



# Estructura de Datos y Algoritmos

ITBA 2024-Q2

# TP 2-C

## Ejer 2



Crear un nuevo proyecto Maven para incorporar ambos procesos.

## Agregar dependencias de Lucene al pom.xml

```
<dependency>
```

```
    <groupId>org.apache.lucene</groupId>
```

```
    <artifactId>lucene-core</artifactId>
```

```
    <version>7.4.0</version>
```

```
</dependency>
```

```
<dependency>
```


```
    <groupId>org.apache.lucene</groupId>
```

```
    <artifactId>lucene-analyzers-common</artifactId>
```

```
    <version>7.4.0</version>
```

```
</dependency>
```

- Crear directorio **lucene**
- Crear directorio **docs** dentro de lucene
- Bajar de campus los archivos
  - a.txt
  - b.txt
  - c.txt
  - d.txty dejarlos en el directorio **docs**



Para no tener sueltos en el file system a los documentos a indexar (docs) y a los archivos que genera el índice, se ha decidido mantener a ambos en un directorio prefijo común. En “docs” guardaríamos los documentos (txt, pdf, a indexar). En “index” los archivos del índice.

Ej Windows: `c:/lucene/docs` y `c:/lucene/index`

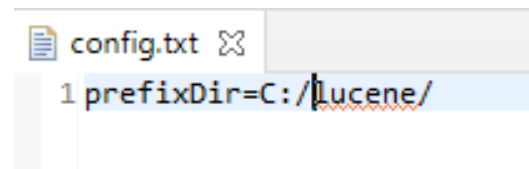
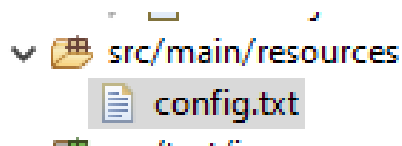
Ej: Linux/Mac

`/var/lucene/docs` y `/var/lucene/index`

## Crear archivo **config.txt** en resources

Además, para no “cablear” dicho prefijo en el código Java, vamos a utilizar un archivo de configuración donde vamos a setear en la propiedad **prefixDir** el valor prefijo que deseemos. El archivo de configuración que estará en resources se llamará **config.txt**

Agregar al proyecto mvn el archivo **config.txt** y setear **prefixDir** con el valor deseado



Bajar de Campus [Utils.java](#) y agregarla en el proyecto

Probar que la clase con alguna de las 3 opciones está OK.  
Configurar el archivo config.txt en resources.

Aclaración de la API

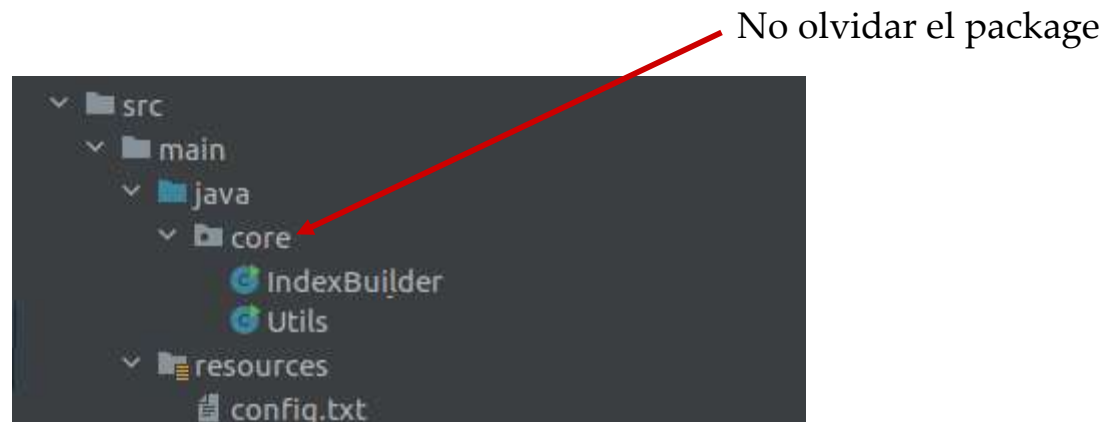
- `String Utils.getPrefixDir()`
- `List<String>` `Utils.listFilesAbsolutePath("c:/tmp",  
Arrays.asList("txt", "pdf"))`
- `List<String>` `Utils.listFilesRelativePath("docs",  
Arrays.asList("txt", "pdf"))`

