



# Estructura de Datos y Algoritmos

ITBA 2024-Q2

## *1. API para las queries*

*1.1 TermQuery: busca un solo término*

*1.2 PrefixQuery: busca por prefijo*

*1.3 TermRangeQuery: busca por rangos*

*1.4 PhraseQuery: busca secuencia*

*1.5 WildcardQuery*

*1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2*

*1.7 BooleanQuery*

*Etc., etc., etc.*

TermRangeQuery buscar si el “término” se encuentra en el intervalo especificado. El mismo puede ser abierto/cerrado a izq, abierto/cerrado a derecha:

- [BytesRefIzq, BytesRefDer]:  
fieldName, BytesRefIzq, BytesRefDer, true, true
- (BytesRefIzq, BytesRefDer):  
fieldName, BytesRefIzq, BytesRefDer, false, false
- [BytesRefIzq, BytesRefDer):  
fieldName, BytesRefIzq, BytesRefDer, true, false
- (BytesRefIzq, BytesRefDer]:  
fieldName, BytesRefIzq, BytesRefDer, false, true

# TP 2-C

## Ejer 3.3

=>TermRangeQuery



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

Realizar los siguiente cambios, re ejecutar y explicar el resultado

```
Query query= new TermRangeQuery(fieldName, .... );
```

Donde los casos a probar son si los **BytesRef** (usar **new BytesRef("string")**) del campo **content** pertenece a los intervalos (fieldName, BytesRefIzq, BytesRefDer, **boolean**, **boolean**)

Como aleternativa: `Query q = TermRangeQuery.newStringRange( fieldName, stringFrom, stringTo, boolean, boolean );`

```
["gam", "gum"]  
["game", "game" ] // equivalente a ????  
["game", "game"]  
["gum", "gam"]  
("game", "gum")  
("gaming", "gum")
```

## *1. API para las queries*

*1.1 TermQuery: busca un solo término*

*1.2 PrefixQuery: busca por prefijo*

*1.3 TermRangeQuery: busca por rangos*

*1.4 PhraseQuery: busca secuencia*

*1.5 WildcardQuery*

*1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2*

*1.7 BooleanQuery*

*Etc., etc., etc.*

# TP 2-C

## Ejer 3.4

=>PhraseQuery



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

Realizar los siguiente cambios, re ejecutar y explicar el resultado

Query query= **new PhraseQuery(fieldName, word1, word2, ... wordN );**  
**si busco la frase: word1 word2 wordN**

Donde los casos a probar son:

- Frase: "store" "game"
- Frase: "store,," "game"
- Frase: "game" "store"
- Frase: "store game"
- Frase: "store,, game"
- Frase: "game" "review"
- Frase: "game" "video" "game"
- Frase: "game" "video" "review"





Pregunta:

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

## 1. API para las queries

1.1 *TermQuery: busca un solo término*

1.2 *PrefixQuery: busca por prefijo*

1.3 *TermRangeQuery: busca por rangos*

1.4 *PhraseQuery: busca secuencia*

1.5 *WildcardQuery: busca por matching de \* o bien ?*

1.6 *FuzzyQuery: Damerau-Levenshtein con MaxEdit 2*

1.7 *BooleanQuery*

Etc., etc., etc.

# TP 2-C

## Ejer 3.5

=>WildcardQuery



```
Query query= new  
WildcardQuery(myTerm);
```

Realizar los siguiente cambios, re ejecutar y explicar el resultado

Query query= **new WildcardQuery(myTerm)**

Donde los casos a probar son:

- queryStr= "g\*e"
  - queryStr= "g?me"
  - queryStr= "g?m"
  - queryStr= "G??e"
  - queryStr= "\*"
- \* representa cualquier secuencia de caracteres inclusive vacío  
? representa un caracter cualquiera

## 1. API para las queries

1.1 *TermQuery: busca un solo término*

1.2 *PrefixQuery: busca por prefijo*

1.3 *TermRangeQuery: busca por rangos*

1.4 *PhraseQuery: busca secuencia*

1.5 *WildcardQuery*

1.6 *FuzzyQuery: Damerau-Levenshtein con MaxEdit 2*

1.7 *BooleanQuery*

Etc., etc., etc.

# TP 2-C

## Ejer 3.6

=>FuzzyQuery



```
Query query= new  
FuzzyQuery(myTerm );
```



Pregunta:

MaxEdit es operaciones, no similitud (no está normalizado).

