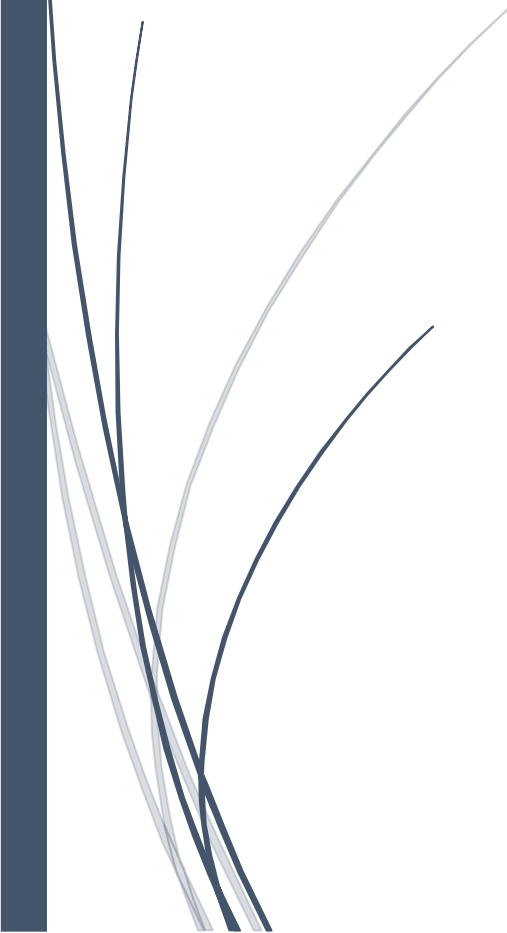
A thick dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the text '2015/2016'.

2015/2016

Práctica 2.3

Consultas simples

Several thin, dark blue wavy lines originate from the bottom left and curve upwards and to the right.

Francisco Javier Moya Viedma
Jose Antonio Medina García
Nasrdine El Houfi
Francisco Javier Díaz Herrera

RECUPERACIÓN DE INFORMACIÓN

- **Trabajo en Grupo: Detalles de la colaboración en el grupo**

El trabajo a realizar en la práctica se ha distribuido equitativamente entre los integrantes del grupo, de manera que todos hemos tratado de implementar las tareas a llevar a cabo para la construcción del índice, tanto en la parte de implementación como para la documentación del mismo.

Siguiendo la metodología SCRUM, hemos ido organizándonos semanalmente.

Para llevar a cabo dicha organización, se ha usado la plataforma `github` en la que cada uno hemos ido incorporando las tareas que nos habíamos propuesto de una forma paulatina, seguido de reuniones semanales para discutir las tareas desarrolladas, probarlas y realizar modificaciones, aclaraciones, etc., cuando se estimen oportunas.

Seguido este proceso, se ha podido obtener un proyecto estructurado y limpio.

- **Arquitectura del sistema**

La mayor parte de la implementación y las posteriores ejecuciones de la misma se han realizado en un equipo con características del tipo:

```
Ubuntu 14.04
Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
64 Bits
8 GB de Memoria RAM
```

- **Estructuras de datos empleadas**

Básicamente, se han usado las estructuras de datos indicadas en el guion de la práctica, esto es:

- **Vocabulario:** Tabla Hash que asocie un término con su ID, `tID`
- **Documentos:** Tabla Hash que asocie un documento con su ID, `dID`
- **IDF:** Tabla Hash que asocie un `tID` con su `idf`
- **NormaDoc:** Tabla Hash que asocie un `dID` con su factor de normalización (el $\max_{t \in D} tf(D)$).
- **Indice:** Tabla Hash que asocie un `tID` con el conjunto de documentos (estos aparecen de forma ordenada), en los que aparece, así como el `tf` del término en el mismo, esto es pares `< dID; tf >`
- **ListaTerminos:** ArrayList de String que contiene los términos de búsqueda por los que podemos buscar y tokenizado
- **Consulta_final:** ArrayList de String con los nombres de los documentos recuperados mas relevantes.

Además de esto, para almacenar las palabras vacías se ha usado un `Map<String, Object>`, para tener un rápido acceso a las propias palabras vacías; un `HashMap<String, Integer>` para almacenar cada token con su respectiva frecuencia.

Se ha creado también la clase abstracta `Pair<S, T>`, para que así pudiéramos usar los pares `<S, T>` en la implementación.

- **Instrucciones de uso (en líneas de comandos)**

1. Descomprima el archivo `.zip` que contiene el código de la práctica.
2. Ejecute en línea de comandos, desde la ubicación del archivo, lo siguiente (`README.txt`):

```
?> java -jar create_index.jar
```
3. Cuando se ejecute el programa, deberá introducir la ruta completa en donde se encuentra la colección.
4. La salida del programa proporciona lo siguiente:
 - Irá mostrando mensajes mientras va creando las estructuras oportunas para la construcción del índice.
 - Con las opciones 1 al 4 podremos ver las estructuras.
 - Con la opción 5, generará un archivo, denominado `index.txt`, que se creará en el directorio desde donde se ejecuta la aplicación, en donde se recogerá todo el contenido del índice para la colección especificada.
 - Podremos realizar una consulta en la opción 6, nos indicará la consulta a realizar y tras meter por pantalla los términos, nos devuelve un ranking ordenado con los documentos encontrados en caso de que encuentre algo o un mensaje en caso de no encontrar documentos.

- **Código desarrollado (en un fichero zip)**

Dentro de `codigos.zip` en el archivo `create_index.jar` se encuentra el código fuente, en la carpeta `/lib` se encuentran las librerías de `libstemer`, pero la librería `tika-app-1.6` se tiene que meter porque excedía el tamaño permitido por correo, usadas, en la carpeta `/clean` se encuentran los ficheros que contienen las palabras vacías. Además, se incluye un fichero `/collection` en donde se encuentran los archivos con la implementación (código) del proyecto.

- **Limitaciones**

Las colecciones tienen que estar en español, de lo contrario no se garantiza la correcta funcionalidad del índice.

- **Tiempos en realizar consultas**

Tras hacer la consulta nos muestra un mensaje con el tiempo que ha tardado nuestra consulta en realizarse y una media calculada con otras consultas estándar para comparar los tiempos.

- **JavaDoc**

Dentro de `Practica2.zip`, se encuentra una carpeta denominada `javadoc` en donde se encuentra una documentación del código, donde se describe detalladamente toda la estructura del sistema.

Para obtener dicha documentación, basta con ejecutar el archivo `index.html` (se abre en un navegador Web).

- **Análisis de eficiencia**

Lo que se ha hecho, para este apartado, es obtener el tiempo en ejecución de la construcción del índice, para las distintas colecciones (`quijote`, `ParlamentoPDF`, `ObrasCervantes` y `ParlamentoXML`), y se ha realizado una gráfica para poder ver de manera más visual las diferencias de tiempos en función del tamaño (número de tokens del vocabulario) que tiene cada colección.

Los tiempos obtenidos, para cada colección, son los siguientes:

	Tamaño (vocabulario)	Tiempo (segundos)
ParlamentoPDF	17497	37
ObrasCervantes	19472	8
ParlamentoXML	14602	11
quijote	9867	1

Por último, se ha realizado una gráfica que muestra de manera más visual la tabla anterior:

