Práctica 1:
Parser de documentos con TIKA.

UNIVERSIDAD DE GRANADA
E.T.S.I. INFORMÁTICA Y TELECOMUNICACIÓN

UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Recuperación de Información (2020-2021)

Daniel Bolaños Martínez
Fernando de la Hoz Moreno
Grupo 10 - Martes 11:30h

Índice

1. Introducción.

En esta práctica veremos cómo poder extraer información (texto) a partir de un documento dado, independiente del formato del archivo, paso previo para cualquier proceso de recuperación de información o análisis de textos.[1]

Se pide realizar un programa capaz de extraer toda la información que cuelga de un directorio que contenga como mínimo 10 ficheros distintos con al menos 3 formatos diferentes y que según la opción, realice las siguientes funciones:

- **-d**: Creación de una tabla con nombre, codificación, tipo e idioma de cada fichero.
- **-l**: Extracción de todos los enlaces de cada fichero.
- **-t**: Generación de un archivo CSV que contenga las ocurrencias de las palabras para cada fichero.

Antes de comenzar a analizar y parsear la lista de archivos del directorio debemos, para todas las opciones:

- Obtener la lista con los nombres de todos los ficheros contenidos en el directorio pasado por parámetro.
- Definir un objeto *tika* y establecer el límite de caracteres.

```
File directorio = new File(args[1]);
String[] archivos = directorio.list();
//Creamos un objeto Tika
Tika tika = new Tika();
//Establecemos el limite de caracteres para los strings
tika.setMaxStringLength(1000000000);
//Representa los metadatos del documento
Metadata metadata = new Metadata();
```

Los documentos que queramos analizar los debemos almacenar en el directorio *docs* dentro de la carpeta *src*.

2. Opción -d.

Para esta opción, crearemos una tabla que recopile para cada archivo del directorio, información sobre su nombre, codificación, tipo de documento y lenguaje.

Declaramos un manejador de tipo *BodyContentHandler* estableciendo el límite de caracteres y definimos los objetos para el contexto y parser (usamos *AutoParser* para crear un *parser* específico según el tipo del fichero). Estos objetos, serán pasados como parámetros a la función *parse* junto con el *FileInputStream* del fichero que estamos analizando.

A partir de los metadatos de cada archivo, podemos extraer el nombre del fichero (RESOURCE_NAME_KEY), la codificación (CONTENT_ENCODING) y el tipo (CONTENT_TYPE).[1]

Finalmente, para identificar el lenguaje declaramos un objeto *LanguageIdentifier* y le pasamos un string con el contenido en texto plano del archivo.

Nota: Usamos la función *tika.parse()* para completar algunos metadatos que aparecen como *null* usando el método *parse*.

```
//Recorremos para cada archivo del directorio
for(String file : archivos){
  File f = new File("./" + args[1] + "/" + file);
  //Establece el limite de caracteres en el constructor
  BodyContentHandler handler = new BodyContentHandler(1000000000);
  //Guarda la informacion del contexto concreto para el ContentHandler
  ParseContext parseContext = new ParseContext();
  //Creamos un objeto para parsear el documento
  AutoDetectParser parser = new AutoDetectParser();
  /*Utilizamos parser para que a traves del stream de datos, content, nos
      devuelva el contenido en el handler y los metadatos en medatada*/
  FileInputStream stream = new FileInputStream(f);
  try {
    parser.parse(stream, handler, metadata, parseContext);
  } finally{
      stream.close();}
  //Usamos tika.parse() para obtener metadatos mas completos.
  tika.parse(f, metadata);
  /*Creamos un objeto LanguageIdentifier al que le pasamos un string con
      el contenido del documento para extraer el idioma en el que este
      escrito.*/
  LanguageIdentifier object = new LanguageIdentifier(handler.toString());
  /*Extraemos el nombre del documento, la codificacion, el MIME tipo del
      archivo y el idioma en que esta escrito.*/
  String nam = metadata.get(Metadata.RESOURCE_NAME_KEY);
  String encod = metadata.get(Metadata.CONTENT_ENCODING);
  String type = metadata.get(Metadata.CONTENT_TYPE);
  String lang = object.getLanguage();
}
```

Para mostrar por pantalla la tabla, hemos creado un formato con un tabulado personalizado que nos proporciona los resutados de la Figura 1.

3. Opción -l.

Para esta opción, queremos extraer todos los enlaces para cada fichero del directorio pasado como parámetro. Para ello iteramos sobre cada fichero del directorio.

Declaramos un manejador de tipo *LinkContentHandler* que como hemos visto en los ejemplos, es necesario para obtener los links. [2] Tenemos que declarar los objetos para el contexto y parser, que junto con el objeto creado para los metadatos y el el *FileInputStream* se pasarán a la función *parse* como parámetros.

