Aplicación para gestionar páginas web

Fecha: 15-10-2017

Participantes:

Alejandro Ye
Urtzi González
Zuhaitz Martínez
David Cuesta

2 Introducción

El objetivo de esta práctica es implementar una aplicación que permita gestionar un número grande de páginas web, de modo que se permitan ejecutar las siguientes funcionalidades

Búsqueda de una web determinada a partir de las palabras clave contenidas en su URL

Añadir un nuevo elemento (web o palabra clave)

Obtener la lista de webs ordenada alfabéticamente

Obtener la lista de webs con las que enlaza cada web

Como esta práctica es puramente instructiva obtendremos una lista de Webs donde se encuentran todas las webs que se deberá tener en cuenta en nuestra base de datos, así como un Diccionario en el que se incluyen todas las palabras clave permitidas.

De este modo deberemos implementar un sistema que permita realizar las siguientes operaciones:

Cargar los datos desde los ficheros (Webs y Diccionario)

Realizar la búsqueda de una página web en función de las palabras clave

Inserción de una nueva página web a la lista de páginas web

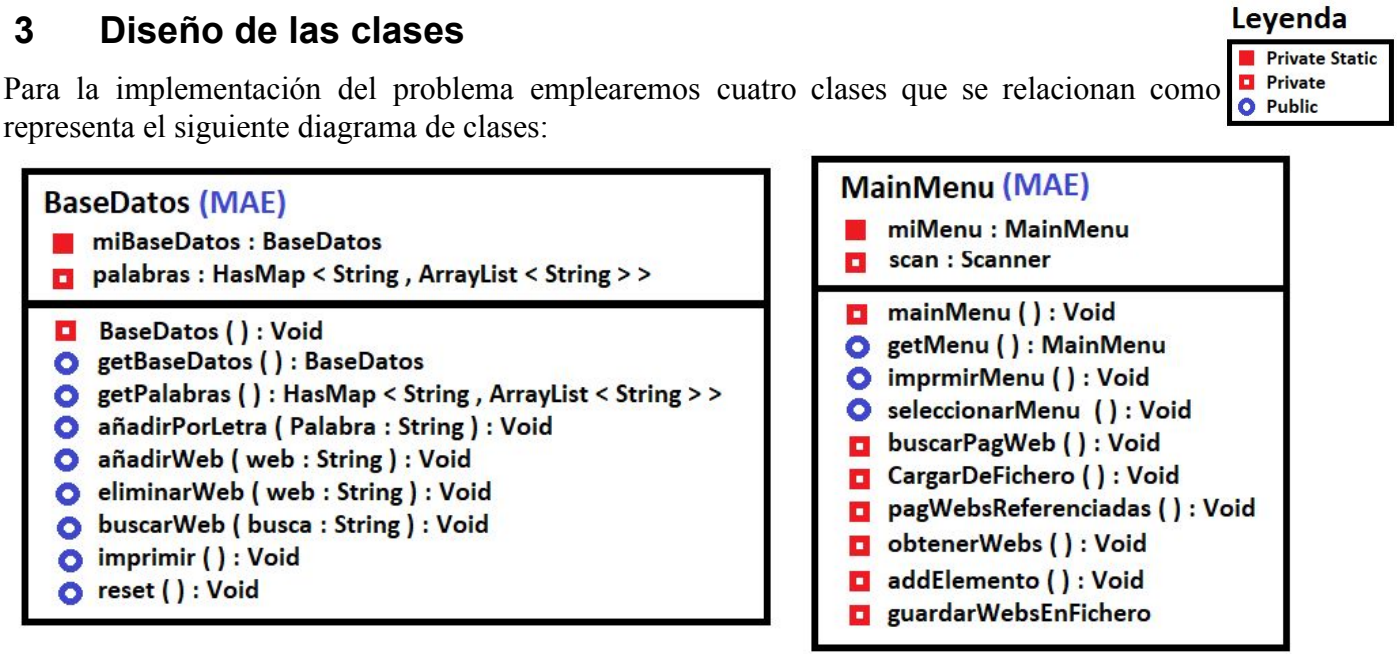
Devolver (no imprimir) las páginas web enlazadas desde una web dada

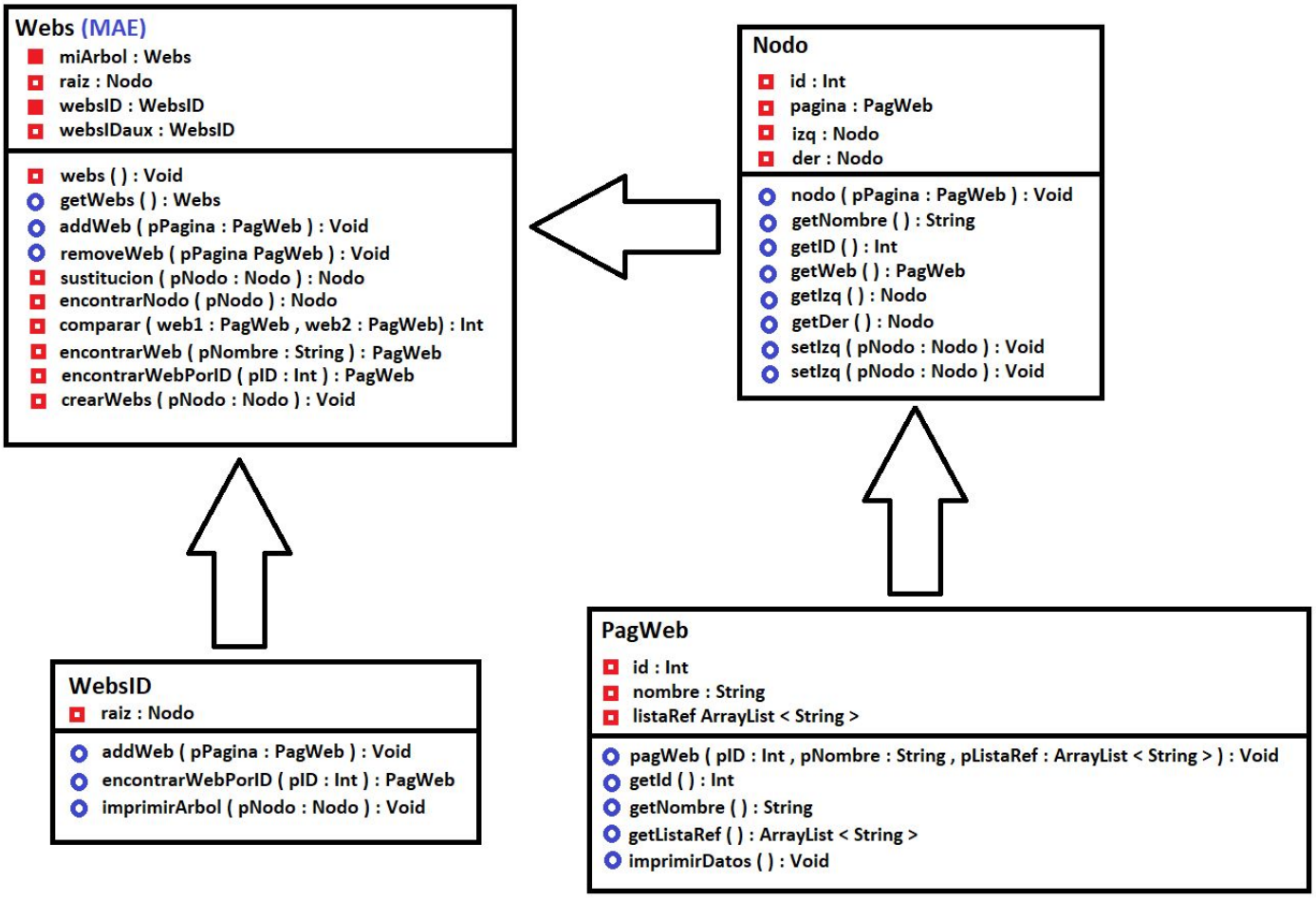
Guardar la lista de webs en ficheros

Obtener una lista de páginas web ordenada (alfabéticamente).

