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 de algoritmos para la jerarquización de páginas Web según su importancia

Iker Fernández, Urko Horas y Eneko Rodríguez

PROFESOR:
Koldo Gojenola

CURSO: 2

GRUPO: 01

Índice

1 Introducción **2**

2 Diseño de las clases **3**
2.1 Clase Graph 4
2.2 Clase Stopwatch 4
2.3 Clase Prueba 5
2.3.1 Resultados obtenidos 6

3 Descripción de las estructuras de datos principales **7**
3.1 HashMap 7
3.2 ArrayList 7
3.3 HashSet 8
3.4 Queue 8
3.5 Array 8

4 Diseño e implementación de los métodos principales **9**
4.1 Clase Graph 9
4.1.1 Método calcularRandomWalkRank() 9
4.1.2 Método calcularPageRank() 11

5 Código **12**
5.1 Clase Graph 12
5.2 Clase Stopwatch 17
5.3 Clase Prueba 17

6 Conclusiones **19**

Apartado 1

Introducción

En esta práctica, hemos trabajado con algoritmos aplicados a grafos para analizar la importancia de sus nodos, algo fundamental en redes como páginas web o redes sociales. Para ello, hemos implementado los métodos Random Walk Rank y Page Rank, que permiten medir la relevancia de cada nodo. El objetivo ha sido aplicar estos algoritmos para comprender cómo se organiza y prioriza la información en este tipo de estructuras.

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.

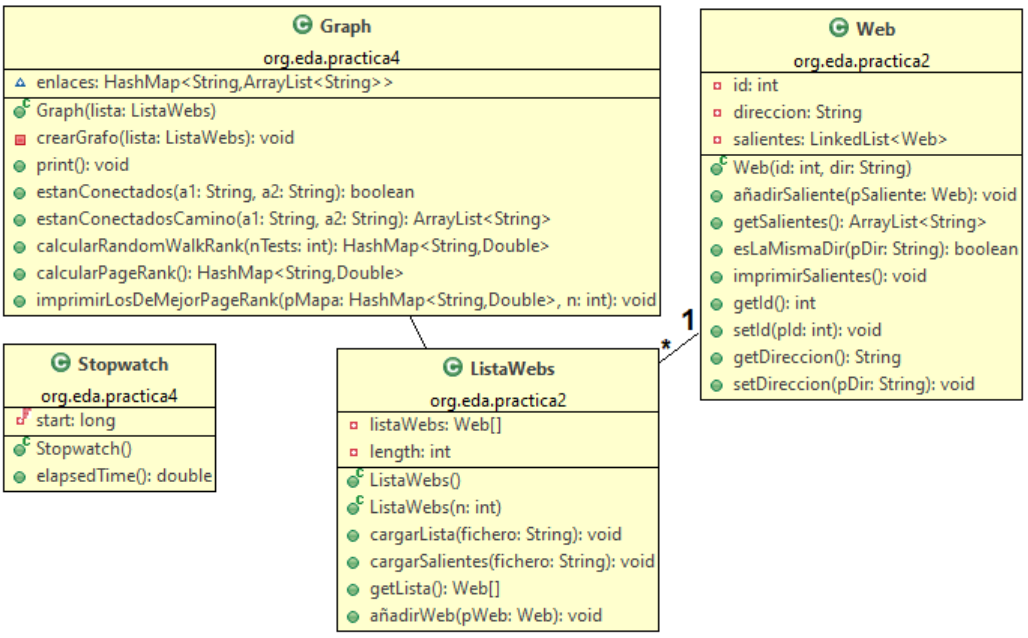

Figure 2.1: Diagrama de clases.

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).
- HashMap< $String, Double$ > calcularRandomWalkRank(int nTests): Ejecutado desde una ListaWebs, calcula mediante el algoritmo Random Walk Rank cuáles son las Webs más importantes. Devuelve un HashMap con el valor de importancia entre 0 y 1 de cada web.
- HashMap< $String, Double$ > calcularPageRank(): Ejecutado desde una ListaWebs, calcula mediante el algoritmo Page Rank cuáles son las Webs más importantes. Devuelve un HashMap con el valor de importancia entre 0 y 1 de cada web.
- imprimirLosDeMejorPageRank(HashMap< $String, Double$ > pMapa, int n): Imprime las n Webs jerárquicamente más importantes junto con su valor de importancia. (No funcionaba correctamente el método propocionado en el enunciado, por lo que buscamos una solución; además, es más eficiente)

2.2 Clase Stopwatch

La clase Stopwatch es un temporizador que cronometra el tiempo transcurrido desde que se crea. Mide el tiempo en nanosegundos, para tener mayor precisión. Esta clase consta de 1 atributo:

- long start: Indica el tiempo cuando que ha creado un Stopwatch.