Finalmente guardaremos los links extraídos a partir del objeto *LinkContentHandler* en una lista y para cada archivo los mostraremos en el caso de que no sea vacía.

```
//Para cada archivo del directorio
for(String file : archivos){
  File f = new File("./" + args[1] + "/" + file);
  //Definimos un handler para links
  LinkContentHandler linkHandler = new LinkContentHandler();
  //Definimos un objeto para el contexto y el parser
  ParseContext parseContext = new ParseContext();
  AutoDetectParser parser = new AutoDetectParser();
  FileInputStream stream = new FileInputStream(f);
  try {
    parser.parse(stream, linkHandler, metadata, parseContext);
  } finally{
      stream.close();
  }
  //Guardamos los enlaces en una lista de Links
  List<Link> enlaces = linkHandler.getLinks();
  //Imprimimos aquellos enlaces que no sean vacios
  System.out.println("ENLACES del archivo " + file + ":\n");
  if(enlaces.isEmpty())
    System.out.println("NO se han encontrado enlaces.");
  else{
    for(Link l:enlaces)
      System.out.println(l);
}
```

Podemos ver su funcionamiento en la Figura 2.

NAME	ENCODING	TYPE	LANG.
memoriaRI.tex	UTF-8	application/x-tex; charset=UTF-8	es
p0_JSON_sesion_de_busqueda.pdf	UTF-8	application/pdf	gl
PR0-RI.docx	UTF-8	application/vnd.openxmlformats-officedocument.wordprocessingml.document	es
Proceso para busqueda de info.odt	UTF-8	application/pdf	es
presentacion_p0.odp	UTF-8	application/vnd.oasis.opendocument.presentation	es
P0_pres_ABHBZP.pptx	UTF-8	application/vnd.openxmlformats-officedocument.presentationml.presentation	gl
JAMM_1.pdf	UTF-8	application/pdf	gl
Presentacion_Daniel_Fernando.odp	UTF-8	application/vnd.oasis.opendocument.presentation	es
quijote.txt	UTF-8	text/plain; charset=UTF-8	es
Resumen de lo aprendido.pdf	UTF-8	application/pdf	es
Recuperación información PWP.pdf	UTF-8	application/pdf	gl
Resumen_Daniel_Fernando.odt	UTF-8	application/vnd.oasis.opendocument.text	es
Resumen_Daniel_Fernando.pdf	UTF-8	application/pdf	es

Figura 1: Tabla creada con la opción **-d** para los archivos contenidos en *docs*.

```
ENLACES del archivo presentacion_p0.odp:

NO se han encontrado enlaces.


ENLACES del archivo P0_pres_ABHBZP.pptx:

NO se han encontrado enlaces.


ENLACES del archivo JAMM_1.pdf:

<a href="https://www.humanlevel.com/diccionario-marketing-online/relevancia-contenidos">https://www.humanlevel.com/diccionario-marketing-onlin
e/relevancia-contenidos</a>
<a href="https://www.youtube.com/watch?v=cFCgFlqF5kw">https://www.youtube.com/watch?v=cFCgFlqF5kw</a>
<a href="https://www.youtube.com/watch?v=rfpv4eRtsO8">https://www.youtube.com/watch?v=rfpv4eRtsO8</a>
<a href="https://www.youtube.com/watch?v=39MCMp1EY8c">https://www.youtube.com/watch?v=39MCMp1EY8c</a>
<a href="https://es.wikipedia.org/wiki/JSON#:~:text=JSON%20(acr%C3%B3nimo%20de%20JavaScript%20Object,para%20el%20intercambio%20de%20datos.%C3%
87">https://es.wikipedia.org/wiki/JSON#:~:text=JSON%20(acr%C3%B3nimo%20de%20JavaScript%20Object,para%20el%20intercambio%20de%20datos.%C3%87</a
>
<a href="https://es.wikipedia.org/wiki/JSON#:~:text=JSON%20(acr%C3%B3nimo%20de%20JavaScript%20Object,para%20el%20intercambio%20de%20datos.%C3%
87">https://es.wikipedia.org/wiki/JSON#:~:text=JSON%20(acr%C3%B3nimo%20de%20JavaScript%20Object,para%20el%20intercambio%20de%20datos.%C3%87</a
>
<a href="https://es.ourcodeworld.com/articulos/leer/126/como-trabajar-con-json-facilmente-en-java">https://es.ourcodeworld.com/articulos/leer/
126/como-trabajar-con-json-facilmente-en-java</a>
<a href="https://es.ourcodeworld.com/articulos/leer/126/como-trabajar-con-json-facilmente-en-java">https://es.ourcodeworld.com/articulos/leer/
126/como-trabajar-con-json-facilmente-en-java</a>
<a href="https://www.40defiebre.com/guia-seo/que-es-seo-por-que-necesito">https://www.40defiebre.com/guia-seo/que-es-seo-por-que-necesito</a>
<a href="https://www.oracle.com/lad/technical-resources/articles/java/api-java-para-json-intro.html">https://www.oracle.com/lad/technical-reso
urces/articles/java/api-java-para-json-intro.html</a>
<a href="https://www.oracle.com/lad/technical-resources/articles/java/api-java-para-json-intro.html">https://www.oracle.com/lad/technical-reso
urces/articles/java/api-java-para-json-intro.html</a>
<a href="https://moz.com/beginners-guide-to-seo/how-search-engines-operate">https://moz.com/beginners-guide-to-seo/how-search-engines-operate<
/a>
<a href="https://www.40defiebre.com/guia-seo/que-es-seo-por-que-necesito">https://www.40defiebre.com/guia-seo/que-es-seo-por-que-necesito</a>
```

