Diseño e implementación para la gestión de páginas web

Iker Fernández, Urko Horas y Eneko Rodríguez

October 7, 2024

Índice

1 Introducción **2**

2 Diseño de las clases **3**
2.1 Clase Web 4
2.2 Clase ListaWebs 4
2.3 Clase ConjuntoPalabras 5

3 Descripción de las estructuras de datos principales **6**
3.1 TreeSet: 6
3.2 ArrayList 7
3.3 Web[] (Array) 7
3.4 HashSet 7

4 Diseño e implementación de los métodos principales **9**
4.1 Método cargarLista() 9
4.2 Método cargarSalientes() 10
4.3 Método cargarPalabraClave() 11
4.4 Método cargarListaEnFicheros() 11
4.5 Método buscarWeb(String pDir) 12
4.6 Método borrarWeb(String pDir) 13
4.7 Método añadirWeb(Web pWeb) 13
4.8 Método ordenarWebs() 14
4.8.1 Método quickSort(ArrayList<*String*>pLista, int pBajo, int pAlto) 14
4.8.2 Método particion(ArrayList<*String*>pLista pLista,int pBajo, int pAlto) 14
4.9 Método salientes(String pWeb) 15
4.10 Método word2Webs(String pS) 16
4.11 Método web2Words(String pWeb) 16
4.12 Método ident2String(int pX) 17
4.13 Método imprimir() 17

5 Algoritmo **18**
5.1 Código 18
5.1.1 Clase ConjuntoPalabras 18
5.1.2 Clase Web 19
5.1.3 Clase ListaWebs 20
5.2 JUnits 24

6 Conclusiones **26**

Apartado 1

Introducción

En esta primera práctica de Estructura de Datos y Algoritmos, se nos presenta un problema de gestión de un número muy grande de páginas web, que puede abarcar millones de URLs. Existe una interconexión entre webs, donde cada web tiene enlaces a otras. Además, cada web cuenta con un conjunto de palabras clave, que a su vez aparecen en diferentes webs.

Algunos de los subprogramas que se nos piden implementar son la búsqueda de una web específica, obtener la lista de palabras clave dada una web y viceversa. Asimismo, se requiere la capacidad de añadir nuevas páginas y enlaces, así como de eliminar webs del sistema cuando sea necesario.

Apartado 2

Diseño de las clases

Como solución al problema para el que se nos ha propuesto buscar un algoritmo, hemos planteado las siguientes clases, que serán explicadas posteriormente.

Web, ListaWebs, ConjuntoPalabras y Main, que únicamente se encarga de ejecutar el programa.

Los enlaces, programas y atributos quedan resumidos en diagrama de clases que se muestra en la figura 2.1.

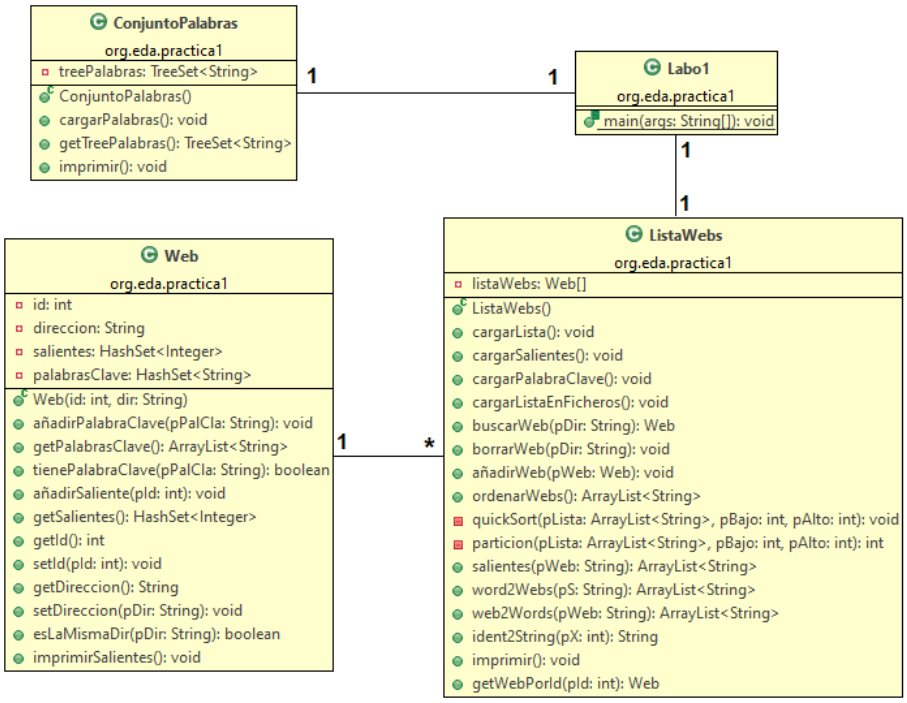

Figure 2.1: Diagrama de clases.