3 Diseño de las clases

Para la implementación del problema emplearemos cuatro clases que se relacionan como representa el siguiente diagrama de clases:





4 Descripción de las estructuras de datos principales

Después de informarnos acerca de las diferentes estructuras de datos:

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Concluimos que la solución más eficiente para el problema planteado consistía en emplear:

- **La estructura HashTable** para la implementación del diccionario. Debido a que este modo tiene un coste de búsqueda, inserción y eliminación constantes O(1) nos permite acceder a las palabras clave almacenadas de una forma muy dinámica
- **La estructura de Árbol Binario** para la implementación de la estructura de webs. Debido a que este método tiene un coste de búsqueda, inserción y eliminación logarítmicos con respecto al número de elementos almacenados O(log(n)) nos permite ordenar alfabéticamente las páginas web de una forma muy dinámica

5 Diseño e implementación de los métodos principales

A continuación se describe la implementación las diferentes clases y sus métodos principales:

Nota: se han omitido los getters, setters y constructoras de las clases

5.1 BaseDatos (MAE)

Esta clase almacena las palabras reales del diccionario mediante la estructura Tabla Hash empleando como llave la propia String de la palabra clave y asociando cada una de las palabras reales del diccionario con todas las páginas web en las que se emplea dicha palabra clave mediante un String.

5.1.1 public void anadirPorLetra(String palabra)

Pre condición: Recibe una String que no debe ser vacía

Post condición: Método que permite introducir una palabra en el diccionario e inicializa la lista de Strings donde se almacenarán los nombres de las webs que emplea dicha palabra.

Descripción: Comprueba si dicha palabra ya está añadida al diccionario, la añade e inicializa el array de Strings. En caso de que ya existiera no realiza ninguna operación.

Coste: O(n) siendo n el número de webs que hay que introducir

5.1.2 public void anadirWeb(String web)

Pre condición: Recibe una String que no debe ser vacía

Post condición: Añade un URL a la lista de webs de una palabra clave

Descripción: Analiza la String de entrada dividiéndola en todas las posibles combinaciones de sub Strings de longitud 4 (o menor) y superior hasta alcanzar la longitud total de la String de entrada. A continuación comprueba para cada posible subString si es una palabra clave contenida en el diccionario usando como Key dicha sub String y la añade a la ArrayList de dicha palabra clave si la encuentra.

Coste: O(n*m) siendo n el número de webs que hay que introducir y m la longitud de la palabra entrante.

5.1.3 public void buscarWeb(String[] busca)

Pre condición: Recibe una String que no debe ser vacía y pueda contener una o varias sub Strings separadas por espacios

Post condición: Identifica las páginas webs asociadas a una o varias palabras clave

Descripción: Analiza la String de entrada dividiéndola varias sub Strings en función de los espacios que encuentre. A continuación analiza todas las posibles combinaciones de sub Strings de longitud 4 (o menor) y superior hasta alcanzar la longitud total de la Sub String previamente dividida por espacios y comprueba para cada posible subString si es una palabra clave contenida en el diccionario usando como Key dicha sub String y devuelve todas las páginas web contenidas en los arrays de URL de cada palabra clave encontrada

Coste: O(n*m) siendo n la longitud de la palabra entrante y m el número de palabras en el array.

5.1.4 public void eliminarWeb(String web)

Pre condición: Recibe una String que no debe ser vacía

Post condición: Elimina una página web asociadas a una palabra clave

Descripción: Analiza la String de entrada dividiéndola en todas las posibles combinaciones de sub Strings de longitud 4 y superior hasta alcanzar la longitud total de la String de entrada. A continuación comprueba para cada posible subString si es una palabra clave contenida en el diccionario usando como Key dicha sub String y la elimina de la ArrayList de dicha palabra clave si la encuentra.

Coste: O(n+m) siendo n la longitud de la String de entrada y m la longitud de la lista donde se encuentra (Coste del método remove() del ArrayList).

5.2 PagWeb

Esta clase almacena la información de cada página web: su identificador, su URL y la lista de webs que son referenciadas desde cada página web

5.2.1 public void imprimirDatos()

Descripción: Método que permite imprimir por pantalla toda la información de

Precondición: Ninguna

Post condición: imprime por pantalla la información de la página web

Coste: constante

5.3 Nodo

Esta clase almacena la información de cada Nodo del árbol binario; su identificador, la página web que contiene dicho nodo y los dos nodos hijos a los que hace referencia, uno a su izquierda y otroa su derecha

5.4 Webs (MAE)

Esta clase almacena las webs ordenadas alfabéticamente y por ID mediante la estructura Árbol Binario. De este modo es posible encontrar rápidamente la web que deseemos

5.4.1 public void addWeb (PagWeb pPagina)

Descripción: Método que permite introducir web en el árbol binario de forma ordenada

Pre condición: Recibe una página web

Post condición: si el árbol original esta vacío crea uno nuevo con el parámetro de entrada como raíz, y de lo contrario añade el parámetro de entrada en su posición ordenada. Para ello recorre el árbol comparando el elemento de cada nodo con el que se quiere introducir y seleccionando la rama de la izquierda si es mayor (considerando a>z) o la de la derecha si es menor (en el caso de encontrar un elemento igual lo añade a su derecha).

Coste: O(n^2) siendo n el número de nodos del árbol. Nota: Este método solo se llama una vez para inicializar el programa y solo tiene coste O(n^2) en el caso de que esté ordenada la lista inicial, sino tiene coste O(n*log(n)) siendo n el número de webs.

5.4.2 public void removeWeb (PagWeb pPagina)

Descripción: Método que permite eliminar la primera aparición de una web del árbol

Pre condición: Recibe una página web

Post condición: recorre el árbol comparando el elemento de cada nodo con el que se quiere eliminar y seleccionando la rama de la izquierda si es mayor (considerando a>z) o la de la derecha si es menor hasta que se encuentre un nodo igual al que se quiere eliminar o se termine de recorrer todo el árbol, si no encuentra la página web buscada no hace nada al árbol.

Coste: O(log(n)) siendo n el número de nodos del árbol.

5.4.3 private PagWeb encontrarWeb (String pNombre)

Descripción: Método que permite encontrar una página web en función de su nombre

Pre condición: Recibe un String no vacío

Post condición: recorre el árbol comparando el elemento de cada nodo con el que se quiere encontrar y seleccionando la rama de la izquierda si es mayor (considerando a>z) o la de la derecha si es menor hasta que se encuentre el nodo que tiene dicho URL o se termine de recorrer todo el árbol. Si no se ha encontrado una web con pNombre devuelve null.

Coste: O(log(n)) siendo n el número de nodos del árbol.

5.4.4 private PagWeb encontrarWebPorID (int pID)

Descripción: Método que permite encontrar una página web en función de su ID

Pre condición: -

Post condición: recorre el árbol que esta ordenado por id comparando el elemento de cada nodo con el que se quiere encontrar y seleccionando la rama de la derecha si es mayor y la de la izquierda si es menor hasta que se encuentre el nodo que tiene dicho ID o se termine de recorrer todo el árbol.