Nuestro primeros documentos Lucene van a estar formados por 2 campos:

- **TextField** => “**content**” que contiene el contenido de los archivos txt que le digamos.
- **FieldType** => “**path**” que contiene el lugar físico donde están los archivos txt. No se busca en ellos, solo se lo quiere almacenar en Lucene. Idea: si consulto y Lucene me da matching quiero saber no solo que hubo X cant de documentos que matchearon sino cuáles son. Si conozco su path, podría desarrollar un front end que los muestre.



Bajar de Campus el archivo [IndexBuilder.java](#)  
Importarlo, ejecutarlo y verificar que se genera el índice.



# 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

Ya estaríamos en condiciones de escribir consultas al índice. Un término es la unidad básica que puede buscarse en un índice.

Un *Término Lucene* es una secuencia de bytes (podrían interpretarse como String, números, etc) asociada a cierto campo.

El lenguaje para escribir consultas en Lucene tiene 2 formatos:

- API Query
- Query builder (menos programática)

# 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

## *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
- 1.5 WildcardQuery
- 1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2
- 1.7 BooleanQuery

*Etc., etc., etc.*

## *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

1.5 WildcardQuery

1.6 FuzzyQuery // Damerau-Levenshtein con MaxEdit 2

1.7 BooleanQuery

*Etc., etc., etc.*



Haremos un búsqueda por [TermQuery](#) en la colección indizada.

Lo buscamos en el field indexado, o sea Content.

Bajar de Campus al archivo [TheSearcher.java](#) e incorporarlo al proyecto. Ejecutarlo.

Explicar el resultado.

# TP 2-C

## Ejer 3.1

=>TermQuery




```
Term myTerm = new  
Term(fieldName,  
queryStr);
```



Realizar los siguiente cambios, re ejecutar y explicar el resultado

- a) Cambiar `queryStr= "Game";`
- b) Cambiar `queryStr= "ga";`
- c) Volver a colocar `queryStr= "game";` y cambiar por el otro campo: `fieldName = "path";`

Al terminar asegurarse que `queryStr= "game";` y `fieldName = "content";`



Como se observa en el código, se obtiene los docIDs relevantes y ordenados por un ranking.

Iterando por dichos docID se puede mostrar al usuario los “stored fields” que se guardaron en Lucene (aunque no formen parte del índice) para indicarle qué documentos matchearon la consulta.



```
for (ScoreDoc aD : orderedDocs) {
```

```
...
```

```
    // print info about finding
```

```
    int docID= aD.doc;
```

```
    // obtain the stored fields
```

```
    Document aDoc = searcher.doc(docID);
```

```
    System.out.println("stored fields: " + aDoc);
```

```
...
```

*Pero también puede haber interés.*

```
C:\dropbox\2021Q1\Unidad02\lucene\docs\b.txt  
null
```

¿por qué null en el segundo?

```
for (ScoreDoc aD : orderedDocs) {
```

```
...
```

```
    // print info about finding  
    int docID= aD.doc;
```

```
    // obtain the stored fields  
    Document aDoc = searcher.doc(docID);
```

```
    System.out.println(aDoc.get("path"));  
    System.out.println(aDoc.get("content"));
```

```
...
```

## *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 3.2

=>PrefixQuery



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

Realizar los siguiente cambios, re ejecutar y explicar el resultado

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

**Por**

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

**Donde los casos a probar son:**

```
String queryStr= "game";
```

```
String queryStr= "ga";
```

```
String queryStr= "Ga";
```

```
String queryStr= "me";
```