2.1 Clase Web

La clase Web cuenta con cuatro atributos: int id, String direccion, HashSet de Integer salientes y HashSet de String palabrasClave. Las webs almacenan tanto su id, como dirección, conjunto de palabras clave que contienen y el conjunto de webs salientes. Los métodos implementados en esta clase son:

- añadirPalabraClave(String pPalabra): Se añade la palabra clave pPalabra al conjunto de palabras clave de la Web.
- obtenerPalabrasClave(): Devuelve un ArrayList con las palabras clave de la Web.
- tienePalabraClave(): Devuelve True si la Web tiene la palabra clave dada en su conjunto de palabras clave, si no False.
- añadirSaliente(int pId): Añade un saliente al conjunto de webs salientes.
- getSalientes(): Devuelve el conjunto de webs salientes.
- getId(): Devuelve el id de la Web.
- setId(int pId): Se le da un valor al id de la Web.
- getDir(): Devuelve la dirección de la Web.
- setDir(String pDir): Se le da un valor a la dirección de la Web.
- esMismaDir(String pDir): Devuelve True si la dirección de la Web es la misma que la dada, si no False.
- imprimirSalientes(): Imprime los salientes de la Web.

2.2 Clase ListaWebs

La clase ListaWebs cuenta con un único atributo de estructura array de Webs. La función de esta clase es almacenar las distintas webs que se encuentran en el sistema. Los métodos implementados en esta clase son:

- cargarLista(): Carga en el array ListaWebs cada Web existente. Utilizamos el archivo de datos "index-2024-25", el cual tiene las webs guardadas con la estructura "0 ::: 0-00.pl". El id corresponde con el primer número (0 en este caso) y la dirección, con los caracteres que suceden a los tres dobles puntos (0-00.pl).
- cargarSalientes(): Carga en las Webs correspondientes sus webs salientes. Utilizamos el archivo de datos "pld-arcs-1-N-2024-25", el cual tiene los salientes de cada web guardados con la estructura "1 >> 1269678 ### 1430007 ### 2016332". El índice corresponde con el primer número (1 en este caso) y los números siguientes corresponden al índice o id de las webs salientes (En el caso de la web 1, tiene como saliente las webs 1269678, 1430007 y 2016332).
- cargarPalabraClave(ConjuntoPalabras pCp): Se añade a cada Web de ListaWeb las palabras clave que tenga.

- cargarListaEnFicheros(): Escribe la información actualizada de ListaWebs en los ficheros "index-2024-25" y "pld-arcs-1-N-2024-25".
- buscarWeb(String pDir): Busca una web en ListaWebs por medio de su dirección.
- borrarWeb(String pDir): Elimina la web asociada a la dirección del array ListaWebs, dejando su hueco en null, para así guardar el índice.
- añadirWeb(Web pWeb): Añade pWeb de tipo Web al array ListaWebs.
- ordenarWebs(): Ordenamos las webs en un ArrayList.
- salientes(String pWeb): Devuelve un ArrayList de los salientes de una Web dada su dirección.
- word2Webs(String pS): Devuelve las Web que contienen la palabra "pS".
- web2Words(String pWeb): Devuelve las palabras que aparecen en la Web con la dirección dada.
- ident2String(int pX): Se obtiene la dirección de la Web que tenga id x.
- imprimir(): Imprime todas las webs que hay en ListaWebs.

2.3 Clase ConjuntoPalabras

La clase ConjuntoPalabras cuenta con un único atributo, un TreeSet de Strings treePalabras, donde se almacenan las palabras clave. Los métodos implementados en esta clase son:

- cargarPalabrasClave(): Carga en treePalabras todas las palabras que se encuentras en el fichero "words.txt".
- getTreePalabras(): Devuelve el treeSet TreePalabras.
- imprimir(): Se imprimen las palabras de treePalabras.