Coste: O(log(n)) siendo n el número de nodos del árbol.

5.4.5 protected Nodo sustitucion (Nodo pNodo)

Descripción: Método que permite sustituir un nodo por otro

Pre condición: Recibe el nodo que se pretende sustituir

Post condición: devuelve la referencia a sustituir de pNodo, si tiene hijos (es decir, si es un subarbol) devuelve el subarbol de manera InOrder

Coste: O(log(n)) siendo n el número de nodos del árbol a partir del pNodo de entrada.

5.4.6 private void imprimirArbol (Nodo pNodo)

Descripción: Método que permite imprimir la información de los nodos inferiores al nodo introducido.

Pre condición: si se quiere imprimir el arbol entero pNodo debe ser la raiz

Post condición: se imprimir el arbol desde el pNodo dado de manera InOrder

Coste: O(n) siendo n el el número de nodos del árbol.

5.4.7 private Integer comparar (PagWeb web1, PagWeb web2)

Descripción: Método que decide cuál de las dos páginas web introducidas como parámetro es alfabeticamente mayor que la otra.

Pre condición: recibe dos webs (utiles).

Post condición: devuelve mediante un integer la compraracion alfabetica entre las dos webs, si web1 es mas pequena (a<z) devuelve un integer positivo, si son iguales 0 y sino negativo.

Coste: O(1)

5.4.8 public void crearWebsID(Nodo pNodo)

Descripción: Método que crea un árbol idéntico al actual pero ordenado por id

Pre condición: pNodo va a ser la raíz del nuevo árbol por lo que hay que pasarle la raiz del arbol actual si se quiere crear uno igual

Post condición: crea un árbol de búsqueda idéntico al actual pero ordenado por ID en vez de alfabeticamente

Coste: O(n^2) siendo n el número de nodos del árbol. Nota: Este método solo se llama una vez para inicializar el programa y solo tiene coste O(n^2) en el caso de que esté ordenada la lista inicial, sino tiene coste O(n*log(n)) siendo n el número de webs.

5.4.9 private PagWeb buscarWebReferenciadas(PagWeb pPagina, int pPos)

Descripción: busca por id en el arbol la web que esta en pPos del arraylist listaRef que tiene que pagina web

Pre condición: -

Post condición: devuelve la PagWeb a la que pPagina referencia de su atributo listRef en la posicion pPos, si no hay ningun elemento en pPos devuelve null

Coste: O(n+log(k)) siendo n el numero de elementos del arraylist listaRef y siendo k el número de nodos del árbol.

5.4.10 private ArrayList<PagWeb> devolverWebsReferenciadas(PagWeb pPagina)

Descripción: devuelve todas las webs que son referenciadas por pPagina

Pre condición: -

Post condición: devuelve un array list con objetos de clase PagWeb estando el array rellenado con las PagWeb a las que referencia pPagina

Coste: O(n+n*log(k)) siendo n el numero de elementos del arraylist listaRef y siendo k el número de nodos del árbol.

5.5 WebsID

Esta clase almacena permite al árbol binario ordenarse en función del identificador de la página web, de modo que contiene un segundo árbol ordenado en función del ID de la página web

5.5.1 public void addWeb (PagWeb pPagina)

Descripción: Método que introduce web en el árbol binario de forma ordenada por ID

Pre condición: Recibe una página web

Post condición: si el árbol original está vacío crea uno nuevo con el parámetro de entrada como raíz, y de lo contrario añade el parámetro de entrada en su posición ordenada. Para ello recorre el árbol comparando el elemento de cada nodo con el que se quiere introducir y seleccionando la rama de la derecha si es mayor y la de la izquierda si es menor (en el caso de encontrar un elemento igual lo añade a su derecha)

Coste: O(n) siendo n el el número de nodos del árbol.

5.5.2 private PagWeb encontrarWebPorID (int pID)

Descripción: Método que permite encontrar una página web en función de su ID

Pre condición: -

Post condición: recorre el árbol comparando el elemento de cada nodo con el que se quiere encontrar y seleccionando la rama de la izquierda si es mayor y la de la derecha si es menor hasta que se encuentre el nodo que tiene dicho ID o se termine de recorrer todo el árbol.

Coste: O(log(n)) siendo n el número de nodos del árbol a partir del pNodo de entrada.

5.5.3 private void imprimirArbol (Nodo pNodo)

Descripción: Método que permite imprimir la información de los nodos inferiores al nodo introducido.

Pre condición: si se quiere imprimir el arbol entero pNodo debe ser la raiz

Post condición: se imprimir el arbol desde el pNodo dado de manera InOrder

Coste: O(n) siendo n el el número de nodos del árbol.

5.6 MainMenu

Esta clase almacena contiene un bucle infinito que tras inicializar la base de datos y los árboles binarios con la información contenida en los ficheros de texto proporcionados por el profesor pregunta al usuario cual es la acción que desea realizar, permitiéndole escoger entre los diferentes objetivos del proyecto o finalizar el programa:

Buscar una página web

Añadir nuevo elemento (web o palabra clave)

Obtener la lista de webs ordenada",

Devolver las páginas web enlazadas desde una web dada

Guardar la lista de webs en ficheros

Cargar datos desde un fichero (Dado que se ordenan durante la inicialización, ya estarán ordenados alfabéticamente)

Salir

5.6.1 public void imprimirMenu()

Descripción: Método que imprime por pantalla las distintas tareas que puede llevar a cabo el programa y sugiere al usuario que escoja una

Pre condición: Ninguna

Post condición: Imprime por pantalla

Coste: constante

5.6.2 public void seleccionarMenu() throws FileNotFoundException

Descripción: Método que lee por teclado la opción elegida por el usuario y ejecuta las operaciones correspondientes para llevar a cabo la tarea

Pre condición: Ninguna

Post condición: lee por teclado

Coste: depende del usuario

5.6.3 private void buscarPagWeb()

Descripción: Método que pide al usuario que introduzca por teclado una o varias String separadas por espacios y devuelve todas las páginas web asociadas a cualquier sub String de longitud cuatro o mayor que se encuentre contenida en el diccionario como una palabra clave

Pre condición: Ninguna

Post condición: llama al método buscarWeb() de la clase BaseDatos con la String que el usuario ha introducido por teclado como argumento de entrada

Coste: Lineal con respecto a la longitud de la String de entrada

5.6.4 private void cargarDeFichero() throws FileNotFoundException

Descripción: Método que lee los ficheros de texto proporcionados por el profesor y carga los datos en la base de datos y los árboles binarios

Pre condición: si el fichero no existe o tiene un nombre distinto se lanza la excepción FileNotFoundException que termina la ejecución del programa

Post condición: llama sucesivamente a los métodos anadirPorLetra y añadirWeb(String) de la base de datos y addElementListaRef(idPagina) del árbol binario

Coste: número de palabras del diccionario + número de webs * número medio de webs referenciadas por cada web + número de webs * número medio de palabras

5.6.5 private void pagWebReferenciadas()

Descripción: Método que representa por pantalla la lista de referencias de una web a partir del ID que introduzca el usuario por teclado

Pre condición:

Post condición: llama al método encontrarWebPorID(Integer) de la clase Webs que busca en el árbol binario la web solicitada y devuelve la lista de webs referenciadas por dicha web

Coste: logarítmico con respecto al número de webs del árbol binario

5.6.6 private void obtenerWebs()

Descripción: Método que llama los métodos del árbol binario e imprime su contenido de forma ordenada (alfabéticamente como se pide y si se quiere numéricamente por ID **también)**

Pre condición: Que el árbol se haya creado sino dará null pointer exception

Postcondición: llama al método imprimirArbol (Nodo) de la clase Webs que busca en el árbol binario la web solicitada y devuelve la lista de webs referenciadas por dicha web

Coste: lineal con respecto al número de webs del árbol binario

5.6.7 private void addElemento()

Descripción: Método que permite añadir una página web o una palabra clave

Pre condición: String introducida sea una opción válida

Postcondición: llama al método addWord() o addPagWeb()

Coste: constante

5.6.8 private void addWord()

Descripción: Método que añade una palabra clave al hashmap

Pre condición: String introducida sea una opción válida

Post condición: llama al método anadirPorLetra(String) de la clase BaseDatos y añade el string como palabra clave para que se pueda utilizar pàra clasificar páginas web

Coste: constante

5.6.9 private void addPagWeb()

Descripción: Método que añade una página web al árbol binario

Pre condición: String introducida sea una opción válida

Post condición: llama al método addWeb(PagWeb) de la clase Webs para añadir la página al árbol y añadirWeb(String) de la clase BaseDatos para clasificar por palabras clave la página nueva.

Coste: logarítmico con respecto al número de webs del árbol binario

5.6.10 private void guardarWebsEnFichero()

Descripción: Método que recorre en orden el árbol binario ordenado alfabéticamente y guarda el URL de las webs en un fichero de texto

Pre condición: Que exista la estructura donde se guardan las páginas web en nuestro caso el árbol binario

Post condición: llama al método createNewFile() de la propia clase de java.io.File y crea un objeto de la clase java.io.printwriter para escribir dentro de un archivo de texto. Para eso llama al método guardar(Nodo, PrintWriter, String) de la clase Webs que itera para guardar todos los nodos imprimiendo sus datos en un archivo de texto.

Coste: lineal con respecto al número de webs del árbol binario

5.7 Casos de prueba:

Buscar una página web

- La página web no existe
- El árbol está vacío
- El árbol existe
    - Es la primera
    - Es la última
    - Está en una posición intermedia

Añadir nuevo elemento (web o palabra clave)

- El árbol Webs no existe
- El árbol Webs si existe
    - Está vacía y es la primera
    - Es la última
    - Está en una posición intermedia

- La base de datos no existe
- La base de datos si existe
    - Está vacía y es la primera
    - Es la última
    - Está en una posición intermedia

Obtener la lista de webs ordenada

- El árbol Webs no existe
- El árbol Webs si existe
    - Está vacía
    - Sólo tiene un elemento
    - Tiene varios elementos

Devolver las páginas web enlazadas desde una web dada

- El árbol Webs no existe
- El árbol Webs si existe
    - Está vacía
    - Sólo tiene un elemento
        - La web no tiene lista
        - La web no tiene ningún enlace (lista vacía)
        - La web sólo un elemento

        - la web tiene varios elementos
    - Tiene varios elementos
        - Encuentra la web
        - No está vacío
        - La web no tiene lista
        - La web no tiene ningún enlace (lista vacía)
        - La web sólo un elemento
        - la web tiene varios elementos

Guardar la lista de webs en ficheros

- No hay fichero
- Hay fichero
    - Está vacía
    - Tiene texto
    - Hay un árbol webs
        - El árbol está vacío
        - Sólo tiene un elemento
        - Tiene varios elementos
        - Recorre bien recursivamente tanto por la derecha como por la izquierda

Cargar datos desde un fichero

- No hay fichero
- Hay fichero
    - Está vacía
    - Tiene texto
- Lee hasta el final
- Separa bien las strings