Figura 2: Extracto funcionamiento opción **-l** para 3 documentos.

4. Opción -t.

Para esta opción, contaremos las ocurrencias de cada palabra para cada archivo del directorio que recorramos. Además, generará un fichero CSV con el siguiente formato:

$$palabra; ocurrencias$$

Donde el número de ocurrencias será descendente.

Antes de empezar, crearemos (si no existe) una carpeta dentro del directorio *src* llamada *CSV*, donde generaremos los archivos CSV para cada fichero.

Definimos una función que introduce las entradas del Map en un stream, lo ordena con los valores del Map y crea otro Map con las entradas ordenadas y con los valores del antiguo.

```
public static Map<String, Integer> sortByValue(final Map<String, Integer>
    wordCounts) {
  return wordCounts.entrySet().stream().sorted((Map.Entry.<String,
      Integer>comparingByValue().reversed())).collect(Collectors.toMap(Map
      .Entry::getKey, Map.Entry::getValue, (e1, e2) -> e1, LinkedHashMap::
      new));
}
```

Para cada archivo del directorio, realizaremos los siguientes pasos:

- Parseamos el archivo con el método *parse* y pasamos el texto plano contenido en el *handler* a un string.
- Creamos un objeto stream a partir del string y mapeamos el elemento del stream a un array de strings con *map()*.
- Los elementos del array de strings son las palabras separadas a través de *split()*. Usaremos una expresión regular en el método *split()* para separar las palabras por espacios y signos de puntuación.
- Cada key tiene como valor 1 en el Map y las colisiones se resuelven sumando los valores. Así obtenemos un Map donde cada Key es la palabra que aparece y el valor su frecuencia.
- Ordenamos el Map con la función definida *sortByValue*.
- Creamos un archivo *.csv* para cada documento con las ocurrencias para cada palabra y lo guardamos en el directorio CSV.

```
for(String file : archivos){
  File f = new File("./" + args[1] + "/" + file);
  //Establece el l mite de caracteres en el constructor
  BodyContentHandler handler = new BodyContentHandler(1000000000);
  //Definimos un objeto para el contexto y el parser
  ParseContext parseContext = new ParseContext();
  AutoDetectParser parser = new AutoDetectParser();
  FileInputStream stream = new FileInputStream(f);
  try {
    parser.parse(stream, handler, metadata, parseContext);
  } finally{
      stream.close();}
  //Pasamos el contenido del documento del objeto handler a un string
  String str = handler.toString();
  //split() utiliza una expresion regular para separar las palabras.
  //flatMap() mapea el array a un stream que tiene como elementos los
      elementos del array.
  //collect() nos devuelve una lista de strings con los elementos del
      stream
  List <String> list = Stream.of(str).map(w -> w.split("[\\s
      \\;\\.\\,\\:\\?\\  \\  \\!\\(\\)]+")).flatMap(Arrays::stream).
      collect(Collectors.toList());
  //stream() crea una Stream cuyos elementos son los elementos de list.
  //collect() nos devuelve un Map que se crea a traves de Collectors.
      toMap().
  //Collectors.toMap() crea un Map donde los keys son los elementos del
      stream.
  Map <String, Integer > wordCounter = list.stream().collect(Collectors.
      toMap(w -> w.toLowerCase(), w -> 1, Integer::sum));
  //Ordenamos el Map segun los valores.
  wordCounter = sortByValue(wordCounter);
  Iterator it = wordCounter.keySet().iterator();
  //Creamos archivos .csv con el recuento de las palabras de cada
      documento en ./CSV
  PrintWriter writer = new PrintWriter("./CSV/" + file + ".csv");
  writer.print("Text" + ";" + "Size\n");
  while(it.hasNext()){
    String key = (String) it.next();
    writer.print(key + ";" + wordCounter.get(key) + "\n");
  }
  writer.close();
}
```

Para el documento *quijote.txt* obtendremos un archivo *.csv* que contiene todas las palabras del libro de Cervantes junto con su número de ocurrencias.