Apartado 3

Descripción de las estructuras de datos principales

Hemos empleado diversas estructuras de datos para gestionar páginas web, enlaces salientes y palabras clave. Implementamos varias de estas estructuras clave para incrementar tanto la funcionalidad como la eficiencia de nuestro sistema.

3.1 TreeSet:

Dentro de la clase ConjuntoPalabras, optamos por usar un TreeSet para registrar las palabras de un archivo de texto. Esta elección es significativa porque evita la repetición de la misma palabra, lo que es fundamental para mantener nuestra lista en su forma única.

El TreeSet organiza los elementos en un orden específico, de una manera similar a como se disponen las palabras en un diccionario, lo que elimina la necesidad de ordenarlos manualmente después de añadirlos. Las operaciones de insertar, eliminar y buscar elementos tienen un coste de O(log n).

Elegimos TreeSet porque ordena y elimina duplicados de manera automática. Esto facilita la gestión de una lista de palabras en un orden definido, lo que hace que el programa sea más claro y eficiente al trabajar con las palabras.

3.2 ArrayList

En nuestro proyecto, utilizamos ArrayList para gestionar listas de sitios web, palabras clave y enlaces salientes. Esta estructura nos resulta muy útil porque puede adaptarse fácilmente a listas de diferentes tamaños.

Un ArrayList puede crecer o disminuir en tamaño según se requiera, a diferencia de los arrays convencionales, lo que resulta útil cuando no tenemos certeza sobre cuántos elementos necesitaremos. Podemos acceder rápidamente a cualquier sitio web o palabra clave en la lista mediante su índice, lo que es beneficioso cuando necesitamos acceder a los datos en múltiples ocasiones.

ArrayList es la elección ideal para manejar listas que cambian de tamaño y permiten el acceso a sus elementos a través de su posición. Esto nos facilita la gestión de listas de palabras, enlaces a otros sitios y sitios web sin preocuparnos por el tamaño, lo cual simplifica el manejo de la información en diversas partes del proyecto.

3.3 Web[] (Array)

Decidimos usar un array de objetos del tipo Web para almacenar las páginas web en la clase ListaWebs. El array tiene un tamaño fijo que corresponde al número total de sitios web del archivo.

Un array nos permite acceder a un sitio web por su índice, que coincide con el identificador de la web. Esto permite un acceso muy eficiente, con un coste de O(1).

Aunque su tamaño es fijo, a diferencia de un ArrayList, sabemos cuántos sitios web estaremos manejando desde el principio, así que no es necesario ampliarlo. En el caso de eliminar webs, no habrá inconvenientes, ya que al ser un array, la web eliminada solo dejará un espacio como null sin desplazar el resto de elementos, por lo que los índices de los otros elementos seguirán siendo los mismos.

3.4 HashSet

En la clase Web, implementamos un HashSet para almacenar tanto los enlaces salientes como las palabras clave asociadas a cada web.

Un HashSet asegura que un mismo enlace o palabra clave no se añadan más de una vez, algo esencial al trabajar con estos elementos, ya que no tendría sentido incluir el mismo enlace o palabra más de una vez.

Las operaciones rápidas (inserción, eliminación y búsqueda) tienen un coste constante (O(1)), lo cual es realmente valioso cuando manejamos grandes volúmenes de información.

Elegimos HashSet porque brinda operaciones rápidas y evita duplicados, incluso con un gran volumen de enlaces o palabras clave.

Apartado 4

Diseño e implementación de los métodos principales

4.1 Método cargarLista()

```
public void cargarLista() {
//Precondicion: Debe existir un fichero
//          "datuak-2024-2025/index-2024-25"
//          con los id y direcciones URL de las webs
//          con la estructura "0  :::  0-00.pl"
//Postcondicion: Se almacena la informaci\'{o}n
//          en el array listaWeb, cada URL se almacena
//          en la posici\'{o}n que corresponde a su id
```

Casos de prueba:

- Cargar un fichero que existe, pero esta vacío.
- Cargar un fichero que existe y contiene una única web.
- Cargar un fichero que existe y contiene muchas webs.

Este es el código del algoritmo resultante:

```
Scanner entrada = nuevo Scanner(nuevo FileReader(
"datuak-2024-2025/index-2024-25"))
mientras entrada tenga otro elemento que coger repetir
        String linea = siguiente linea de entrada
        String[] lista=separar linea("\\s+:+\\s+")
        int id=convertir int a string(lista[0])
        String dir = lista[1]
        Web web = nueva Web(id, dir)
        this.listaWebs[id] = web
```

Coste: el algoritmo, siempre recorre todos los elementos del fichero. Por lo que su coste será de O(n), lineal.