6 Código

```
    //Metodo que se utilizara para eliminar el nombre de una Web en las palabras del Diccionario.
    //pre: Un nombre no vacio de una web.
    //post: Elimina el nombre de la web en todas las palabras que utiliza esa web.
    public void eliminarWeb(String web)
    {
        String[] w = web.split("\\.");
        String palabra = w[0];

        for (int l = 4; l<=palabra.length(); l++)
        {
            for (int pos = 0; pos<=(palabra.length()-l); pos++)
            {
                String posible = palabra.substring(pos, pos+l);
                if(palabras.containsKey(posible)){ palabras.get(posible).remove(web); }
            }
        }
    }

    //pre: Recibe lo que ha sido escrito por teclado.
    //post: Devuelve la lista de paginas relacionadas con la palabra.
    public void buscarWeb(String[] busca)
    {
        for (int i = 0; i< busca.length; i++)
        {
            for (int l = 4; l<=busca[i].length(); l++)
            {
                for (int pos = 0; pos<=(busca[i].length()-l); pos++)
                {
                    String posible = busca[i].substring(pos, pos+l);
                    if(palabras.containsKey(posible)){ System.out.println(palabras.get(posible)); }
                }
            }
        }
    }

    //Imprime todas las keys y sus listas.
    public void imprimir()
    {
        for(String key: palabras.keySet()){ System.out.println(key + " = " + palabras.get(key)); }
    }

    //Vacia el hashmap.
    public void reset(){ this.palabras.clear(); }
}
```

```
class Nodo
{

// INICIALIZACIONES
        private int id; // el nodo recibe el mismo id que la pagina web que contiene
        private PagWeb pagina;
        private Nodo izq;
        private Nodo der;

// CONSTRUCTORA
        public void Nodo(PagWeb pPagina)
        {
            this.id = pPagina.getId();
            this.pagina = pPagina;
            this.izq = null;
            this.der = null;
        }

        public void print()
        {
            System.out.println(this.pagina.getNombre() + " (id: " + this.id + ")");
        }

// GETTERS Y SETTERS
        public String getNombre(){ return this.pagina.getNombre(); }

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

        public PagWeb getWeb(){ return this.pagina; }

        public Nodo getIzq(){ return this.izq; }

        public Nodo getDer(){ return this.der; }

        public void setIzq(Nodo pNodo){ this.izq = pNodo; }

        public void setDer(Nodo pNodo){ this.der = pNodo; }
```

```java
package eda;

import java.util.ArrayList;

public class PagWeb{

// ATRIBUTOS

    private int id;
    private String nombre;
    private ArrayList<String> listaRef = new ArrayList<>();

// METODOS

    //Constructora

    public PagWeb (int pId, String pNombre, ArrayList<String> pListaRef)
    {
        this.id = pId;
        this.nombre = pNombre;
        this.listaRef = pListaRef;
    }

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

    public String getNombre() { return this.nombre; }

    public ArrayList<String> getListaRef() { return this.listaRef; }

    public void imprimirDatos()
    {
        System.out.println("ID: " + this.id + "Nombre: " + this.nombre + "Nombre: " + this.listaRef + "\n");
    }


public class WebsID
{

// INICIALIZACIONES

    Nodo raiz = null;

// METODOS

    public Nodo getRaiz(){ return this.raiz; }

    public void addWeb (PagWeb pPagina)
    {
        // crea un nodo y lo inicializa
        Nodo nuevoNodo = new Nodo(pPagina);

        // si no hay raiz se convierte en raiz
        if (raiz == null) { raiz = nuevoNodo; }

        else
        {
            // establece raiz como el nodo inicial a comparar
            Nodo nodoAuxiliar = raiz;
            boolean anadido = false;

            while (!anadido)
            {
                // por la izquierda
                if (nuevoNodo.getId() < nodoAuxiliar.getId())
                {
                    if (nodoAuxiliar.getIzq() == null)
                    {
                        nodoAuxiliar.setIzq(nuevoNodo);
                        anadido = true;
                    }
                    else { nodoAuxiliar = nodoAuxiliar.getIzq(); }

                // por la derecha
                } else
                {
                    if (nodoAuxiliar.getDer() == null)
                    {
                        nodoAuxiliar.setDer(nuevoNodo);
                        anadido = true;
                    }
                    else{ nodoAuxiliar = nodoAuxiliar.getDer(); }
                }
            }
        }
    }
```

```
public PagWeb encontrarWebPorID (int pID)
{
    // empezamos arriba del arbol
    Nodo nodoAuxiliar = raiz;

    while (nodoAuxiliar.getWeb().getId() != pID)
    {
        //compara los ID
        if ( nodoAuxiliar.getWeb().getId() > pID) { nodoAuxiliar = nodoAuxiliar.getIzq(); }
        else { nodoAuxiliar = nodoAuxiliar.getDer(); }

        // si el nodo no esta
        if (nodoAuxiliar == null) { return null; }
    }
    return nodoAuxiliar.getWeb();
}

//pre: si se quiere imprimir el arbol entero pNodo debe ser la raiz
//pre: se imprimir el arbol desde el pNodo dado de manera InOrder
public void imprimirArbol (Nodo pNodo)
{
    if (pNodo != null)
    {
        imprimirArbol(pNodo.getDer());
        pNodo.print();
        imprimirArbol(pNodo.getIzq());
    }
}
```

```
 public class Webs {

 // INICIALIZACIONES

      private static Webs miArbol = null;
      Nodo raiz = null;

      static WebsID websID = new WebsID();

      WebsID websIDaux = new WebsID();

// CONSTRUCTORA
     private Webs(){}

     public static Webs getWebs()
     {
         if(miArbol == null){ miArbol = new Webs(); }
         return miArbol;
     }
```

```
// METODOS
    private void addWeb (PagWeb pPagina)
    {
        // crea un nodo y lo inicializa
        Nodo nuevoNodo = new Nodo(pPagina);

        // si no hay raiz se convierte en raiz
        if (raiz == null) { raiz = nuevoNodo; }

        else{
            // establece raiz como el nodo inicial a comparar
            Nodo nodoAuxiliar = raiz;
            boolean anadido = false;

            while (!anadido) {

                int comparacion = comparar((nuevoNodo.getWeb()), (nodoAuxiliar.getWeb()));

                // por la izquierda
                if (comparacion > 0)
                {
                    if (nodoAuxiliar.getIzq() == null)
                    {
                        nodoAuxiliar.setIzq(nuevoNodo);
                        anadido = true;
                    }
                    else{ nodoAuxiliar = nodoAuxiliar.getIzq(); }
                }

                // Por la derecha
                else
                {
                    if (nodoAuxiliar.getDer() == null)
                    {
                        nodoAuxiliar.setDer(nuevoNodo);
                        anadido = true;
                    }
                    else{ nodoAuxiliar = nodoAuxiliar.getDer(); }
                }
            }
        }
    }


    @SuppressWarnings("unused")
    private PagWeb encontrarWeb (String pNombre)
    {
        pNombre = pNombre.toLowerCase();

        Nodo nodoAuxiliar = raiz; // empezamos arriba del arbol

        while ( (nodoAuxiliar.getWeb().getNombre()) != (pNombre) )
        {
            //compara los nombres
            if ( nodoAuxiliar.getNombre().toLowerCase().compareTo(pNombre.toLowerCase()) < 0)
            {
                nodoAuxiliar = nodoAuxiliar.getIzq();
            }
            else { nodoAuxiliar = nodoAuxiliar.getDer(); }

            // si el nodo no esta
            if (nodoAuxiliar == null) { return null; }
        }
        return nodoAuxiliar.getWeb();
    }
```

```
private void removeWeb (PagWeb pPagina)
{
    if (raiz == null){ return; }

    else
    {
        int compRaiz = comparar(raiz.getWeb(),pPagina);

        if(compRaiz == 0){ raiz = sustitucion(raiz);}
        else
        {
            // establece raiz como el nodo inicial a comparar
            Nodo actual = raiz;
            Nodo padre = raiz;
            boolean borrado = false;
            int comparacion = comparar(pPagina,actual.getWeb());

            if (comparacion > 0){ actual = actual.getIzq(); }
            else{ actual = actual.getDer(); }

            while ( (actual != null) && (!borrado) )
            {
                int comp1 = comparar(pPagina, actual.getWeb());

                if (comp1 == 0)
                {
                    borrado = true;
                    int comp2 = comparar(padre.getIzq().getWeb(), actual.getWeb());
                    if (comp2 == 0){ padre.setIzq(sustitucion(actual)); }
                    else{ padre.setDer(sustitucion(actual)); }
                }
                else
                {
                    padre = actual;
                    int comp3 = comparar(pPagina, actual.getWeb());
                    if (comp3 > 0){ actual = actual.getIzq(); }
                    else{ actual = actual.getDer(); }
                }
            }

            if (!borrado){ return; }
        }
    }
}
```

```
private void crearWebsID(Nodo pNodo)
{
    if (pNodo != null)
    {
        crearWebsID(pNodo.getIzq());
        websIDaux.addWeb(pNodo.getWeb());
        crearWebsID(pNodo.getDer());
    }

    websID = websIDaux;
}
```

```
protected Nodo sustitucion (Nodo pNodo)
{
    Nodo resultado = null;
    if ((pNodo.getIzq() == null)&&(pNodo.getDer()==null)){ resultado = null; }

    else if ((pNodo.getIzq() != null)&&(pNodo.getDer() == null)) { resultado = pNodo.getIzq(); }

    else if ((pNodo.getIzq() == null)&&(pNodo.getDer() != null)) { resultado = pNodo.getDer(); }

    else
    {
        Nodo actual = pNodo.getDer();
        Nodo padre = pNodo;

        while (actual.getIzq() != null)
        {
            padre = actual;
            actual = actual.getIzq();
        }

        int comp = comparar (pNodo.getDer().getWeb(), actual.getWeb());

        if (comp == 0) { actual.setIzq(pNodo.getIzq()); }
        else
        {
            padre.setIzq(actual.getDer());
            actual.setDer(pNodo.getDer());
            actual.setIzq(pNodo.getIzq());
        }
        resultado = actual;
    }
    return resultado;
}
```

```
private PagWeb encontrarWebPorID (int pID)
{
    PagWeb webEncontrada = null;
    webEncontrada = websID.encontrarWebPorID(pID);
    return webEncontrada;
}

@SuppressWarnings("unused")
private void imprimirArbol (Nodo pNodo)
{
    if (pNodo != null)
    {
        imprimirArbol(pNodo.getDer());
        pNodo.print();
        imprimirArbol(pNodo.getIzq());
    }
}