Además, cuenta con 2 métodos:

- 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.

2.3 Clase Prueba

En la clase Prueba se realizan los tests a los métodos implementados. Los métodos randomPageRank y pageRank se han probado de dos maneras, con la lista de webs que se está utilizando en todas las prácticas y, además, con el grafo que se da como ejemplo en la Wikipedia representado en la figura 2.2.

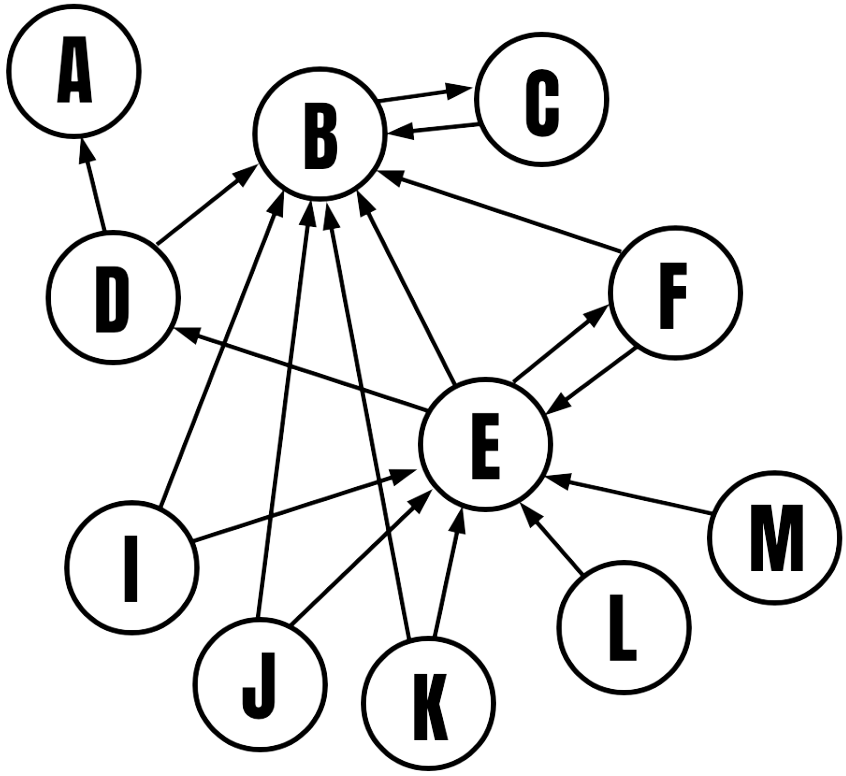

Figure 2.2: Grafo ejemplo Wikipedia.

2.3.1 Resultados obtenidos

Los resultados obtenidos y, por tanto, las Webs más importantes para el grafo ejemplo han sido las siguientes:

Resultados por algoritmo y orden de importancia			
Nodo	**Valor importancia aprox.**	**Nodo**	**Valor importancia aprox.**
B	0.2253 = 22.53%	B	0.3242 = 32.42%
C	0.2016 = 20.16%	C	0.2892 = 28.92%
E	0.1606 = 16.06%	E	0.0682 = 6.82%
D	0.0826 = 8.26%	F	0.0330 = 3.30%
F	0.0758 = 7.58%	D	0.0330 = 3.30%
A	0.0708 = 7.08%	A	0.0276 = 2.76%
M	0.0392 = 3.92%	I	0.0136 = 1.36%
L	0.0367 = 3.67%	J	0.0136 = 1.36%
I	0.0361 = 3.61%	K	0.0136 = 1.36%
J	0.0361 = 3.61%	L	0.0136 = 1.36%
K	0.0351 = 3.51%	M	0.0136 = 1.36%

Table 2.1: Resultados del ejemplo

Los resultados obtenidos para la lista de webs y, por tanto, las Webs más importantes (sólo se representarán las 10 primeras webs) son:

Resultados por algoritmo y orden de importancia			
Web	**Valor aprox.**	**Web**	**Valor aprox.**
www.info.at	$1.6260 \cdot 10^{-3}$	www.info.at	$1.1050 \cdot 10^{-3}$
modssl.org	$8.7145 \cdot 10^{-4}$	purl.org	$5.6512 \cdot 10^{-4}$
allaboutcookies.org	$7.5125 \cdot 10^{-4}$	modssl.org	$4.6117 \cdot 10^{-4}$
purl.org	$6.6110 \cdot 10^{-4}$	winzip.com	$3.0870 \cdot 10^{-4}$
macromedia.com	$5.1085 \cdot 10^{-4}$	steuerberater.at	$2.8787 \cdot 10^{-4}$
simplemachines.org	$4.8080 \cdot 10^{-4}$	macromedia.com	$2.8202 \cdot 10^{-4}$
winzip.com	$4.8080 \cdot 10^{-4}$	mixi.jp	$2.5348 \cdot 10^{-4}$
rechtsanwalt.at	$4.8080 \cdot 10^{-4}$	cybersitter.com	$2.5258 \cdot 10^{-4}$
section508.gov	$4.5075 \cdot 10^{-4}$	rechtsanwalt.at	$2.5045 \cdot 10^{-4}$
mapy.cz	$4.2070 \cdot 10^{-4}$	allaboutcookies.org	$2.4564 \cdot 10^{-4}$

Table 2.2: Resultados de la lista de webs

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 HashMap

La estructura HashMap permite almacenar pares clave-valor y ofrece acceso eficiente promedio en tiempo constante a los valores mediante sus claves. En el proyecto, se utiliza para diversas aplicaciones: representar enlaces web mediante un HashMap¡String, ArrayList¡String¿¿ llamado enlaces, donde las claves son direcciones de páginas web y los valores son listas de páginas enlazadas, modelando un grafo dirigido; rastrear rutas en el método estanConectadosCamino con un HashMap¡String, String¿ que guarda el nodo previo de cada nodo explorado; y calcular rankings en los métodos calcularRandomWalkRank y calcularPageRank, utilizando un HashMap¡String, Double¿ que almacena el ranking de cada página.

3.2 ArrayList

La clase ArrayList implementa una lista dinámica que permite almacenar elementos del mismo tipo y acceder a ellos mediante índices. En el proyecto, se usa para manejar los enlaces salientes desde una página web, representados en los valores del HashMap¡String, ArrayList¡String¿¿ como ArrayList¡String¿, y para devolver caminos en el grafo, como en el método estanConectadosCamino, que genera un ArrayList¡String¿ con el camino entre dos nodos si existe. Esta estructura facilita el acceso y manipulación secuencial de elementos, además de permitir el crecimiento dinámico del tamaño según sea necesario.

3.3 HashSet

La clase HashSet proporciona una colección que no permite duplicados y ofrece acceso eficiente promedio en tiempo constante. En este proyecto, se utiliza para el seguimiento de nodos visitados en los métodos estanConectados y calcularRandomWalkRank, donde un HashSet¡String¿ almacena los nodos ya explorados durante la búsqueda en el grafo, evitando procesar nodos repetidos y ciclos. Entre sus ventajas, se encuentra la verificación rápida de la existencia de elementos y la prevención de duplicados, características cruciales para rastrear nodos ya procesados.

3.4 Queue

La interfaz Queue y su implementación LinkedList permiten manejar colecciones con una estructura de cola FIFO (First In, First Out). Estas estructuras son fundamentales para la exploración BFS en los métodos estanConectados y estanConectadosCamino, donde se utiliza una Queue¡String¿ para procesar nodos en el orden en que se encuentran. Esto facilita la exploración nivel por nivel en grafos de manera eficiente.

3.5 Array

Aunque no son la estructura principal del proyecto, los arrays se utilizan en casos específicos para simplificar operaciones. Por ejemplo, en el método calcularRandomWalkRank, las claves del HashMap se convierten en un array para seleccionar nodos de forma aleatoria. Los arrays permiten acceso rápido a elementos mediante índices y son útiles en operaciones de selección puntual o transformación de colecciones.

Apartado 4

Diseño e implementación de los métodos principales

4.1 Clase Graph

4.1.1 Método calcularRandomWalkRank()

```
public HashMap<String, Double> calcularRandomWalkRank(int nTests) {
//Precondicion: -
//Postcondicion: Devuelve un HashMap que asigna a cada pagina un valor
//               numerico representando su ranking.
```

Casos de prueba:

- Grafo reducido (Figura 2.2).
- Grafo grande (Lista de Webs).