4.2 Método cargarSalientes()

```
public void cargarSalientes() {
//Precondicion: Debe existir un fichero
//          "datuak-2024-2025/pld-arcs-1-N-2024-25"
//          con los id e \'{i}ndice de los salientes
//Postcondicion: Se almacena en cada web, su
//          conjunto de webs salientes
```

Casos de prueba:

- Cargar un fichero que existe, pero esta vacío.
- Cargar un fichero que existe, que contiene una única web y sus salientes.
- Cargar un fichero que existe, que contiene una única web, pero que no tiene salientes.
- Cargar un fichero que existe, que contiene muchas webs y sus salientes.
- Cargar un fichero que existe, que contiene muchas webs, pero alguna no tiene salientes.

Este es el código del algoritmo resultante:

```
Scanner entrada = nuevo Scanner(nuevo FileReader
("datuak-2024-2025/pld-arcs-1-N-2024-25"))
mientras(entrada tenga otro elemento que coger)
        String linea=siguiente linea de entrada
        String [] datos=separar linea("\\s+>+\\s+")
        int id=convertir integer a string(datos[0])
        si el tamano de datos es mayor de 1 entonces
                String[] salientes=datos[1] separar("\\s+#+\\s+")
                Web web=this.listaWebs[id]
                para todas las posiciones de salientes repetir
                        int saliente=convertir int a string(str)
                        web.anadirEnlace(saliente)
```

Coste: el algoritmo, siempre recorre todos los elementos del fichero. Por lo que su coste será de O(n), lineal.

4.3 Método cargarPalabraClave()

```
public void cargarPalabraClave() {
//Precondicion: Debe existir un fichero
//          "datuak-2024-2025/words.txt"
//          con todas las palabras del diccionario
//Postcondicion: Se almacena en cada web, su
//          conjunto de palabras clave
```

Casos de prueba:

- Cargar un fichero que existe, pero esta vacío.
- Cargar un fichero que existe, que contiene una única palabra.
- Cargar un fichero que existe, que contiene una muchas palabras.

Este es el código del algoritmo resultante:

```
Scanner entrada = nuevo Scanner(nuevo FileReader
("datuak-2024-2025/words.txt"))
mientras entrada tenga otro elemento que coger repetir
        String palabra=siguiente linea de entrada
        para cada web de listaWebs repetir
                si web.tienePalabraClave(palabra) entonces
                        web.anadirPalabraClave(palabra)
```

Coste: el algoritmo, siempre recorre todos los elementos del fichero. Por lo que su coste será de O(n), lineal.

4.4 Método cargarListaEnFicheros()

```
public void cargarListaEnFicheros() {
//Precondicion: -
//Postcondicion: Crea los ficheros
//          "datuak-2024-2025/index-2024-25-ACTUALIZADO"
//          y "datuak-2024-2025/pld-arcs-1-N-2024-25-ACTUALIZADO"
//          actualizando la informacion
```

Casos de prueba:

- Que listaWebs este vacía
- Que listaWebs tenga una web
- Que listaWebs tenga muchas webs

Este es el código del algoritmo resultante:

```
PrintWriter pw = nuevo PrintWriter
("datuak-2024-2025/index-2024-25-ACTUALIZADO", "UTF-8")
PrintWriter pw2 = nuevo PrintWriter
("datuak-2024-2025/pld-arcs-1-N-2024-25-ACTUALIZADO", "UTF-8")
para cada web de listaWebs repetir
    escribir(id de web ::: dir. de web)
    escribir(id de web >>>>)
    para cada saliente de cada web repetir
        escribir(saliente ###)
```

Coste: el algoritmo recorre todas las webs de listaWebs(n) y todos los salientes de cada web(x) siempre. Por lo que el coste del algoritmo será de O(n*x).

4.5 Método buscarWeb(String pDir)

```
public Web buscarWeb(String pDir) {
//Precondicion: Se da la direccion URL de la web
//Postcondicion: Se devuelve la Web asociada
//              a la direccion
```

Casos de prueba:

- Que la Web no exista
- Que la Web exista en la primera posición
- Que la Web exista en la última posición
- Que la Web exista en el medio

Este es el código del algoritmo resultante:

```
boolean enc = false;
Web web = null;
int i=0;
mientras no encontrado e i sea menor
que el tamano de listaWebs repetir
    web=this.listaWebs[i]
    encontrado=web.esLaMismaDir(pDir)
    i=i+1
si no encontrado entonces
    web = null
devolver web
```

Coste: el algoritmo recorre en el peor de los casos todas las webs siendo el coste O(n), lineal.