private Integer comparar (PagWeb web1, PagWeb web2)
{
    //creamos nuevas variable string
    String nombreWeb1 = new String();
    String nombreWeb2 = new String();

    //almacenamos el nombre las dos webs a comparar en minuscula
    nombreWeb1 = web1.getNombre().toLowerCase();
    nombreWeb2 = web2.getNombre().toLowerCase();

    //las comparamos
    int resultado = nombreWeb1.compareTo(nombreWeb2);

    return resultado;
}

public Nodo encontrarNodo(Nodo pNodo)
{
    Nodo nodoAuxiliar = raiz; // empezamos arriba del arbol

    int comparacion = comparar((pNodo.getWeb()), (nodoAuxiliar.getWeb()));

    // seguir buscando mientras que no lo ha encontrado
    while (comparacion != 0)
    {
        if (comparacion > 0) { nodoAuxiliar = nodoAuxiliar.getIzq(); }
        else { nodoAuxiliar = nodoAuxiliar.getDer(); }

        // si el nodo no esta
        if (nodoAuxiliar == null) { return null; }
    }
    return nodoAuxiliar;
}
```

```java
@SuppressWarnings("unused")
public class MainMenu {

    //Esto es lo que se ejecutara mientras este encendido
    public static void main(String[] args)
    {
        // "Empieza" el programa
        MainMenu miMenu = MainMenu.getMenu();
        BaseDatos miBase = BaseDatos.getBaseDatos();

        try
        {
            miMenu.cargarDeFichero();
            System.out.print("Operacion completada\n");
            miMenu.imprimirMenu();
            miMenu.seleccionarMenu();
        }

        catch(FileNotFoundException nSe)
        {
            System.out.print("Agur no se ha encontrado los archivos de texto\n");
            nSe.printStackTrace();
        }
    }
   //Atributos del Menu principal

       private Scanner scan;
       private static MainMenu miMenu = null;

   //Constructora
       private MainMenu() { }


   //Metodos del menu principal
   public static MainMenu getMenu()
   {
       if(miMenu == null) { miMenu = new MainMenu(); }
       return miMenu;
   }

   public void imprimirMenu()
   {
       System.out.print("Elija una opcion:\n");
       String[] opciones = {"Buscar una pagina web",
                            "Añadir nuevo elemento (web o palabra clave)",
                            "Obterner la lista de webs ordenada",
                            "Devolver las paginas web enlazadas desde una web dada",
                            "Guardar la lista de webs en ficheros",
                            "Cargar datos desde un fichero (se añadira ordenado)",
                            "Salir"};

       for(int i=0; i<7; i++)
       {
           System.out.print(i+1);
           System.out.println("- " + opciones[i]);
       }
   }
```

```java
public void seleccionarMenu() throws FileNotFoundException{
    boolean salir = false;
    while (!salir){
        String opcionElegida = miMenu.leer();
        String[] a = new String[] {"1","b","buscar","buscar pagina","buscar pagina web"};
        String[] b = new String[] {"2","a","añadir","anadir","cargar nuevo elemento","añadir nuevo elemento"};
        String[] c = new String[] {"3","o","Obtener","webs","lista ordenada","obtener la lista de webs ordenada"};
        String[] d = new String[] {"4","d","Devolver","devolver las paginas","lista webs","Devolver las paginas web enlazadas desde una web dada"};
        String[] e = new String[] {"5","g","guardar","guardar lista","guardar la lista webs","guardar fichero","save"};
        String[] f = new String[] {"6","c","cargar","load","cargar datos","cargardatos"};
        String[] g = new String[] {"7","q","salir","exit","quit","escape","leave"};

        //MODIFICAR
        if(miMenu.comparar(a, opcionElegida)){
            miMenu.buscarPagWeb();
        }
        else if(miMenu.comparar(b, opcionElegida)){
            miMenu.addElemento();
        }
        else if(miMenu.comparar(c, opcionElegida)){
            miMenu.obtenerWebs();
        }
        else if(miMenu.comparar(d, opcionElegida)){
            miMenu.pagWebReferenciadas();
        }
        else if(miMenu.comparar(e, opcionElegida)){
            miMenu.guardarWebsEnFichero();
        }
        else if(miMenu.comparar(f, opcionElegida)){
            miMenu.cargarDeFichero(false);
        }
        else if(miMenu.comparar(g, opcionElegida)){
            miMenu.salirApp();
            salir = true;
        }
        else{
            System.out.println("La opción escogida no es valida, comprueba que este bien escrito o se listo y utiliza los numeros\n");
        }
        if(!salir){
            System.out.println();
            imprimirMenu();
        }
    }
```

```
private void cargarDeFichero(boolean cargarTodo) throws FileNotFoundException {
	boolean cIndex=false, cLinks=false, cWords=false;
	boolean ok = false;

	System.out.print("¿Que desea cargar? ");

	if(!cargarTodo){
		System.out.print("(Elija una opcion:)\n");
		String[] opciones = {"Las paginas webs(Indice)",
							"Los enlaces entre Webs(Links)",
							"Base de datos nueva (Nuevas palabras clave
con las que buscar)",
							"Aceptar"};

		for(int i=0; i<4; i++){
			System.out.print(i+1);
			System.out.println("- " + opciones[i]);
		}
	}

	while (!ok && !cargarTodo){
		String opcionElegida = miMenu.leer();
		String[] a = new String[] {"1","indice","index","page","pagina","pagina
web","paginaweb","pag","webpage", "web"};
		String[] b = new String[] {"2","enlaces","link","links","en clave","word","key"};
		String[] c = new String[] {"3","base","datos","base de
datos","pal","palabra","clave","palabra clave","word","key"};
		String[] d = new String[] {"4","ok","confirmar","aceptar","cargar"," ",""};
		if(miMenu.comparar(a, opcionElegida)){
			if(!cIndex)	cIndex = true;
			else		cIndex = false;
		}
		else if(miMenu.comparar(b, opcionElegida)){
			if(!cLinks)	cLinks = true;
			else		cLinks = false;
		}
		else if(miMenu.comparar(c, opcionElegida)){
			if(!cWords)	cWords = true;
			else		cWords = false;
		}
		else if(miMenu.comparar(d, opcionElegida)){
			ok = true;
		}
		else{
			System.out.println("La opción escogida no es valida, comprueba que este
bien escrito o se listo y utiliza los numeros\n");
		}
		if(!ok){
			System.out.print("Se cargaran los archivos seleccionados: ");
			if(cIndex){
				System.out.print(" Indice ");
			}
			if(cLinks){
				System.out.print(" Enlaces ");
			}
			if(cWords){
				System.out.print(" Diccionario ");
			}
			System.out.println("");
		}
	}

	if (cargarTodo||cIndex||cLinks||cWords){

		String index=null, links=null, words=null;
		if(cargarTodo||cIndex){
			System.out.println("NOTA:  Escribir el nombre completo con la extension
(Ej:'.txt')");
```

```
                        System.out.print("¿Nombre del fichero de las paginas con su indice? (por
defecto 'index')");
                        index = miMenu.leer();
                        if (index == null || index.isEmpty()){
                                index = "index";
                        }
                        System.out.println("El nombre es: "+ index+"\n");
                }
                if(cargarTodo||cLinks){
                        System.out.print("¿Nombre del fichero de las paginas con sus enlaces? (por
defecto 'pld-arc')");
                        links = miMenu.leer();
                        if (links == null || links.isEmpty()){
                                links = "pld-arc";
                        }
                        System.out.println("El nombre es: "+ links+"\n");
                }
                if(cargarTodo||cWords){
                        System.out.print("¿Nombre del fichero con las palabras del diccionario?
(por defecto 'words.txt')");
                                words = miMenu.leer();
                        if (words == null || words.isEmpty()){
                                words = "words.txt";
                        }
                        System.out.println("El nombre es: "+ words+"\n");

                        System.out.println("Cargando archivos...");
                }



                if(cargarTodo||cWords){
                        BaseDatos.getBaseDatos().reset();
                        StopWatch timer = new StopWatch();
                        //Escanear palabras del diccionario para crear nuestra base de datos
                        FileInputStream dicc = new FileInputStream("./"+words);
                        scan = new Scanner(dicc);

                        while (scan.hasNextLine())
                        {
                                String linea = scan.nextLine();
                                String[] sp = linea.split(" +");

                                String word = sp[0];
                                BaseDatos.getBaseDatos().anadirPorLetra(word);

                        }
                        scan.close();
                        System.out.println("Ha tardado: "+timer.elapsedTime()+"s en cargar la base
de datos(palabras del diccionario).");
                }


                if(cargarTodo||cIndex){
                        Webs.getWebs().reset();
                        StopWatch timer = new StopWatch();
                        int indice = 0;
                        //Mira las webs y separa las palabras y se crea el arbol binario que
contiene todas las webs
                        FileInputStream webs = new FileInputStream("./"+index);
                        ArrayList<Integer> paginas = new ArrayList<>();
                        PagWeb pag = new PagWeb(0, null, paginas);
                        scan = new Scanner(webs);
                        while (scan.hasNextLine())//Hasta que no haya mas lineas
                        {

                                String linea = scan.nextLine();             //Leer por linea
                                String[] sp = linea.split(" +");    //Separar por /
```

```
                        String nombre = sp[0];
                        indice = Integer.parseInt(sp[1]);
                        pag = new PagWeb(indice, nombre, paginas);
                        Webs.getWebs().addWeb(pag);
                        BaseDatos.getBaseDatos().añadirWeb(nombre);

                }
                scan.close();
                Webs.getWebs().setLastID(indice);
                System.out.println("Ha tardado "+timer.elapsedTime()+"s en cargar las webs
ordenadas por Nombre (arbol binario con las paginas).");
                StopWatch timer1 = new StopWatch();
                Webs.getWebs().crearWebsID(Webs.getWebs().raiz);
                System.out.println("Ha tardado: "+timer1.elapsedTime()+"s en cargar las
webs ordenadas por ID (arbol binario con las paginas).");
                System.out.println("Ha tardado: "+timer.elapsedTime()+"s en cargar total