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

Realizar los siguiente cambios, re ejecutar y explicar el resultado.

Query query= **new FuzzyQuery(myTerm )**;

Donde los casos a probar son:

- queryStr= “gno”
- queryStr= “gem”
- queryStr= “agem”
- queryStr= “hm”
- queryStr= “ham”



## 1. API para las queries

1.1 *TermQuery: busca un solo término*

1.2 *PrefixQuery: busca por prefijo*

1.3 *TermRangeQuery: busca por rangos*

1.4 *PhraseQuery: busca secuencia*

1.5 *WildcardQuery*

1.6 **FuzzyQuery: Damerau-Levenshtein con MaxEdit 2**

1.7 **BooleanQuery**

Etc., etc., etc.



Las consultas no siempre son tan puntuales...

Muchas veces se necesita combinar esas características.  
Inclusive entre varios campos...

Ej: que cierta frase aparezca en el campo “content” pero  
que no aparezca tal término.

Ej: que cierta frase aparezca en el campo “content” y  
también tal término parezca en otro campo.

Ej: que empiece con tal prefijo en el campo “content” o  
bien se parezca en otro campo indexado.



Sin duda, *BooleanQuery* parece resolver este problema.

Pero con API es tedioso usarlo porque hay que combinar varios constructores (para el AND, para el OR, para el NOT).

En vez de analizar BooleanQuery vamos a conocer la otra forma que tiene Lucene de realizar consultas sin API: QueryBuilder!

# Lucene

- *Concepto de documento, campos.*
- *Almacenamiento en Lucene: en el índice y fuera del índice*
- *Aplicaciones*
  - *IndexBuilder (creación de los documentos)*
  - *TheSearcher (búsqueda de documentos)*
- *Query:*
  - *API*
  - *QueryBuilder*
- Formas de separar en tokens
- Ranking de documentos

## 2. QueryBuilder para las queries

Lucene **define un lenguaje de consulta** y él se encarga de parsearlo y transformarlo en varias invocaciones de APIs (las mismas que vimos antes).

Resulta muy práctico, pero para poder usarlo debemos conocer dicho lenguaje. Si no lo respetamos, obtenemos error en tiempo de ejecución en el parser.

Mostraremos cada una de las invocaciones anteriores como sería con el QueryParser.

API	QueryBuilder
1.1 TermQuery	fieldName:termino
1.2 PrefixQuery	fieldName:term*
1.3 TermRangeQuery	fieldName:[start TO end]
1.4 PhraseQuery	fieldName:"term1 ... termN"
1.5 WildcardQuery	fieldName:*subterm?
1.6 FuzzyQuery	fieldName:termino~2
1.7 BooleanQuery	AND OR NOT (+ -)

EJ: Query query= queryParser.parse("+content:game -content:review");




Pregunta:

Cuando con API preguntamos por TermQuery por “Game”, ¿lo encontró? ¿Por qué?

Rta

Porque para ingresar al índice le hemos aplicado un StandardAnalyzer() que detectó tokens por espacios y símbolos de puntuación y además los insertó en minúsculas.

Si la query no es igual, no lo va a encontrar.



Y mejor aún si usamos algún Analyzer (como usamos en la construcción del índice) para que pase a minúsculas, elimine stopwords en la propia query

```
QueryParser qp = new QueryParser(null, new StandardAnalyzer());
```

Antes de comenzar a usar el QueryParser, veamos con los tipos de separación en tokens que pueden usarse...



# Lucene

- *Concepto de documento, campos.*
- *Almacenamiento en Lucene: en el índice y fuera del índice*
- *Aplicaciones*
  - *IndexBuilder (creación de los documentos)*
  - *TheSearcher (búsqueda de documentos)*
- *Query:*
  - *API*
  - *QueryBuilder*
- *Formas de separar en tokens*
- *Ranking de documentos*



Hemos usado StandardAnalyzer en la creación del índice.

¿Qué otro analyzer podríamos usar?

¿Cómo afecta el índice y la búsqueda?

Lucene viene con diferentes clases para separar en tokens:

- SimpleAnalyzer()
- StandardAnalyzer()
- WhitespaceAnalyzer()
- StopAnalyzer()=>

```
CharCharacterSet sw = StopFilter.makeStopSet("de", "y");  
new StopAnalyzer(sw));
```

- EnglishAnalyzer() // opcional stop words
- SpanishAnalyzer() // opcional stop words.
- CustomAnalyzer()
- etc

# TP 2C- Ejer 4

Bajar de Campus el código  
TestAnalyzer.java

Explicamos a continuación  
cómo se usa.