4.6 Método borrarWeb(String pDir)

```
public void borrarWeb(String pDir) {
//Precondicion: Se da la direccion URL de la web
//Postcondicion: Se borra la web del array y se
//           deja su espacio a null
```

Casos de prueba:

- Que la Web no exista
- Que la Web exista

Este es el código del algoritmo resultante:

```
Web web = this.buscarWeb(pDir)
this.listaWebs[El indice de web]=null
```

Coste: el algoritmo hace uso del programa buscarWeb, que tiene gasto O(n), lineal. Por lo que como la otra instrucción es constante, su coste es también O(n), lineal.

4.7 Método añadirWeb(Web pWeb)

```
public void anadirWeb(Web pWeb) {
//Precondicion: Se da una Web
//Postcondicion: Se anade la web en el espacio
//            correspondiente a su id
```

Casos de prueba:

- Añadir la web dada

Este es el código del algoritmo resultante:

```
this.listaWebs[id de la web]=pWeb;
```

Coste: el algoritmo tiene un coste O(1), constante, ya que la instrucción únicamente coloca un elemento en la posición dada.

4.8 Método ordenarWebs()

```
public ArrayList<String> ordenarWebs(){
//Precondicion: Lo ejecuta listaWebs que ya tiene los datos
//Postcondicion: Se devuelve esa misma lista de webs pero ordenada
```

Casos de prueba (se prueban al ejecutar el programa):

- La lista ya está ordenada
- La lista está desordenada
- La lista está vacía

Este es el código del algoritmo resultante:

```
ArrayList<String> lista=bew ArrayList<String>()
para todas las Webs de la listaWebs repetir
    si web es distinto de null entonces
        lista.add(web.getDireccion())
quickSort(lista,0,lista.size()-1
return lista
```

El método quickSort() se explica a continuación:

4.8.1 Método quickSort(ArrayList<*String*>pLista, int pBajo, int pAlto)

```
private void quickSort(ArrayList<String> pLista,
int pBajo, int pAlto){
//Precondicion: Recibe una lista de Webs, y dos
//          integers que representan el indice superior e inferior
//Postcondicion: La lista recibida se ordena con el metodo QuickSort
```

Este es el código del algoritmo restante:

```
si pBajo<pAlto entonces
    int p=particion(pLista,pBajo,pAlto)
    this.quickSort(pLista,pBajo,p-1)
    this.quickSort(pLista,p+1,pAlto)
```

El método partición() se explica a continuación:

4.8.2 Método particion(ArrayList<*String*>pLista pLista,int pBajo, int pAlto)

```
private int particion(ArrayList<String> pLista,int pBajo, int pAlto) {
//Precondicion: Recibe un ArrayList de Strings y
//              dos integers que vuelven a representar un
//              indice superior e inferior
//Postcondicion: Devuelve la posicion final del pivote,
//              que sera utilizado en el algoritmo QuickSort
```

Este es el código del algoritmo restante:

```
String aux
String pivote = pLista.get(pAlto)
int i = pBajo-1
para un indice j que va desde pBajo hasta pAlto-1 repetir
    si pLista.get(j).compareTo(pivote) es menor o igual que 0 entonces
        i=i+1
        aux=pLista.get(i)
        pLista.set(i,pLista.get(j))
        pLista.set(j,aux)
aux=pLista.get(i+1)
pLista.set(i+1,pLista,get(pAlto))
pLista.set(pAlto,aux)
devolver i+1
```

4.9 Método salientes(String pWeb)

```
public ArrayList<String> salientes(String pWeb){
//Precondicion: Se da un String pWeb
//Postcondicion: Devuelve un ArrayList con las
//              direcciones de los salientes de la web dada
```

Casos de prueba:

- Comprobar que las direcciones que devuelve el programa son las correctas
- Web que no existe

Este es el código del algoritmo resultante:

```
ArrayList<String> lista = new ArrayList<>()
Web web = buscarWeb(pWeb)
si web es distinto de null entonces
    para todos los salientes de la Web repetir
        String dir = this.listaWebs[saliente].getDireccion()
        lista.add(dir)
devolver lista
```

Coste: El algoritmo recorre los salientes de la web, consigue su dirección y la añade, que como peor caso tiene coste O(n), cuando se redimensiona el ArrayList, pero el coste promedio es de O(1). Por lo que el coste es O(n), lineal, siendo n el número de salientes.