para cargar las webs");

        }

        if(cargarTodo||cLinks){
                //Escanear paginas que referencia
                StopWatch timer = new StopWatch();
                FileInputStream link = new FileInputStream("./"+links);
                scan = new Scanner(link);
                while (scan.hasNextLine())
                {
                        String linea = scan.nextLine();
                        String[] sp = linea.split(" +");

                        int indice = Integer.parseInt(sp[0]);
                        int idPagina = Integer.parseInt(sp[1]);

Webs.getWebs().encontrarWebPorID(indice).addElementListaRef(idPagina);

                }
                scan.close();
                System.out.println("Ha tardado: "+timer.elapsedTime()+"s en cargar la lista
de paginas que referencia una web (Links).");
        }
    }
    System.out.println("Operacion completada\n");
}

private void guardarWebsEnFichero() {
    System.out.println("NOTA:  Escribir el nombre completo con la extension (Ej:'.txt')");
    System.out.print("¿Nombre del fichero de las paginas con su indice? (por defecto
'listaWebs.txt')");
    String webs = miMenu.leer();
    if (webs == null || webs.isEmpty()){
        webs = "listaWebs.txt";

        StopWatch timer = new StopWatch();
        //Mirar SI HAY fichero, si no hay fichero crea uno vacio sino NO HARA NADA
        File websFich = new File("./"+webs);
        try {
                websFich.createNewFile();
                PrintWriter w = null;
                Webs.getWebs().guardar(Webs.getWebs().raiz, w, webs);
                System.out.println("Exito, se han guardado los datos en el archivo ../" +
webs);
                System.out.println("Ha tardado: "+timer.elapsedTime()+"s en guardar la
lista de paginas.");

        } catch (IOException e) {
                System.out.println("Ha habido algun error para crear el archivo de texto
para guardar las webs");
                e.printStackTrace();
```

```
                }

        }
}

private ArrayList<PagWeb> pagWebReferenciadas() {
        System.out.println("Introduzca el nombre o el ID de la pagina web que quiere saber que
paginas tiene enlazadas: ");
        String strings = miMenu.leer();
        PagWeb pagina = null;
        ArrayList<PagWeb> listaRef = new ArrayList<>();
        listaRef = null;
        StopWatch timer = new StopWatch();
        if (strings != null && !strings.isEmpty()){
                String[] sp = strings.split(" +");
                boolean esInt = true;

                for(int i=0; i<strings.length();i++)
                        if (!Character.isDigit(strings.charAt(i))){
                                esInt = false;
                        }
        //Buscar pagina por id(int)
                if(esInt){
                        try{
                                pagina = Webs.websID.encontrarWebPorID(Integer.parseInt(sp[0]));
                                listaRef = Webs.getWebs().devolverWebsReferenciadas(pagina);
                                System.out.println("Devuelve las paginas web enlazadas a -> "+
Webs.websID.encontrarWebPorID(Integer.parseInt(sp[0])) +":\n  (el enunciado es un poco ambiguo no
sabiamos si devolvia el objeto pagina web \n   o imprimir una lista con el nombre pero por si
acaso)");

System.out.println(Webs.getWebs().devolverWebsReferenciadas(pagina).toString());
                        }catch(NullPointerException ex){
                                System.out.println("El ID de la web que ha introducido es incorrecta
o no existe en nuestra base de datos.");
                        }

                }
        //Buscar pagina por nombre(String)
                else{
                        try{
                                pagina = Webs.getWebs().encontrarWeb(sp[0]);
                                listaRef = Webs.getWebs().devolverWebsReferenciadas(pagina);
                                System.out.println("Devuelve las paginas web enlazadas a ->"+
Webs.getWebs().encontrarWeb(sp[0])+":\n(el enunciado es un poco ambiguo no sabiamos si devolvia
el objeto pagina web \n   o imprimir una lista asi que por si acaso)");

System.out.println(Webs.getWebs().devolverWebsReferenciadas(pagina).toString());
                        }catch(NullPointerException e){
                                System.out.println("El nombre de la web que ha introducido es
incorrecta o no existe en nuestra base de datos.");
                        }


                }
                System.out.println("Ha tardado: "+timer.elapsedTime()+"s en encontrar la pagina
web y devolver su lista de paginas referenciadas (Links).");
        }
        else{
                System.out.println("Error no se ha introducido nada. ");
                pagWebReferenciadas();

        }

        return listaRef;
}

private void obtenerWebs() {
        System.out.print("¿Quiere obtener la lista ordenada alfabeticamente(por defecto) o
```

```
numericamente por id?");
        System.out.print("Elija una opcion:\n");
        String[] opciones = {"Por NOMBRE (Alfabeticamente)",
                                        "Por ID (Numericamente)",
                                        "Volver al menu anterior"};

        for(int i=0; i<2; i++){
                System.out.print(i+1);
                System.out.println("- " + opciones[i]);
        }
        boolean atras = false;
        while (!atras){
                String opcionElegida = miMenu.leer();
                String[] a = new String[] {"1","alfa","alfabeticamente", "por nombre","nombre"};
                String[] b = new String[] {"2","por id","id","numericamente","num"};
                String[] c = new String[] {"3","back","volver","cancel","atras"," ",""};
                if(miMenu.comparar(a, opcionElegida)){
                        StopWatch timer = new StopWatch();
                        Webs.getWebs().imprimirArbol(Webs.getWebs().raiz);
                        atras = true;
                        System.out.println("Ha tardado: "+timer.elapsedTime()+"s en imprimir por
Nombre (alfabeticamente).");
                }
                else if(miMenu.comparar(b, opcionElegida)){
                        StopWatch timer = new StopWatch();
                        Webs.websID.imprimirArbol(Webs.websID.raiz);
                        atras = true;
                        System.out.println("Ha tardado: "+timer.elapsedTime()+"s en imprimir por ID
(numericamente).");
                }
                else if(miMenu.comparar(c, opcionElegida)){
                        atras = true;
                }
                else{
                        System.out.println("La opción escogida no es valida, comprueba que este
bien escrito o se listo y utiliza los numeros\n");
                }
                if(!atras){
                        System.out.println();
                }
        }

}

private void addElemento() {

        System.out.print("¿Que quiere anadir una pagina o palabra clave?");
                System.out.print("Elija una opcion:\n");
                String[] opciones = {"Una pagina web",
                                                "Una palabra clave",
                                                "Volver al menu anterior"};

                for(int i=0; i<2; i++){
                        System.out.print(i+1);
                        System.out.println("- " + opciones[i]);
                }
        boolean atras = false;
        while (!atras){
                String opcionElegida = miMenu.leer();
                String[] a = new String[] {"1","page","pagina","pagina
web","paginaweb","pag","webpage", "web"};
                String[] b = new String[] {"2","pal","palabra","clave","palabra
clave","word","key"};
                String[] c = new String[] {"3","back","volver","cancel","atras"," ",""};
                StopWatch timer = new StopWatch();
                if(miMenu.comparar(a, opcionElegida)){
                        miMenu.addPagWeb();
                        atras = true;
                        System.out.println("Ha tardado: "+timer.elapsedTime()+"s en anadir una
```

```
pagina web.");
                }
                else if(miMenu.comparar(b, opcionElegida)){
                        miMenu.addWord();
                        atras = true;
                        System.out.println("Ha tardado: "+timer.elapsedTime()+"s en anadir una
palabra al diccionario.");
                }
                else if(miMenu.comparar(c, opcionElegida)){
                        atras = true;
                }
                else{
                        System.out.println("La opción escogida no es valida, comprueba que este
bien escrito o se listo y utiliza los numeros\n");
                }
                if(!atras){
                        System.out.println();
                }
        }
}


private void addWord() {

        System.out.print("Introduzca la palabra clave que desea anadir para que el buscador la
reconozca\n");
        String word = miMenu.leer();
        BaseDatos.getBaseDatos().anadirPorLetra(word);
        System.out.println("Se a anadido la palabra clave a la base de datos(diccionario)\n");
        System.out.println("(Para que el buscador encuentre por esta nueva palabra clave tiene que
cargar el indice de las paginas web. NOTA: Si carga las webs no tendran la lista de paginas
enlazadas.)\n");
}

@SuppressWarnings("null")
private void addPagWeb() {

        ArrayList<Integer> refs = new ArrayList<Integer>();  ;
        System.out.print("¿Cual es el nombre de la nueva pagina que va a ser anadida?\n");
        String nombre = miMenu.leer();
        int indice = Webs.getWebs().getLastID()+1;

        System.out.println("¿A que paginas web esta enlazadas? (Introducir IDs, para finalizar
presionar enter)\n");

        try{
                String ins = "ww";
                while (ins != null || !ins.isEmpty()){
                        ins = miMenu.leer();
                        if(ins != null || !ins.isEmpty()){
                                refs.add(Integer.parseInt(ins));
                        }

                }
        }catch(NumberFormatException ex){ // handle your exception

                PagWeb newpag = new PagWeb(indice, nombre, refs);
                newpag.anadirListaRef(refs);
                Webs.getWebs().addWeb(newpag);
                Webs.websID.addWeb(newpag);
                BaseDatos.getBaseDatos().añadirWeb(nombre);

        }
        Webs.getWebs().setLastID(indice);
        System.out.println("Se a anadido la pagina web a la base de datos (arbol))\n");
}
```

