Universidad del País Vasco
Euskal Herriko Unibertsitatea

Ingeniería Informática de gestión y sistemas de la información
Estructura de Datos y Algoritmos
2024-25

Diseño e implementación para la obtención de caminos de Webs conectadas

Iker Fernández, Urko Horas y Eneko Rodríguez

PROFESOR:
Koldo Gojenola

CURSO: 2

GRUPO: 01

Índice

Apartado 1

Introducción

En esta práctica de la asignatura Estructuras de Datos y Algoritmos, se nos pide analizar las conexiones entre diferentes nodos de un grafo. El grafo está representado mediante un HashMap, donde cada nodo (URL) está asociado a una lista de enlaces salientes que representan sus relaciones con otros nodos.

El objetivo principal era determinar si existen conexiones entre diferentes nodos. Para ello, hemos empleado técnicas de búsqueda eficientes, utilizando las nuevas estructuras aprendidas, que permiten explorar el grafo de manera óptima.

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.

Las clases implementadas son Graph y Stopwatch. Además, hemos reutilizado las clases ListaWebs y Web de la primera práctica. Por último, utilizamos la clase Prueba para ejecutar los métodos.

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

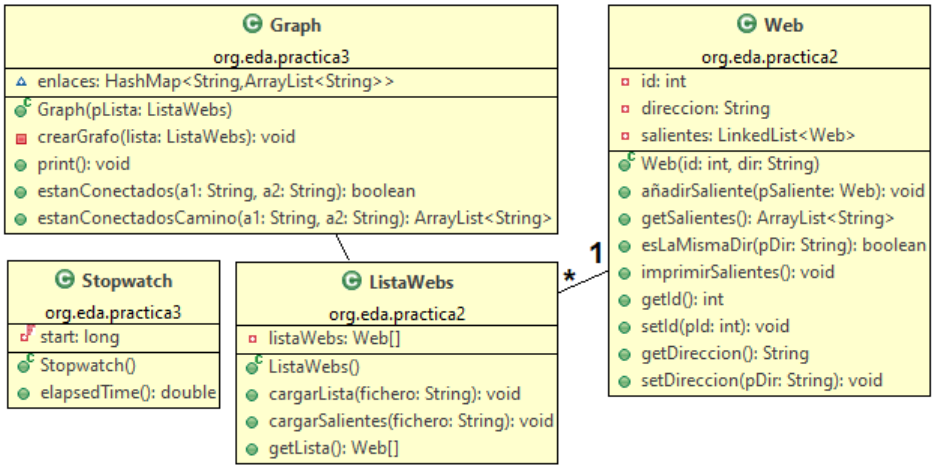

Figure 2.1: Diagrama de clases.

2.1 Clase Graph

La clase Graph representa los enlaces entre webs. Esta clase consta de 1 atributo:

- HashMap< $String, ArrayList < String >>$ enlaces: Indica los salientes que tiene cada web.

Además, cuenta con 4 métodos:

- crearGrafo(ListaWebs lista): Se utiliza para crear el grafo de enlaces entre webs, es privado porque se le llama desde la contructora.
- print(): Imprime las webs y sus salientes.
- boolean estanConectados(String a1, String a2): Devuelve un booleano que indica si la hay algun camino para ir de la web (a1) a la web (a2).
- ArrayList< $String$ > estanConectadosCamino(String a1, String a2): Devuelve el camino que va de la web (a1) a la web (a2).

2.2 Clase Stopwatch

La clase Stopwatch es un temporizador que cronometra el tiempo transcurrido desde que se crea. Esta clase consta de 3 atributos:

- Stopwatch(): Es la constructora, una vez se crea una variable inicia el temporizador.
- double elapsedTime(): Devuelve el tiempo transcurrido desde que se inicio el temporizador.

Apartado 3

Descripción de las estructuras de datos principales

En este proyecto, se han utilizado varias estructuras de datos. Estas estructuras son ArrayList, HashMap, y Queue. A continuación, se describen detalladamente:

3.1 ArrayList

Un ArrayList implementa una lista dinámica. Permite almacenar elementos y acceder a ellos mediante un índice. En este proyecto, se utiliza para representar listas de cadenas de texto que corresponden a las direcciones de páginas web.

3.2 HashMap

Un HashMap es una estructura que almacena elementos de par en par, donde el primer elemento de cada par es la clave y el segundo el valor. Gracias a esto se puede acceder a cualquier valor de manera eficiente. Además, tiene la ventaja de que las operaciones de inserción, búsqueda y eliminación rápidas, tienen un coste promedio de O(1).

3.3 Queue

Una cola es una estructura de datos que sigue el principio FIFO (First In, First Out). En este proyecto, se utiliza para realizar búsquedas que permiten explorar sistemáticamente las conexiones entre páginas web. Para implementar la cola se ha usado una LinkedList.