4.10 Método word2Webs(String pS)

```
public ArrayList<String> word2Webs(String pS){
//Precondicion: s representa una palabra clave
//Postcondicion: devuelve las webs que contienen la palabra s
```

Casos de prueba:

- Comprobar que devuelve las webs que contengan la palabra s
- Web que no tenga la palabra s

Este es el código del algoritmo resultante:

```
ArrayList<String> lista = new ArrayList<>()
para todas las Web de listaWebs repetir
    si web.getDireccion() contiene pS entonces
        lista.add(web.getDireccion())
devolver lista
```

Coste: El algoritmo recorre todas las web, buscando a ver si estas contienen la palabra clave y la añade, que como peor caso tiene coste O(n), cuando se redimensiona el ArrayList, pero el coste promedio es de O(1). Por lo que el coste es de O(n).

4.11 Método web2Words(String pWeb)

```
public ArrayList<String> web2Words(String pWeb){
//Precondicion: -
//Postcondicion: devuelve las palabras que aparecen en la web
```

Casos de prueba:

- Comprobar que devuelve las palabras de web
- Web que no tiene palabras clave

Este es el código del algoritmo resultante:

```
Web web = this.buscarWeb(pWeb)
si web no es null entonces
    devolver web.getPalabrasClave()
si no
    devolver null
```

Coste: El algoritmo utiliza el método buscarWeb, que tiene un coste de O(n), el lineas del método son constantes. Por lo que el coste es de O(n).

4.12 Método ident2String(int pX)

```
public String ident2String(int pX) {
//Precondicion: x es un valor entero >= 0
//Postcondicion: devuelve la web asociada a x
```

Casos de prueba:

- Comprobar que devuelve la web asociada a x

Este es el código del algoritmo resultante:

```
return this.listaWebs[pX].getDireccion()
```

Coste: El algoritmo tiene una instrucción de coste constante, O(1).

4.13 Método imprimir()

```
public void imprimir()
//Precondicion: -
//Postcondicion: imprime por pantalla la lista de webs
```

Casos de prueba:

- Comprobar que imprima correctamente la lista
- Lista vacía

Este es el código del algoritmo resultante:

```
para todas las webs de listaWebs repetir
    imprimir(web.getId()+"    :::    "+web.getDireccion());
    imprimir("Salientes de "+web.getId()+" : ")
    web.imprimirSalientes()
    imprimir()
```

Coste: El algoritmo recorre todas las webs de listaWebs y posteriormente, recorre todos los salientes de la web. Por lo que el coste es O(n*x), siendo x el total de salientes entre todas las webs.

Apartado 5

Algoritmo

En este apartado se presentará el código de las clases, junto con los programas de prueba o Junits desarrollados.

5.1 Código

5.1.1 Clase ConjuntoPalabras

```
public void cargarPalabras() {
    try {
        Scanner entrada = new Scanner
            (new FileReader("datuak-2024-2025/words.txt"));
        while(entrada.hasNextLine()) {
                String linea = entrada.nextLine();
                treePalabras.add(linea);
        }
        entrada.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public TreeSet<String> getTreePalabras() {
        return treePalabras;
}

public void imprimir() {
        for(String palabra: treePalabras) {
                System.out.println(palabra);
        }
}
```

5.1.2 Clase Web

```
public void anadirPalabraClave(String pPalCla) {
        this.palabrasClave.add(pPalCla);
}
public ArrayList<String> getPalabrasClave(){
        ArrayList<String> lista = new ArrayList<>(this.palabrasClave);
        return lista;
}

public boolean tienePalabraClave(String pPalCla) {
        return this.palabrasClave.contains(pPalCla);
}

public void anadirEnlace(int pId) {
        this.salientes.add(pId);
}

public HashSet<Integer> getSalientes() {
        return this.salientes;
}

public int getId() {
        return this.id;
}

public void setId(int pId) {
        this.id = pId;
}

public String getDireccion() {
        return this.direccion;
}

public void setDireccion(String pDir) {
        this.direccion = pDir;
}

public boolean esLaMismaDir(String pDir) {
        return this.direccion.equals(pDir);
}

public void imprimirSalientes() {
        for(int saliente:this.salientes) {
                System.out.println(saliente+"-");
        }
}
```