7 Conclusiones

Durante la elaboración de este proyecto hemos aprendido:

- A utilizar la clase HashMap, así como a valorar la considerable reducción de tiempo de compilación que conlleva el poder buscar un determinado elemento en un tiempo constante
- A utilizar los árboles binarios, así como a valorar la considerable reducción de tiempo al manipular datos, aunque el tiempo de compilación sigue siendo alto al obtener una lista ordenada, ya que el árbol se crea desnivelado y eso conlleva a muchos casos recursivos mientras que si estuviera desordenado el árbol se crearia mas equilibrado en vez de crear nodos hacia la derecha
- Al poder ver los tiempos de compilación con una muestra grande de datos en un programa desarrollado por nosotros mismos, hemos comprendido la importancia de determinar el coste cada método antes de implementarlo de una forma que no podríamos haber comprendido con un código que no fuese nuestro.
- Que no debemos dejar los trabajos para tan tarde

Las principales dificultades que hemos encontrado durante la elaboración de este proyecto:

- En un principio teníamos la intención organizar el árbol binario alfabéticamente durante la introducción de los datos del fichero de texto, no obstante el hecho de que las referencias que hacía cada página web a otras páginas web emplea los identificadores de las mismas en vez del URL conllevaba un problema a la hora de buscar dicha página en el árbol binario para introducir su lista de referencias.

  Finalmente optamos por tener dos árboles binarios al mismo tiempo, uno ordenado alfabéticamente en función del URL y otro ordenado por los identificadores de las páginas web. De este modo aunque se duplica la información almacenada en memoria el tiempo de cada búsqueda era más que aceptable.

  Si bien somos conscientes de que este problema se habría solucionado utilizando una tabla hash como base de datos para las páginas web (utilizando como llave su identificador) y posteriormente utilizar un algoritmo de ordenación para devolver las webs ordenadas alfabéticamente. Consideramos que al emplear los árboles binarios ampliamos los horizontes del proyecto.
- Durante las primeras semanas del proyecto teníamos la intención de implementar una tabla hash para estructurar las palabras del diccionario de modo que la llave fuera la longitud de la palabra y el contenido un String de las palabras clave con dicha longitud. En un principio esta solución nos pareció adecuada, debido a que el método que descomponía los URL de las páginas webs en subStrings para comprobar si pertenecía o no al diccionario podría acceder sucesivamente a uno u otro campo en función de la longitud de la descomposición que estuviera llevando a cabo en cada iteración.

  No obstante, el coste de este método era considerablemente superior debido a su semejanza a una matriz de datos en vez de al propio de una tabla hash.

  Finalmente optamos por seguir los consejos del profesor y emplear como llave la propia String de la palabra clave, y como campo una lista de las páginas web que incluían dicha palabra. De este modo el coste de buscar si una palabra pertenecía al diccionario era totalmente constante reduciendo el tiempo de búsqueda de forma considerable y desprendiéndose de la clase PalabraClave que en un principio asociaba cada String de la palabra clave con la lista de webs que la empleaban.

- Al llevar a cabo el análisis de costes de los métodos, en más de una ocasión encontrábamos discrepancias entre los miembros del grupo, lo que sin duda nos llevó a una mejor comprensión de la materia descubrir los errores por nosotros mismos

- Como ya habíamos experimentado en otros trabajos en grupo, cuando repartimos el trabajo y cada uno trabaja por su cuenta, después cuesta bastante poner en común los métodos realizados dado que no suelen reflejar las consideraciones que han realizado otros miembros del grupo

Algunos posibles campo de ampliación en la elaboración del proyecto:

- Cuando el usuario introduce como parámetro de búsqueda más de una palabra separada por espacios nuestro programa devuelve la conjunción de los resultados de las webs de ambas palabras, podríamos implementar un método de intersección para devolver exclusivamente las páginas webs que empleen ambas palabras al mismo tiempo
- Buscar una forma de modificar la estructura de los árboles binarios que permita que no se duplique la información de las páginas web

Finalmente aclarar que el trabajo ha sido muy instructivo para poner en práctica los conocimientos adquiridos en clase