Como vimos en clase, las palabras más frecuentes son: *que, de, y, la* como se puede apreciar en el fragmento del archivo obtenido.

```
Text;Size
que;20413
y;17962
de;17947
la;10224
...
quijote;2158
sancho;2140
es;2118
```

Haciendo una representación gráfica en escala logarítmica del ranking de palabras frente al número de ocurrencias, podemos ver cómo el Quijote, cumple la **ley de Zipf**: la posición en el ranking de una palabra (R) por su frecuencia (F) es aprox. constante (k). [4]

$$F = \frac{k}{R}$$

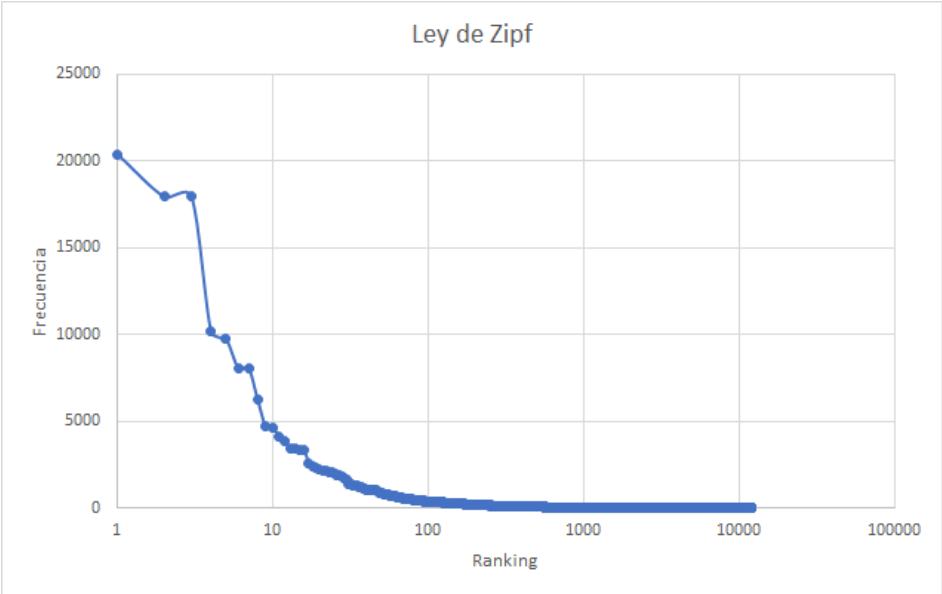

Figura 3: Gráfica en escala logarítmica para el archivo *quijote.txt.csv*

.

5. Nube de palabras.

Podemos generar nubes de palabras a partir de los ficheros CSV obtenidos con la opción **-t** desde la web de *wordart*. [3]

Hemos elegido dos de entre todos los ficheros CSV generados y a continuación mostraremos el resultado. Además, hemos configurado una forma y color diferente para cada nube generada.

Podemos ver las nubes de palabras resultantes en la Figura 4.

6. Método de compilación.

Ejecutamos las siguientes órdenes a través de terminal situados en el directorio *src*.

```
> javac -cp ./tika-app-1.23.jar P1Tika.java
> java -cp ./tika-app-1.23.jar:. P1Tika -option docs
```

Donde -*option* puede ser una de las opciones de la siguiente lista $[-d, -l, -t]$.

7. Trabajo en Grupo.

El trabajo lo hemos repartido, en primera instancia, de la siguiente manera:

- **Daniel Bolaños Martínez**: Opción -d (formato de tabla), Opción -l y elaboración de la memoria.
- **Fernando de la Hoz Moreno**: Opción -d y Opción -t.

No obstante, hemos mantenido el contacto durante el desarrollo de la práctica y hemos colaborado conjuntamente en la elaboración del proyecto en su totalidad.

Figura 4: Nubes para *Resumen_Daniel_Fernando.pdf.csv* y *quijote.txt.csv*

Referencias

[1] Guión de la práctica 1 de la asignatura.

[2] `https://tika.apache.org/1.24.1/examples.html`

[3] `https://wordart.com/create`

[4] `https://en.wikipedia.org/wiki/Zipf%27s_law`