3.4 LinkedList

Una LinkedList o lista doblemente enlazada, que es la estructura que se implemento en la práctica 2, es una estructura basada en la conexión entre nodos. Cada nodo tiene un enlace al nodo anterior y siguiente.

Apartado 4

Diseño e implementación de los métodos principales

4.1 Clase Graph

4.1.1 Método crearGrafo(ListaWebs lista)

```
public void crearGrafo(ListaWebs lista){
//Precondicion:
//Postcondicion: Crea el grafo desde la lista de webs
//               Los nodos son los nombres de las paginas webs
```

Casos de prueba:

- Lista vacía
- Lista no vacía

Este es el código del algoritmo resultante:

```
Para cada web de lista {
    this.enlaces.put(w.getDireccion(),w.getSalientes());
}
```

Coste: El coste de este método es O(n), siendo n el número de webs en la lista que recibe el método.

4.1.2 Método estanConectados(String a1, String a2)

```
public boolean estanConectados(String a1, String a2) {
//Precondicion: Las webs a1 y a2, existen
//Postcondicion: Se devuelve un booleano que indica
//                si a1 y a2 estan conectados
```

Casos de prueba:

- a1 es distinto a a2.
- a1 es igual a a2.

Este es el código del algoritmo resultante:

```
Stopwatch reloj=new Stopwatch();
boolean enc = false;
Queue<String> porExaminar <- ();
HashSet<String> examinados <- ();
porExaminar.add(a1);
examinados.add(a1);
String actual=null;
Mientras no enc y porExaminar no sea vacio repetir {
    actual=porExaminar.remove();
    Si actuales igual a a2 entonces enc=true;
    Si no para cada saliente de enlaces.get(actual) {
        Si examinados no contiene a saliente) {
            examinados.add(saliente);
            porExaminar.add(saliente);
        }
    }
}
Imprime("\nSe han tardado "+String.format("%.4f",
reloj.elapsedTime())+" segundos en realizar la llamada
al metodo estanConectados");
```

Coste: El coste es O(n+e) siendo n el número de nodos del grafo que se recorren y e(n*m) el total de enlaces. m es el número medio de enlaces por cada nodo, es decir, que el coste sería n(1+m) = n + n*m = n + e.

4.1.3 Método estanConectadosCamino(String a1, String a2)

```
public ArrayList<String> estanConectadosCamino(String a1, String a2) {
//Precondicion: Las webs a1 y a2, existen
//Postcondicion: Se devuelve el camino que
//               va de a1 a a2
```

Casos de prueba:

- Grafo vacío.
- Grafo con varias webs, a1 es distinto a a2.
- Grafo con varias webs, a1 es igual a a2.

Este es el código del algoritmo resultante:

```
Stopwatch reloj=new Stopwatch()
ArrayList<String> camino <- ()
double tiempoEstanConectados=0;
Si estanConectados(a1, a2) entonces {
    tiempoEstanConectados=reloj.elapsedTime();
    HashMap<String,String> deDonde <- ()
    Queue<String> porExaminar<- ()
    boolean enc=false
    deDonde.put(a1, null)
    porExaminar.add(a1)
    String actual=null
    Mientras no enc y porExaminar no sea vacio repetir {
        actual=porExaminar.remove()
        Si actual es igual a a2 entonces enc=true
        Si no para cada saliente de enlaces.get(actual) {
                Si deDonde no contiene a saliente {
                    deDonde.put(saliente, actual)
                    porExaminar.add(saliente)
                }
        }
    }
    String aux=a2
    Mientras aux no sea null repetir {
        camino.add(0, aux)
        aux=deDonde.get(aux)
    }
}
imprime("Se han tardado "+String.format
("%.4f",reloj.elapsedTime()-tiempoEstanConectados)+" segundos
en realizar la llamada al metodo estanConectadosCamino \n");
devuelve camino;
```

Coste: El coste es O(n) siendo n el número de nodos del grafo que se recorren y e el número total de enlaces. Esto ocurre porque el coste de estanConectados es O(n+e) y la creación del ArrayList camino tiene un coste k (Tamaño del camino). Como, k $\leq$ n, se obtiene que O(n+e).

Apartado 5

Código

En este apartado se presentará el código de las clases.

