a?"));

Valor	FieldName
g*e	content
g?me	content
g?m	content
G??e	content
*	content

¿Qué información del índice le permite a Lucene responder a las consultas por WildQuery?

3.6) Probar **FuzzyQuery** sobre los campos indicados y valores pedidos. Explicar resultados.

Usar:

Query query = new **FuzzyQuery**(new Term("content", "bla"));

Corresponde al algoritmo de Damerau-Levenshtein con máxima distancia 2 (es distancia, no similitud, o sea solo cantidad de operaciones).

Valor	FieldName
gno	content
agen	content
agem	content
hm	content
ham	content



¿Qué información del índice le permite a Lucene responder a las consultas por FuzzyQuery?

Ejercicio 4

Cuando usamos TextFields, los strings se separan en tokens para luego guardarlos separadamente en el índice.

Lucene viene equipada con varias clases que permite separar en tokens y se aplican antes de guardar en el índice.

- SimpleAnalyzer()
- StandardAnalyzer()
- WhitespaceAnalyzer()
- StopAnalyzer()=>
sw= new ArrayList<>();
sw.add("sw1");
sw.add("sw1");
stw = new CharArraySet(sw, false);
new StopAnalyzer(stw);
- EnglishAnalyzer() // optional stop words
- SpanishAnalyzer() // optional stop words
- etc

Bajar de Campus la clase **TestAnalyzer.java**.

La misma analiza con Low LevelAPI los tokens que genera Lucene en cada caso. Como verán ni siquiera estamos generando documentos, solo queremos ver qué tokens se generan en cada caso, según el Analyzer que usemos.

4.1) ¿Qué tokens se obtiene en cada caso?

```
String fieldValue=
```

```
"Estructura de datos. Y algoritmos; 2020-Q1 en eda.ita.edu";
```

Tokenizador	Tokens obtenidos
SimpleAnalyzer	
StandardAnalyzer	
WhitespaceAnalyzer	
StopAnalyzer. Probarlo por ejemplo con 2 stop words: de y	
SpanishAnalyzer	



4.2)

Probar con el siguiente CustomAnalyzer y explicar resultados

```
Analyzer analyzer = CustomAnalyzer.builder()
    .withTokenizer("standard")
    .addTokenFilter("lowercase")
    .addTokenFilter("stop")
    .addTokenFilter("porterstem") // familia de palabras
    .addTokenFilter("capitalization")
    .build();
```

4.3) Qué CustomAnalyzer habría que usar para obtener: blanco como separador de tokens (si sabemos que no hay signos de puntuación en el texto), y familia de palabras?

Aclaración:

Más allá de que StopAnalyzer está diseñado para aceptar stops words si prefieren usar el comportamiento de StandardAnalyzer, pueden hacerlo porque también acepta una lista de stop words como parámetro.

Ejercicio 5

Vamos a utilizar QueryParser conjunto con algún Tokenizador para el propio query.

Bajar el archivo TheSearcherQueryParser.java

La misma nos permitirá escribir consultas si usar cada una de las APIs anteriores, pero deberemos expresar consulta en el lenguaje de más alto nivel de Lucene. Lucene se encargará de parsear la query y mapearlo a invocaciones a las APIs.

En vez de usar

```
Query aQuery= new TermQuery(t);
```

Estaremos usando:

```
QueryParser queryparser = new QueryParser(field, new
StandardAnalyzer() ); // o el que quieran
```

```
Query aQuery = queryparser.parse(query);
```

Eso sí, las consultas deberán seguir la siguiente convención el momento de escribir el texto del query.



Query Implementation	Purpose	Sample expressions
TermQuery	Single term query, which effectively is a single word.	reynolds
PhraseQuery	A match of several terms in order, or in near vicinity to one another.	"light up ahead"
RangeQuery	Matches documents with terms between beginning and ending terms, including or excluding the end points.	[A TO Z] {A TO Z}
WildcardQuery	Lightweight, regular-expression-like term-matching syntax.	j*v? f??bar
PrefixQuery	Matches all terms that begin with a specified string.	cheese*
FuzzyQuery	Levenshtein algorithm for closeness matching.	tree~
BooleanQuery	Aggregates other Query instances into complex expressions allowing AND, OR, and NOT logic.	reynolds AND "light up ahead" cheese* -cheesewhiz

5.1) Buscar un término único (equivalente a la API TermQuery). Completar cómo se escribe dicho texto en el query.

