

Taller 3 y 4 - Contador de palabras e índice

Amir Rodriguez Mejia 506222032
Fundación Universitaria Konrad Lorenz

I. INTRODUCCIÓN

En este documento, vamos a dar a conocer el uso de dos códigos totalmente independientes. En el primer código, contaremos el número de repeticiones de un documento o texto, mientras que en el segundo código, dividiremos las líneas del texto en secciones utilizando un hashmap. Esto nos permitirá determinar en qué índice se encuentra cada palabra. Para ello, tendremos en cuenta el tiempo en que demora la ejecución del código visto como milisegundos y el espacio visto en megabytes.

II. REQUISITOS DEL PROYECTO

Para llevar a cabo este taller, se hizo uso de una cadena de texto, que se encuentra organizada línea por línea y estas se encuentra agrupada por comillas "", la cual se presenta de la siguiente manera `my_documents = [texto]`, el ejercicio consiste en la creación de dos algoritmos, por un lado, uno nos debe indicar el número de veces que se encuentra una palabra en el texto, claramente, hicimos una excepción de los conectores como: "La, el, los, a, como, entre otros", por el otro lado, el segundo código consiste en la lectura del texto y que el algoritmo nos indique en que índice se encuentra ubicada cada palabra, para ello se hace uso de un `arraylist` y `hashmap`, que nos permite almacenar cada línea o bloque en un espacio de memoria.

III. CÓDIGOS USADOS EN LA PRÁCTICA

III-A. Código en Java contador de palabras

Figura 1. Código en Java contador de palabras-primeras partes

Figura 2. Código en Java contador de palabras-segunda parte

A continuación vamos a indicar, que proceso maneja cada línea o función del código:

1-Definimos las palabras más comunes o que deseamos retirar con la siguiente palabra (*stopwords*): Se declara un arreglo tipo arraylist, de las palabras que no deseamos tener en cuenta o que sean comunes, luego se crea el arreglo como stopWords.

```
String[] myDocuments = ...;  
List<String>stopWords = Arrays.asList("el", "la", "los", ...);
```

2-Se crea un mapa para contar la frecuencia de cada palabra en los documentos.

```
Map<String, Integer>wordCount = new HashMap<>();
```

3-Procesa cada documento:

- Divide el documento en palabras.
 - Convierte cada palabra a minúsculas.
 - Verifica si la palabra es una palabra común (stopword).
Si no lo es, la cuenta.

```
for (String document : myDocuments) {  
    String[] words = document.split("\\s+");  
    for (String word : words) {  
        word = word.toLowerCase();  
        if (!stopWords.contains(word)) {  
            wordCount.put(word, wordCount  
                .getOrDefault(word, 0) + 1);  
        }  
    }  
}
```

Organizamos las palabras por frecuencia en orden descendente en la lista sortedWordCount.

```
List<Map.Entry<String, Integer> sortedWordCount = new
ArrayList<>(wordCount.entrySet());
sortedWordCount.sort((entry1, entry2) -> entry2.getValue().compareTo(entry1.getValue()));
```

Finalmente imprimimos las palabras con mayor frecuencia.

```
for (Map.Entry<String, Integer> entry : sortedWordCount)
System.out.println(entry.getKey() + ": " + entry.getValue());
```

III-B. Código en Java contador con índice

```
import java.util.*;
public class Contadorindice {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new FileReader("text.txt"));
        ArrayList<String> my_documents = new ArrayList<>();
        while ((line = reader.readLine()) != null) {
            my_documents.add(line);
        }
        reader.close();
        List<String> indice = new ArrayList<>();
        for (String documento : my_documents) {
            String[] palabras = documento.toLowerCase().split("\\s+");
            for (String palabra : palabras) {
                if (!palabrasOmitidas.contains(palabra)) {
                    indice.computeIfAbsent(palabra, k -> new ArrayList<>()).add(i + 1);
                }
            }
        }
        for (String palabra : indice.keySet()) {
            System.out.println(palabra + ": " + indice.get(palabra));
        }
    }
}
```

Figura 3. Código Python primera parte

```
import java.io.*;
import java.util.*;
public class Contadorindice {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new FileReader("text.txt"));
        ArrayList<String> my_documents = new ArrayList<>();
        while ((line = reader.readLine()) != null) {
            my_documents.add(line);
        }
        reader.close();
        List<String> indice = new ArrayList<>();
        for (String documento : my_documents) {
            String[] palabras = documento.toLowerCase().split("\\s+");
            for (String palabra : palabras) {
                if (!palabrasOmitidas.contains(palabra)) {
                    indice.computeIfAbsent(palabra, k -> new ArrayList<>()).add(i + 1);
                }
            }
        }
        for (String palabra : indice.keySet()) {
            System.out.println(palabra + ": " + indice.get(palabra));
        }
    }
}
```