5.1 Clase Graph

```
public void crearGrafo(ListaWebs lista){
        for(Web w:lista.getLista()) {
                this.enlaces.put(w.getDireccion(), w.getSalientes());
        }
}
public boolean estanConectados(String a1, String a2){
    Stopwatch reloj=new Stopwatch();
    boolean enc = false;
    Queue<String> porExaminar = new LinkedList<String>();
    HashSet<String> examinados = new HashSet<String>();
    porExaminar.add(a1);
    examinados.add(a1);
    String actual=null;
    while(!enc && !porExaminar.isEmpty()) {
        actual=porExaminar.remove();
        if(actual.equals(a2)) enc=true;
        else for(String saliente:this.enlaces.get(actual)) {
            if(!examinados.contains(saliente)) {
                examinados.add(saliente);
                porExaminar.add(saliente);
            }
        }
    }
    System.out.println("\nSe han tardado "+String.format("%.4f",
    reloj.elapsedTime())+" segundos en realizar la llamada al metodo
----estanConectados");
    return enc;
}
```

```
public ArrayList<String> estanConectadosCamino(String a1, String a2) {
    Stopwatch reloj=new Stopwatch();
    ArrayList<String> camino=new ArrayList<String>();
    double tiempoEstanConectados=0;
    if(this.estanConectados(a1, a2)) {
        tiempoEstanConectados=reloj.elapsedTime();
        HashMap<String,String> deDonde=new HashMap<String,String>();
        Queue<String> porExaminar=new LinkedList<String>();
        boolean enc=false;
        deDonde.put(a1, null);
        porExaminar.add(a1);
        String actual=null;
        while(!enc&&!porExaminar.isEmpty()) {
            actual=porExaminar.remove();
            if(actual.equals(a2)) enc=true;
            else for(String saliente:this.enlaces.get(actual)) {
                    if(!deDonde.containsKey(saliente)) {
                        deDonde.put(saliente, actual);
                        porExaminar.add(saliente);
                    }
            }
        }
        String aux=a2;
        while(aux!=null) {
            camino.add(0, aux);
            aux=deDonde.get(aux);
        }
    }
    System.out.println("Se han tardado "+String.format
    ("%.4f",reloj.elapsedTime()-tiempoEstanConectados+" segundos en
----realizar la llamada al metodo estanConectadosCamino \n");
    return camino;
}
```

5.2 Clase Stopwatch

```
public Stopwatch() {
    start = System.currentTimeMillis();
}

public double elapsedTime() {
    long now = System.currentTimeMillis();
    return (now - start)/1000.0;
}
```

5.3 Clase Prueba

```
public static void main(String[] args) {
    Stopwatch reloj=new Stopwatch();
    ListaWebs listaWebs=new ListaWebs();
    listaWebs.cargarLista("datuak-2024-2025/index-2024-25");
    listaWebs.cargarSalientes("datuak-2024-2025/pld-arcs-1-N-2024-25");
    double tiempoCarga=reloj.elapsedTime();
    System.out.println("Se han tardado "+String.format("%.4f",
    tiempoCarga)+" segundos en cargar la lista de webs y cargar los
    salientes de cada una");

    Graph grafo=new Graph(listaWebs);

    boolean estan=grafo.estanConectados("0-00.pl",
    "dziennikustaw.gov.pl");
    System.out.println("Estan 0-00.pl y dziennikustaw.gov.pl
    conectados?"+ estan);

    ArrayList<String>a=grafo.estanConectadosCamino("0
    00.pl","dziennikustaw.gov.pl");
    System.out.println("El camino recorrido es :");
    for(String str:a) System.out.print(str+" -> "+"");

    System.out.println();
    double tiempoConectados=reloj.elapsedTime()-tiempoCarga;
    System.out.println("\nSe han tardado "+String.format("%.4f",
    tiempoConectados)+" segundos en saber si estan
    conectados y en calcular el camino");
}
```

Apartado 6

Conclusiones

Las nuevas estructuras aprendidas e implementadas, son muy útiles a la hora de almacenar, manejar y trabajar con datos. En esta práctica, hemos podido hacer uso de estructuras como HashMap, HashSet (que es una derivación del HashMap) y las colas, aunque también hemos tenido en cuenta las pilas (se menciona posteriormente el porqué no se han utilizado).

Por un lado, las estructuras Hash son extremadamente útiles para buscar datos en un tiempo muy pequeño, de hecho, constante, O(1). Es por esto que en el método estanConectados, examinados es un HashSet; tras ponerlo como cola, nos dimos cuenta de que el contains pasaba de coste O(n) en las colas a coste O(1) con los HashSet, reduciendo considerablemente el tiempo.

Por otro lado, las colas son muy útiles a la hora de simular problemas de la vida real, ya que las colas son muy frecuentes en el día a día. Además, de que nos han servido para asegurarnos de que íbamos a obtener el camino más corto a la web a2, algo que las pilas no nos aseguraban; esto es gracias a que las colas insertan los elementos al final y las pilas los insertan al principio.

Una vez más, se ha probado con la práctica 1. Ha resultado muy interesante ver la eficacia de los métodos para un gran volumen de datos. Asimismo, nos ha acercado a entender como funcionan de manera interna las webs de internet, algo que suele ser invisible para los usuarios.