El código sirve para ver cómo hace Lucene por dentro (para inspeccionar la separación en tokens). Como verán, no estamos creando documentos, solo usando un Low Level API para ver qué tokens genera.

Probar cuáles son los tokens que genera en cada caso.

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





Pregunta:

Qué hizo SpanishAnalyzer?

Rta: busco la raíz de las palabras “stemmer algorithm”.

C.J. van Rijsbergen, S.E. Robertson and M.F. Porter, 1980 en *“New models in probabilistic information retrieval”* propusieron un algortimo para encontrar la raíz de las palabras en idioma ingles.

*Así los IR engine guardan menos letras y matchean más.*

*Si tienen curiosidad por el algoritmo en diferentes idiomas*  
<https://snowballstem.org/algorithms/>



Para usar el

- CustomAnalyzer:

```
Analyzer analyzer = CustomAnalyzer.builder()  
    .withTokenizer("standard")  
    .addTokenFilter("lowercase")  
    .addTokenFilter("stop")  
    .addTokenFilter("porterstem")  
    .build();
```




# TP 2C- Ejer 5

Vamos resolver las consultas  
con un  
QueryAnalyer (para tokenizar  
el query)  
Y un  
QueryParser


Así, ya no tenemos que invocar  
diferentes subclases...





Para rehacer las queries hechas previamente con **QueryParser**, incorporar (desde Campus) **TheSearcherQueryParser.java**.

En `QueryString` hay que colocar la query en el Lenguaje Explicado.



```
<dependency>  
  <groupId>org.apache.lucene</groupId>  
  <artifactId>lucene-queryparser</artifactId>  
  <version>7.4.0</version>  
</dependency>
```

*Rehacer las queries realizadas con API por medio de QueryParser*

*1.1 TermQuery: busca un solo término*

1.2 PrefixQuery: busca por prefijo

1.3 TermRangeQuery: busca por rangos

1.4 PhraseQuery

1.5 WildcardQuery

1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2

1.7 BooleanQuery

*Etc., etc., etc.*

API	QueryParser
<pre> queryStr="game"; Term myTerm = new Term("content", queryStr); Query query= new TermQuery(myTerm ) </pre>	<pre> queryStr="content:game"; Query query= queryparser.parse(queryStr); </pre>

## Ejercicio 5.1

Realizar los siguiente cambios, re ejecutar y explicar el resultado

- a) Cambiar queryStr= "Game";
- a) Cambiar queryStr= "ga";
- a) Cambiar queryStr= "game,";

## *1. API para las queries*

*1.1 TermQuery: busca un solo término*

*1.2 PrefixQuery: busca por prefijo*

*1.3 TermRangeQuery: busca por rangos*

*1.4 PhraseQuery: busca secuencia*

*1.5 WildcardQuery*

*1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2*

*1.7 BooleanQuery*

*Etc., etc., etc.*

API	QueryParser
<pre> queryStr="ga"; Term myTerm = new Term("content", "Game"); Query query= new PrefixQuery(myTerm ) </pre>	<pre> queryStr="content:ga*"; Query query= queryparser.parse(queryStr); </pre>

## Ejercicio 5.2

Realizar los siguiente cambios, re ejecutar y explicar el resultado

**Donde los casos a probar son:**

```

String queryStr= "game";
String queryStr= "ga";
String queryStr= "Ga";
String queryStr= "me";

```

## *1. API para las queries*

*1.1 TermQuery: busca un solo término*

*1.2 PrefixQuery: busca por prefijo*

*1.3 TermRangeQuery: busca por rangos*

*1.4 PhraseQuery: busca secuencia*

*1.5 WildcardQuery*

*1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2*

*1.7 BooleanQuery*

*Etc., etc., etc.*



API			QueryParser
Query	query=	new	queryStr="content:{gaming TO
TermRangeQuery("content",		new	gum]";
BytesRef("gaming"), new BytesRef("gum"),			Query query=
false,true )			queryparser.parse(queryStr);

### Ejercicio 5.3

Realizar los siguiente cambios, re ejecutar y explicar el resultado

```

["gam", "gum"]
["game", "game" ] // equivalente a ????
["game", "game")
["gum", "gam"]
("game", "gum"]
("gaming", "gum")

