FAQ – v.2
Ejercicio 4 – Preguntas frecuentes

josé a. mañas

18.3.2016

1 ¿Qué hay que hacer?

Hay que leer un fichero de texto, extrayendo las palabras que aparecen.

Estas palabras se van metiendo en un diccionario que llamaremos tabla 1, de forma que cuando una palabra no está en el diccionario, se mete como nueva, contando que ha aparecido 1 vez. Si al ir a meterla ya está, se incrementa el contador. Al final de esta fase tenemos un diccionario con las palabras que son diferentes y una cuenta de cuántas veces aparece cada palabra.

Usaremos la estructura Registro para almacenar esta información. Ver Anexo.

A continuación, la tabla se ordena. Para ello, descargamos los datos de la tabla a un array

```
Registro[] datos = new Registro[tabla.size()];
```

Ordenamos los datos.

Podemos responder a preguntas sobre estos datos ordenados.

Tabla 1: usaremos un Diccionario para la primera parte del ejercicio, donde tenemos que localizar rápidamente el contador asociado a cada palabra e irlo incrementando.

Tabla 2: usaremos un array ordenado para la segunda parte del ejercicio, donde tenemos que consultar rápidamente en función del número de apariciones de cada palabra.

El paso de la tabla 1 a la tabla 2 es costoso y solo lo haremos cuando haga falta: calculo perezoso (lazy evaluation).

1.1 WordCounter.java

Se facilita el método para cargar un fichero de texto. Cada vez que se carga un fichero de texto se vacía el diccionario y se invalida la tabla 2.

```
package es.upm.dit.adsw.ej4;

import java.io.File;
import java.io.IOException;
import java.util.*;

/**
 * Analizador de textos.
 *
 * @version 11.3.2016
 */
public class WordCounter {

    /**
```

```
     * Constructor.
     */
    public WordCounter() { … }

    /**
     * Carga un fichero de texto.
     *
     * @param file fichero.
     * @throws IOException si hay problemas con el fichero.
     */
    public void load(File file)
            throws IOException {
        dict.clear();
        Scanner scanner = new Scanner(file, "UTF-8");
        scanner.useDelimiter("[^\\p{javaLowerCase}\\p{javaUpperCase}]+");
        while (scanner.hasNext()) {
            String word = scanner.next().toLowerCase();
            mete WORD en el el diccionario
        }
        scanner.close();
    }

    /**
     * Tamano de la tabla de contadores y del registro de palabras.
     *
     * @return numero de palabras.
     */
    public int size() { … }

    /**
     * Devuelve las n palabras mas usadas (si n es positivo).
     * Devuelve las n palabras menos utilizadas (si n es negativo).
     *
     * @param n numero de palabras contando desde el principio
     *          o desde el final.
     * @return lista de n registros.
     */
    public List<Registro> getTop(int n) { … }

    /**
     * Devuelve cuantas palabras hay por debajo de un umbral c.
     *
     * @param c umbral de cuenta.
     * @return numero de palabras que aparcecen en el texto menos de c veces.
     */
    public int countBelow(int c) { … }

}
```

2 Tabla 1 – Opciones

Vale cualquier diccionario que asocie una palabra a un registro y sea rápido.

Medidas de los profesores procesando el quijote:

diccionario	tiempo de carga	comentario
HastListas(1000)	650ms	hasta 1.000 datos es O(1) y si luego seguimos con listas, será O(n); pero dividiendo el tiempo de ejecución por 1.000
BST	700ms	si el árbol nos sale equilibrado, será O(log n); si no, puede empeorar hasta O(n)
java.util.HashMap	650-700ms	java ajusta dinámicamente el tamaño de la lista para que la búsqueda sea siempre O(1)
java.util.TreeMap	700-800ms	java reequilibra el árbol para que la profundidad no pase de 2 log(n), de forma que las operaciones se mantienen O(log n).

Como se ve, los tiempos son muy similares. Elija la que quiera, o varias para experimentar.

2.1 Tabla hash propia

Puede adaptar el código del ejercicio 3.

Cuando detecte un duplicado (una clave que ya está) hay que incrementar la cuenta.

2.2 BST – Árbol binario de búsqueda

Puede adaptar el código del ejercicio 2. Sólo necesita la operación put().

Cuando detecte un duplicado (una clave que ya está) hay que incrementar la cuenta.

