Practica 1. Preprocesado de documentos

Cómo lo hemos hecho

Recorrer directorio

Para recorrer el directorio recuperamos el segundo parámetro de la terminal (El nombre del archivo), y creamos un File con ese nombre, con `listFiles()` obtenemos los archivos que cuelgan del directorio (ignorando subdirectorios), de esta forma tenemos el listado de archivos que analizaremos.

Flags de los ejercicios

El parámetro de terminal 0 será donde ponemos el flag (-d, -l, -t). Se llama a la función determinada dependiendo del flag que se haya puesto en la linea de comandos.

Dentro de cada función, se recorrerá la lista de archivos que teníamos, y por cada archivo utilizaremos la clase auxiliar DocumentAnalyzer, el cual nos permitirá a los datos necesarios encapsulando el analisis del documento del main, finalmente usaremos la clase auxiliar OutputHelper para mostrar en pantalla o guardar en csv los datos que nos aporta DocumentAnalyzer.

DocumentAnalyzer

Es la clase encargada de analizar el documento y obtener su nombre, metadatos, contenido, enlaces y la lista de palabras.

Constructor

En el constructor generamos todos los datos que podemos extraer directamente con Tika

- Nombre
- Contenido
- Enlaces
- Lenguaje
- Metadatos

Para ello usamos directamente el `AutoDetectParser`

Métodos

Los métodos se encargan de extraer información más específica y en un formato concreto:

- contador()

  Este método devuelve una lista de palabras junto al número de veces que ha aparecido en el documento

Para ello toma el contenido del documento y con split, separa las palabras por espacios, ahora por cada palabra, comprobamos que no contiene carácteres extraños con la expresión regular `[a-zA-Z\\u00C0-\\u024F\\u1E00-\\u1EFF]+`, que basicamente comprueba que este formado únicamente por carácteres y letras con tilde.

Para poder ir contando si aparece o no, vamos metiendo las palabras en un **HashMap**, de tal forma de que podemos comprobar rápidamente si la palabra ya ha aparecido, y cuantas veces lleva, y poder ir introduciendo palabras e incrementar el contador de las que ya estaban.

Después lo pasamos a un `ArrayList` para poder ordenarlo, para ello hemos usado una función de comparación para luego usar en el método de ordenación `Collections.sort()`, que comparará por el número de ocurriencias de cada palabra.

Para pasar de `HashMap` a `ArrayList`, necesitamos de forma intermedia un Set.

OutputHelper

Esta clase tiene funciones para asistir a la hora de mostrar en pantalla o exportar los datos en ".csv".

Método csvWriter

Toma la lista con palabra-valor, y lo va introduciendo linea a linea con el formato adecuado de ".csv" (separado en este caso por ';').

Método csvWriterMetadata

Este método recibe como parametros una lista con los nombres de los archivos, una lista con los lenguajes de los diferentes archivos del directorio, una lista de objetos `Metadata` con los metadatos de los archivos y un objeto `String` que define el nombre del archivo generado como salida.

La finalidad de este método es generar un archivo ".csv" con los metadatos de los archivos que se especifican en el guión de la práctica.

Para conseguirlo se hace uso de `FileWriter.append()` para primero añadir los titulos de la tabla y dentro de un bucle for añadir los diferentes valores de los metadatos accediento a los diferentes parámetros del método.

Método pcsvWriterLinks

Este método recibe una lista con los enlaces que han sido recogidos de los diferentes archivos con ayuda de la clase `DocumentAnalyzer` y un objeto `String` que define el nombre del archivo generado como salida.

Este método al igual que el anterior, hace uso de `FileWriter.append()` para volcar el contenido de la lista de enlaces al archivo de salida.

¿Qué ha hecho cada miembro?

Adrián

- Apartado (-d).

- Apartado (-l) -> Exportado a CSV
- Constructor de DocumentAnalyzer.java
- Método csvWriterMetadata de OutputHelper.java
- Memoria

Andrés

- Clase DocumentAnalyser.java
- Métodos csvWriter y print de OutputHelper.java
- Apartado (-l)
- Apartado (-t)
- Memoria