5.1.3 Clase ListaWebs

```
public void cargarLista() {
    try {
        Scanner entrada = new Scanner
        (new FileReader("datuak-2024-2025/index-2024-25"));
        while(entrada.hasNextLine()) {
                String linea = entrada.nextLine();
                String[] lista=linea.split("\\s+:+\\s+");
                int id = Integer.parseInt(lista[0]);
                String dir = lista[1];
                Web web = new Web(id, dir);
                this.listaWebs[id]=web;
        }
        entrada.close();
    } catch (IOException e) {
            e.printStackTrace();
     }
}

public void cargarSalientes() {
    try {
        Scanner entrada = new Scanner
        (new FileReader("datuak-2024-2025/pld-arcs-1-N-2024-25"));
        while(entrada.hasNextLine()) {
                String linea = entrada.nextLine();
                String [] datos = linea.split("\\s+>+\\s+");
                int id = Integer.parseInt(datos[0]);
                if(datos.length>1) {
                        String [] salientes = datos[1].split
                        ("\\s+#+\\s+");
                        Web web = this.listaWebs[id];
                        for(String str : salientes) {
                                int saliente = Integer.parseInt(str);
                                web.anadirSaliente(saliente);
                        }
                }
        }
        entrada.close();
    } catch (IOException e) {
            e.printStackTrace();
    }
}
```

```
public void cargarPalabraClave() {
    try {
        Scanner entrada = new Scanner
        (new FileReader("datuak-2024-2025/words.txt"));
        while(entrada.hasNextLine()) {
                String palabra = entrada.nextLine();
                for(Web web: this.listaWebs) {
                        if(web.tienePalabraClave(palabra)) {
                                web.anadirPalabraClave(palabra);
                        }
                }
        }
        entrada.close();
    } catch (IOException e) {
            e.printStackTrace();
    }
}

public void cargarListaEnFicheros() {
    try {
        PrintWriter pw = new PrintWriter
        ("datuak-2024-2025/index-2024-25-ACTUALIZADO", "UTF-8");
        PrintWriter pw2 = new PrintWriter
        ("datuak-2024-2025/pld-arcs-1-N-2024-25-ACTUALIZADO", "UTF-8");
        for(Web web : listaWebs) {
                pw.println(web.getId()+"-:::-"+web.getDireccion());
                pw2.print(web.getId()+"->>>>-");
                for(int saliente: web.getSalientes()) {
                        pw2.print(saliente +"-###");
                }
        }
        pw.close();
    } catch (IOException e) {
            e.printStackTrace(); }
}

public Web buscarWeb(String pDir) {
        boolean enc = false;
        Web web = null;
        int i=0;
        while(!enc && i<this.listaWebs.length){
                web=this.listaWebs[i];
                enc=web.esLaMismaDir(pDir);
                i++;
        }
        if(!enc) {
                web = null;
        }
        return web;
}
```

```
public void borrarWeb(String pDir) {
        Web web = this.buscarWeb(pDir);
        this.listaWebs[web.getId()]=null;
}

public void anadirWeb(Web pWeb) {
        this.listaWebs[pWeb.getId()]=pWeb;
}


public ArrayList<String> ordenarWebs(){
        ArrayList<String> lista= new ArrayList<String>();
        for(Web web:this.listaWebs) {
                if(web!=null) {
                        lista.add(web.getDireccion());
                }
        }
        quickSort(lista,0,lista.size()-1);
        return lista;
}

private void quickSort(ArrayList<String> pLista,int pBajo, int pAlto){
                if (pBajo < pAlto) {
                int p = particion(pLista, pBajo, pAlto);
                this.quickSort(pLista, pBajo, p - 1);
                this.quickSort(pLista, p + 1, pAlto);
            }
}

private int particion(ArrayList<String> pLista,int pBajo, int pAlto) {
        String aux;
        String pivote = pLista.get(pAlto);
    int i=pBajo-1;
    for (int j=pBajo;j<pAlto;j++) {
        if (pLista.get(j).compareTo(pivote) <= 0) {
            i++;
            aux = pLista.get(i);
            pLista.set(i, pLista.get(j));
            pLista.set(j, aux);
        }
    }
    aux= pLista.get(i + 1);
    pLista.set(i + 1, pLista.get(pAlto));
    pLista.set(pAlto, aux);
    return i + 1;
}
```

```
public ArrayList<String> salientes(String pWeb){
    ArrayList<String> lista = new ArrayList<>();
    Web web = buscarWeb(pWeb);
    if(web!=null) {
            for(int saliente: web.getSalientes()) {
                    String dir = this.listaWebs[saliente].getDireccion();
                    lista.add(dir);
            }
    }
    return lista;
}

public ArrayList<String> word2Webs(String pS){
    ArrayList<String> lista = new ArrayList<>();
    for(Web web : this.listaWebs) {
            if(web.getDireccion().contains(pS)) {
                    lista.add(web.getDireccion());
            }
    }
    return lista;
}

public ArrayList<String> web2Words(String pWeb){
    Web web = this.buscarWeb(pWeb);
    if(web!=null) {
            return web.getPalabrasClave();
    }else {
            return null;
    }
}

public String ident2String(int pX) {
    return this.listaWebs[pX].getDireccion();
}

public void imprimir() {
    for(Web web: this.listaWebs) {
            System.out.println(web.getId()+"---:::---"+web.getDireccion());
            System.out.println("Salientes de "+web.getId()+"::");
            web.imprimirSalientes();
            System.out.println();
    }
}
```