Valor	FieldName	El query se escribe así
game	content	
Game	content	
ga	content	
game,,	content	

5.2) Buscar un término único (equivalente a la API PrefixQuery). Completar cómo se escribe dicho texto en el query.

Valor	FieldName	El query se escribe así
game	content	
ga	content	
Ga	content	
me	content	

5.3) Buscar por rango (equivalente a la API TermRangeQuery). Completar cómo se escribe dicho texto en el query.

Valor	FieldName	El query se escribe así
Rango ["game", "gum"]	content	
Rango ["game", "game"]	content	
Rango ["game", "game")	content	
Rango ["gum", "gam"]	content	
Rango ("game", "gum"]	content	
Rango ("gaming", "gum")	content	



5.4) Buscar una frase (equivalente a la API PhraseQuery). Completar cómo se escribe dicho texto en el query.

Valor (cada token en ese orden)			FieldName	El query se escribe así
store	game	content		
store,,	game	content		
game	store	content		
store game		content		
store,, game		content		
game	review	content		
game	video	game	content	
game	video	review	content	

5.5) Buscar por wildcard. Para ello se usa el símbolo * (matchean varios) o ? (matchean de a uno). No permiten comenzar el query con estos por eficiencia. Completar cómo se escribe dicho texto en el query.

Valor	FieldName	El query se escribe así
g*e	content	
g?me	content	
g?m	content	
G??e	content	
*	content	

5.6) Buscar por FuzzyQuery. Completar cómo se escribe dicho texto en el query.

Valor	FieldName	El query se escribe así
gno	content	
agen	content	
agem	content	
hm	content	
ham	content	

5.7) Buscar por expresiones booleanas. Para ello se usa AND &&, OR || y NOT -.

Valor	FieldName	El query se escribe así
Buscar por cualquiera de estas 2 palabras: store game	content	
Buscar por ambas palabras: store game	content	



Buscar por documentos que contengan palabras que no difieran en más de 2 caracteres respecto de ga o bien que contengan s??re	content	
Buscar por documentos que contengan a palabras que no difieran en más de 2 caracteres respecto a ga y que además contenga s??re	content	

5.8) ¿Son equivalente estas 2 expresiones booleanas?. Explicar resultados

content:review OR (content:game AND NOT content:yo)

(content:review OR content:game) AND NOT content:yo

Ejercicio 6

Proponer una query que permita encontrar cuando 2 palabras consecutivas pero que no se sabe en qué orden aparecen. Ej: las palabras game store

Ejercicio 7

Des comentar la línea

```
// Explanation rta = searcher.explain(query, docID);  
// System.out.println(rta);
```

Que permite obtener la explicación del ranking.

Sabiendo que si el query es de un solo término aplica la fórmula

Score(DOCi, query) =
FormulaLocal(DOCi,term) * FormulaGlobal(D, term)

Donde



$$\text{FormulaLocal}(\text{DOCi}, \text{query}) = \sqrt{\frac{\#freq(\text{term in DOCi})}{\#term \text{ existentes en DOCi}}}$$

Y

$$\text{FormulaGlobal}(\text{DOC}, \text{query}) = \frac{1 + \log_e \frac{1 + \#docs \text{ en la coleccion}}{1 + \#docs \text{ que contienen term}}}{1 + \log_e \frac{1 + \#docs \text{ en la coleccion}}{1 + \#docs \text{ que contienen term}}}$$

Salvo que sea de tipo FuzzySearch, Range, Prefix, Wildcard en cuyo caso devuelve siempre 1 a los documentos que matchean.

Explicar el valor obtenido en las siguientes consultas:

Valor	FieldName	score
game	content	
Ga*		

Ejercicio 8

Sabiendo que si el query es de multi-término aplica la fórmula (excluido NOT)

$$\text{Score}(\text{DOCi}, \text{query}) = \sum_{\text{term in query y no tiene NOT}} \text{FormulaLocal}(\text{DOCi}, \text{term}) * \text{FormulaGlobal}(\text{D}, \text{term})$$

Explicar el valor obtenido en las siguientes consultas:

Valor	FieldName	score
Game AND NOT store	content	
Game AND store	content	
Game OR store	content	
Ga* OR store	content	
Game AND game	content	
Game OR game	content	