2.3 Tabla hash de la biblioteca de java

Use

```
Map<String, Registro> tabla1 = new HashMap<String, Registro>();
```

Para cargar la tabla lo primero que debe hacer es ver si una cierta clave ya está. Si está, se incrementa el registro. Si no está, se crea un registro nuevo y se mete en la tabla.

2.4 Árbol binario de la biblioteca de java

Use

```
Map<String, Registro> tabla1 = new TreeMap<String, Registro>();
```

Para cargar la tabla lo primero que debe hacer es ver si una cierta clave ya está. Si está, se incrementa el registro. Si no está, se crea un registro nuevo y se mete en la tabla.

3 De tabla 1 a tabla 2

Solo hará cuando sea necesario (lazy evaluation). Para ello, cuando cargue un nuevo fichero de datos, marque la tabla 2 como inválida.

Cuando sea necesario usar la tabla 2 y esté marcada como inválida, debe

1. descargar los datos de la tabla 1 a un array
2. ordenar el array
3. marcar la tabla como válida

Si va a consultar la tabla 2 y está marcada como válida, puede usarla directamente.

4 Descarga de la tabla 1 a un array

El array que necesitamos es

```
Registro[] datos = new Registro[tabla1.size()];
```

Dependiendo de la estructura de la tabla 1, haremos la descarga de una forma u otra.

4.1 Tabla hash propia

Necesita un bucle por todas las ranuras (slots). En cada ranura va copiando todos los datos de la lista al array de datos.

4.2 BST – Árbol binario de búsqueda

Necesita recorrer el árbol en cualquier orden. Por ejemplo, en preorden.

En cada nodo, se copia el registro al array de datos y seguimos recorriendo:

```
private int descarga(Registro[] data, int at, Nodo node) {
    if (node == null)
        return at;
    data[at] = node.getRegistro();
    int nd = descarga(data, at + 1, node.getIzq());
    int nx = descarga(data, nd, node.getDer());
    return nx;
}
```

4.3 Tabla hash de la biblioteca de java

Hay que iterar sobre los valores de la tabla e ir copiándolos al array de datos. Algo así:

```
private Registro[] descarga(Map<String, Registro> dict) {
    Registro[] datos = new Registro[dict.size()];
    int at = 0;
    for (Registro registro : dict.values())
        datos[at++] = registro;
    return datos;
}
```

o incluso

```
private Registro[] descarga(Map<String, Registro> dict) {
    Registro[] datos = new Registro[dict.size()];
    Collection<Registro> values = dict.values();
    values.toArray(datos);
    return datos;
}
```

4.4 Árbol binario de la biblioteca de java

Igual que con el HashMap.

5 Tabla 2 – Opciones

5.1 Algoritmo de ordenación propio

Puede usar cualquiera de los algoritmos vistos en clase dentro de los razonables. Razonable quiere decir que el caso peor sea O(n log n).

Para comparar 2 registros, use el contador.

5.2 Algoritmo de ordenación de la biblioteca de java

Puede usar la biblioteca de java

```
Arrays.sort(datos)
```

si bien para ello hay que contarle a java qué relación de orden hay que usar para comparar registros. Esto requiere modificar Registro para que implementa la interfaz Comparable

```
public class Registro implements Comparable<Registro> {
```

y esto requiere implementar un nuevo método

```
@Override
/**
 * Compares this object with the specified object for order.
 * Returns a negative integer, zero, or a positive integer
 * as this object is less than, equal to, or greater than
 *  the specified object.
 *
 * @param   o the object to be compared.
 * @return a negative integer, zero, or a positive integer as this object
 *          is less than, equal to, or greater than the specified object.
 */
public int compareTo(Registro o) {
    return cnt - o.cnt;
}
```

6 Pruebas

Haga pruebas de humo (*smoke test*) con un fichero de texto pequeño y controlado donde pueda calcular a mano los resultados esperados.

Luego haga pruebas con casos límite: fichero vacío, con menos de 10 palabras, con 10 palabras iguales, etc.