```

## *1. API para las queries*

*1.1 TermQuery: busca un solo término*

*1.2 PrefixQuery: busca por prefijo*

*1.3 TermRangeQuery: busca por rangos*

*1.4 PhraseQuery: busca secuencia*

*1.5 WildcardQuery*

*1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2*

*1.7 BooleanQuery*

*Etc., etc., etc.*

API	QueryParser
<pre>Query query= new PhraseQuery("content", "store", "game")</pre>	<pre>queryStr="content: \\"store game\\""; Query query= queryparser.parse(queryStr);</pre>

## Ejercicio 5.4

Realizar los siguiente cambios, re ejecutar y explicar el resultado

Donde los casos a probar son (adaptar al QueryBuilder):

- Frase: "store game"
- Frase: "store," "game"
- Frase: "game" "store"
- Frase: "store game"
- Frase: "store,, game"
- Frase: "game" "review"
- Frase: "game" "video" "game"
- Frase: "game" "video" "review"

## 1. API para las queries

1.1 *TermQuery: busca un solo término*

1.2 *PrefixQuery: busca por prefijo*

1.3 *TermRangeQuery: busca por rangos*

1.4 *PhraseQuery: busca secuencia*

1.5 *WildcardQuery: busca por matching de \* o bien ?*

1.6 *FuzzyQuery: Damerau-Levenshtein con MaxEdit 2*

1.7 *BooleanQuery*

Etc., etc., etc.

API	QueryParser
<pre> queryStr="g??e"; Term myTerm = new Term("content", queryStr); Query query= WildcardQuery(myTerm); </pre>	<pre> queryStr="content:g??e*"; Query query= queryparser.parse(queryStr); </pre>

### Ejercicio 5.5

Realizar los siguiente cambios, re ejecutar y explicar el resultado

Query query= **new WildcardQuery(myTerm)**

Donde los casos a probar son:

- queryStr= "g\*e"
- queryStr= "g?me"
- queryStr= "g?m"
- queryStr= "G??e"
- queryStr= "\*" // OJO NO PUEDE COLOCARSE EN PRIMERA POSICION

## 1. API para las queries

1.1 *TermQuery: busca un solo término*

1.2 *PrefixQuery: busca por prefijo*

1.3 *TermRangeQuery: busca por rangos*

1.4 *PhraseQuery: busca secuencia*

1.5 *WildcardQuery*

1.6 *FuzzyQuery: Damerau-Levenshtein con MaxEdit 2*

1.7 *BooleanQuery*

Etc., etc., etc.

API	QueryParser
<pre>queryStr="gno"; Term myTerm = new Term("content", queryStr); Query query= new FuzzyQuery(myTerm);</pre>	<pre>queryStr="content:gno~2"; Query query= queryparser.parse(queryStr);</pre>

### Ejercicio 5.6

Realizar los siguiente cambios, re ejecutar y explicar el resultado.

Donde los casos a probar son:

- queryStr= "gno"
- queryStr= "agen"
- queryStr= "agem"
- queryStr= "hm"
- queryStr= "ham"

## 1. API para las queries

1.1 *TermQuery: busca un solo término*

1.2 *PrefixQuery: busca por prefijo*

1.3 *TermRangeQuery: busca por rangos*

1.4 *PhraseQuery: busca secuencia*

1.5 *WildcardQuery*

1.6 **FuzzyQuery: Damerau-Levenshtein con MaxEdit 2**

1.7 **BooleanQuery**

Etc., etc., etc.



## Ejercicio 5.7

Buscar por cualquiera de estas 2 palabras:

content:store OR content: game

content:store content:game

content:store || content:game

Buscar por ambas palabras:

content:store AND content: game

content:store && content: game

## Ejercicio 5.8

Son equivalentes estas expresiones?:

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

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

Explicar resultados

## Aclaración

Si todas las query se hace sobre el mismo fieldName puede especificarse una sola vez (un default)

En vez de :

```
String queryStr= "content:game AND content:store";  
QueryParser queryparser = new QueryParser(null, new  
StandardAnalyzer() );  
Query query= queryparser.parse(queryStr);
```

Escribimos:

```
String queryStr= "game AND store";  
QueryParser queryparser = new QueryParser("content", new  
StandardAnalyzer() );  
Query query= queryparser.parse(queryStr);
```