5.2 JUnits

Para los casos de prueba de nuestro código, hemos implementado los JUnits de algunos métodos de la clase ListaWebs. Los demás métodos se prueban ejecutando el programa.

```
@Test
public void testBuscarWeb() {
        //La web no existe
        assertNull(lista.buscarWeb("51351251.com"));
        //La web esta en la primera posicion
        Web w=lista.getWebPorId(0);
        assertEquals(w, lista.buscarWeb("0-00.pl"));
        //La web esta en la mitad
        Web w2 =lista.getWebPorId(500000);
        assertEquals(w2, lista.buscarWeb("deventertelevisie.nl"));
        //La web esta en la ultima posicion
        Web w3 =lista.getWebPorId(2039804);
        assertEquals(w3, lista.buscarWeb("zzzz6666.com"));
}

@Test
public void testBorrarWeb() {
        //Borrar una web
        lista.borrarWeb("deventertelevisie.nl");
        Web w =lista.getWebPorId(500000);
        assertNull(w);
}

@Test
public void testAnadirWeb() {
        //Anadir una web
        Web web=lista.getWebPorId(500000);
        lista.borrarWeb("deventertelevisie.nl");
        lista.anadirWeb(web);
        Web webaux=lista.getWebPorId(500000);
        assertEquals(web,webaux);
}

@Test
public void testSalientes() {
        //Comprobar que los salientes de una web son los correctos
        lista.cargarSalientes();
        ArrayList<String> al=new ArrayList<String>();
        al=lista.salientes("0-100.com.cn");
        assertEquals(al.get(0),"pooban.com");
        assertEquals(al.get(1),"yojochina.com");
        assertEquals(al.get(2),"nen.net.cn");
}
```

```
@Test
public void testIdent2String() {
        //Comprobar que se devuelve la direccion correcta
        assertEquals("0-00.pl",lista.ident2String(0));
}
```

Apartado 6

Conclusiones

Para la resolución del problema planteado,ha sido necesario para nosotros comprender y aprender a implementar nuevas estructuras de datos así como implementar con ellas los métodos necesarios para completar la práctica. Además de emplear estructuras ya conocidas como los Array List y los array, hemos implementado nuevas estructuras que han sido muy útiles como los HashSet y los TreeSet. Además, hemos podido aprender nuevas utilidades de los Array List.

Por otro lado, hemos podido poner en práctica los algoritmos de ordenación tratados en clase, hemos optado por el algoritmo de ordenación QuickSort, ya que suponía menos tiempo de ejecución respecto al resto. Los tiempo de ejecución de cada uno, en el método ordenarLista(), para su comparación están representados en la tabla 6.1.

Algoritmos para ordenación		
Algoritmo de Ordenación	**Coste O()**	**Tiempo**
Burbuja	n^2	4000
Selección	n^2	2000
Inserción	n^2	2000
MergeSort	$nlogn$	41
QuickSort	$nlogn$	41

Table 6.1: Resultados (en segundos) para diferentes algoritmos, tomando n=2000000 que se acerca a nuestro caso.