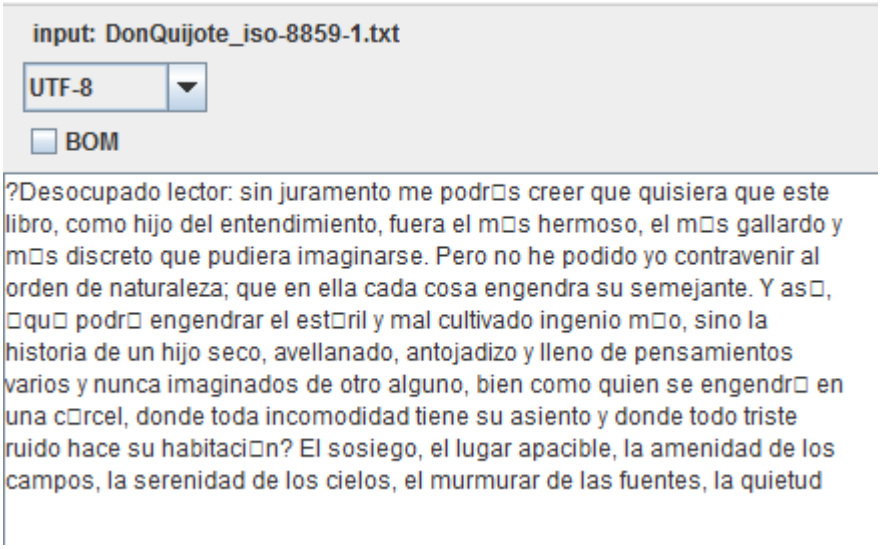
Por último, pruebe que el tiempo de ejecución es aceptable con un fichero de gran tamaño.

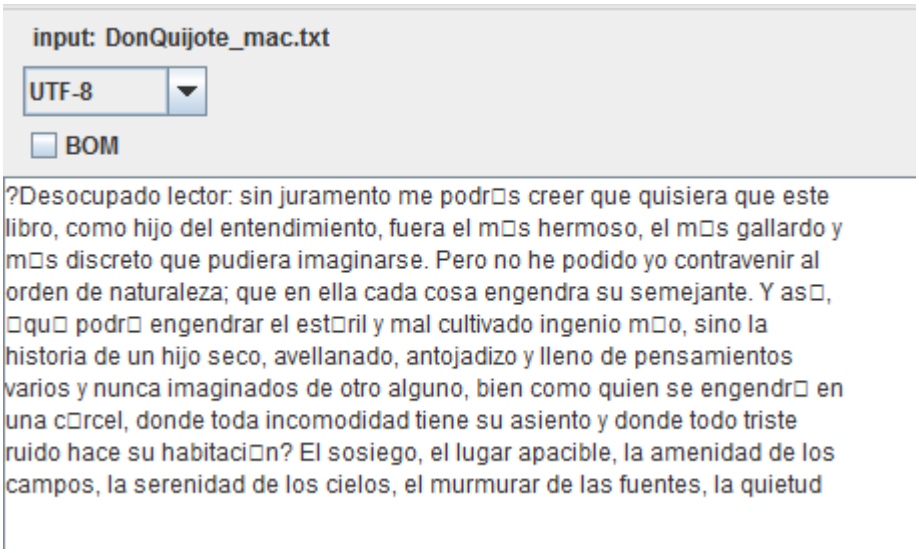
7 Sobre la codificación de los caracteres

El código que les facilitamos está preparado para leer caracteres Unicode codificados en formato UTF-8. Es posible que el sistema operativo en el que trabaja esté utilizando otro formato. Por ejemplo

Windows	ISO-8859-1, windows-1252, UTF-16BE, UTF-16LE, …
Mac OSX	MacRoman, UTF-16BE, UTF-16LE, …

Esto se aprecia en que aparecen caracteres raros





Para solventarlo, cambie la codificación en el método load()

```
public void load(File file)
        throws IOException {
    dict.clear();
    Scanner scanner = new Scanner(file, "UTF-8");
```

8 Entrega

- WordCounter.java
- Registro.java

- otros métodos que haya usado

9 Anexo – Registro.java

```java
package es.upm.dit.adsw.ej4;

/**
 * Contenedor para asociar un contador a una palabra.
 *
 * @author jose a. manas
 * @version 8.3.2016
 */
public class Registro {
    private final String clave;
    private int cnt;

    /**
     * Constructor.
     * El contador de inicaliza a 1.
     *
     * @param clave palabra.
     */
    Registro(String clave) {
        this.clave = clave;
        this.cnt = 1;
    }

    /**
     * Getter.
     *
     * @return palabra.
     */
    public String getClave() {
        return clave;
    }

    /**
     * Getter.
     *
     * @return contador.
     */
    public int getCnt() {
        return cnt;
    }

    /**
     * Incrementa el contador.
     */
    public void inc() {
        this.cnt++;
    }
}
```