Figura 4. Código Python primera parte

1- Creamos un arraylist con las palabras que deseamos omitir.

```
Set<String> palabrasOmitidas = new HashSet<>(Arrays.asList( ".el", "la", "los", "las", "ün", "üna", ...));
```

2- Creamos una instancia de memoria y con ayuda de un hashmap, almacenamos las palabras en cada bloque o espacio.

```
Map<String, List<Integer> indice = new HashMap<>();
```

3- Creamos un arreglo tipo arraylist donde almacenamos las líneas de texto

```
List<String> my_documents = Arrays.asList();
```

4- Creamos un ciclo for para realizar las siguientes funciones:

- La primera línea inicia un bucle que itera a través de los documentos.
- La segunda línea obtiene un documento específico de la lista de documentos.
- La tercera línea divide el contenido del documento en palabras, convierte todas las letras a minúsculas y divide en palabras en función de espacios en blanco.
- Las líneas dentro del segundo bucle for realizan operaciones en cada palabra del documento, como eliminar caracteres no alfabeticos y verificar si la palabra está en la lista de palabras omitidas.
- Si la palabra no está en la lista de palabras omitidas, la última línea agrega la ubicación de la palabra al índice.

```
for (int i = 0; i < my_documents.size(); i++) {
    String documento = my_documents.get(i);
    String[] palabras = documento.toLowerCase().split("\\s+");
    for (String palabra : palabras) {
        palabra = palabra.replaceAll("[^a-zA-Z]", "");
        if (!palabrasOmitidas.contains(palabra)) {
            indice.computeIfAbsent(palabra, k -> new ArrayList<>()).add(i + 1);
        }
    }
}
```

5- por último se imprime el contenido

```
for (Map.Entry<String, List<Integer> entry : indice.entrySet())
System.out.println(entry.getKey() + ": " + entry.getValue()); //O(1)
```

```
System.out.println(entry.getKey() + ": " + entry.getValue()); //O(1)
```

IV. EJECUCIÓN DE CÓDIGO

- Sistema Operativo: Windows 11
- Procesador: Core I7 11va Gen
- RAM: 16GB
- Disco duro SSD: 473 GB

Intentos en Java contador de palabras:

Primer intento Cpu Time Java:

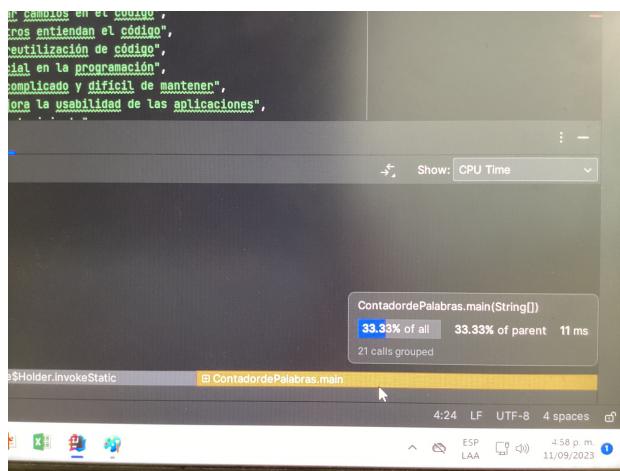


Figura 5. Primer intento Cpu Time Java

Primer intento Memory Allocations Java:

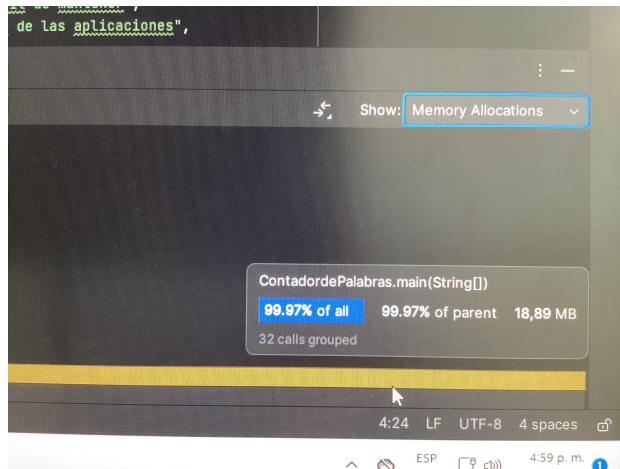


Figura 6. Primer intento Memory Allocations Java

Primer intento Total Time Java:

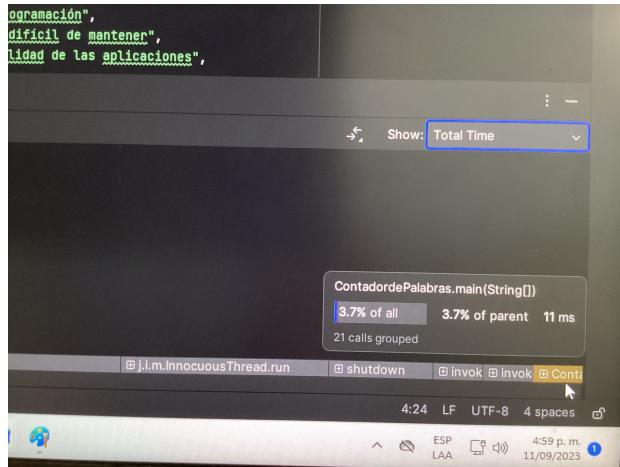


Figura 7. Primer intento Total Time Java

Intentos en Java contador de indice:

Primer intento Cpu Time Java:

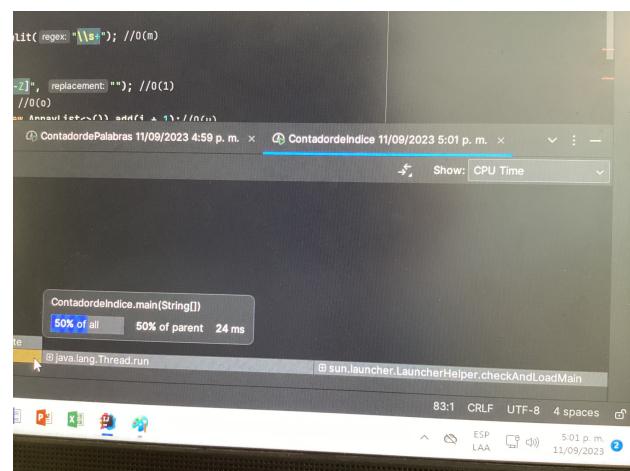


Figura 8. Primer intento Cpu Time Java

Primer intento Memory Allocations Java:

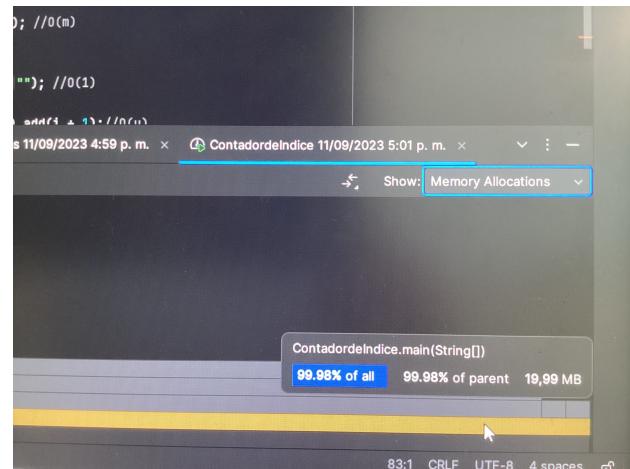


Figura 9. Primer intento Memory Allocations Java

Primer intento Total Time Java:

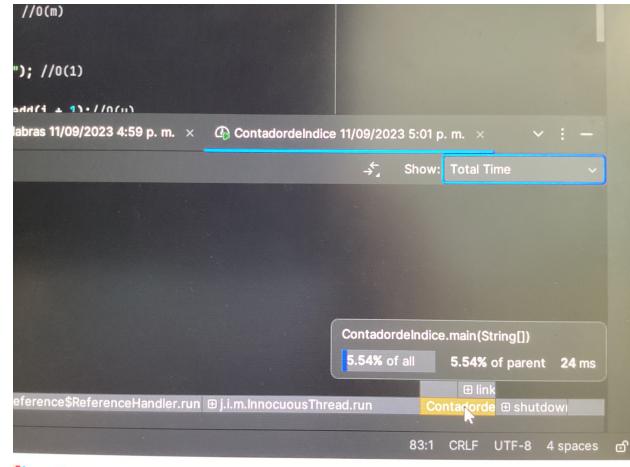


Figura 10. Primer intento Total Time Java

V. CONCLUSIONES

Podemos concluir que, gracias al uso de un HashMap y un ArrayList, que son estructuras de datos que nos permiten almacenar información de manera eficiente, podemos obtener las repeticiones de palabras y su ubicación en los documentos.

Este código demuestra cómo aprovechar estas estructuras de datos para procesar y analizar texto de manera efectiva. La combinación de ciclos y funciones nos brinda la capacidad de realizar tareas avanzadas, como la construcción de un índice de palabras. Con estas bases, estamos bien posicionados para explorar aplicaciones más complejas en el futuro, como la creación de un motor de búsqueda.

En un futuro, podríamos utilizar estas mismas funciones para desarrollar un buscador que nos permita realizar búsquedas específicas de palabras.