Este es el código del algoritmo resultante:

```
d = 0.85 // factor de damping
Random r <- ()
String[] webs <- this.enlaces.keySet().toArray(new String[0])
HashMap<String,Double> resultado <- ()
Para cada web de webs repetir {
    resultado.put(web, 0)
}
numElem = 0
Para cada i desde 0 hasta nTests repetir {
    HashSet<String> examinados <- ()
    x = r.nextInt(webs.length)
    web = webs[x]
    a adir web a examinados
    parar = false
    numElem++
```

```
    Mientras parar sea false repetir {
        y = r.nextDouble()
        Si y<=0.85 entonces {
            Si enlaces.get(web) est   vac a entonces{
                x = r.nextInt(this.enlaces.get(web).size())
                web = this.enlaces.get(web).get(x)
                Si examinados contiene web parar=true
                Si no {
                    numElem++
                    anadir web a examinados
                }
            }
        } Sino parar=true
    }
    Para cada w en examinados repetir resultado.put(w, resultado.get(w)+1.0)
}
para cada w en resultado.keySet() repetir{
    resultado.put(w, resultado.get(w)/numElem)
}
devolver resultado
```

Coste: El coste del método depende de los pasos de simulación (M) y del número de nodos (N) en el grafo. En cada iteración, se selecciona aleatoriamente un nodo y se sigue uno de sus enlaces, lo cual implica operaciones de búsqueda y acceso en el grafo. Estas operaciones son en promedio O(1) con `HashMap`. Sin embargo, recorrer todas las iteraciones lleva un coste total de O(M). Si se considera también el coste de inicialización y cálculo del ranking final para todos los nodos, el coste global podría aproximarse a O(N + M).

4.1.2 Método calcularPageRank()

```
public HashMap<String, Double> calcularPageRank() {
//Precondicion: -
//Postcondicion: Devuelve un HashMap con el valor de
//               importancia correspondiente a cada Web.
```

Casos de prueba:

- Grafo reducido (Figura 2.2).
- Grafo grande (Lista de Webs).

Este es el código del algoritmo resultante:

```
d = 0.85 // factor de damping
valorDamping = (1-d)/N
HashMap<String,Double> iteracionAnterior <- ()
HashMap<String,Double> resultado <- ()
Para cada web de enlaces repetir {
    iteracionAnterior.put(web, 1/N)
    resultado.put(web, 0)
}
sumatorioIteracion = 1.0
Mientras sumatorioIteracion > 0.0001 repetir {
    sumatorioIteracion=0.0
    Para cada web de enlaces repetir {
        contrib.=Valor importancia anterior/N. salientes
        Para cada saliente de la web repetir
            resultado.put(saliente,resultado.get(saliente)+contrib.)
    }
    Para cada web de resultado {
        resultado.put(w, (1-d)/N+d*resultado.get(w))
        sumatorioIteracion+=Valor absoluto de
        iteracionAnterior.get(w)-resultado.get(w))
    }
    Si el sumatorioIteracion>0.0001 entonces
        Para cada web de resultado repetir {
            iteracionAnterior toma los valores de resultado
            resultado <- (mantiene la web,0)
        }
}
devolver resultado
```

Coste: El primer for tiene coste O(N), siendo N el número de webs. El segundo for es un for anidado, en el que el primero tiene coste O(N) y el segundo coste O(M), siendo M el número medio de webs salientes por web, por lo que, el for anidado tiene coste O(N*M). Nuevamente, el tercer y cuarto for, tienen coste N. El while, en cambio, tiene coste I, siendo I el número de iteraciones necesarias para alcanzar ese valor. Por lo que, el coste resultante es O(I*N*M).

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;
}
```

```
public HashMap<String, Double> calcularRandomWalkRank(int nTests) {

Stopwatch reloj=new Stopwatch();

double d = 0.85; // damping factor
Random r = new Random();
String[] webs=this.enlaces.keySet().toArray(new String[0]);
HashMap<String,Double> resultado = new HashMap<String,Double>();

for (String web : webs) {
    resultado.put(web, 0.0);
}
int numElem=0;

for(int i=0;i<nTests;i++) {
    HashSet<String> examinados=new HashSet<String>();
    int x=r.nextInt(webs.length);
    String web=webs[x];
    examinados.add(web);
    boolean parar=false;
    numElem++;

    while(!parar) {
        double y=r.nextDouble();
        if(y<=0.85) {
            if(!this.enlaces.get(web).isEmpty()) {
                x=r.nextInt(this.enlaces.get(web).size());
                web=this.enlaces.get(web).get(x);
                if(examinados.contains(web)) parar=true;
                else {
                    numElem++;
                    examinados.add(web);
                }
            }
        } else parar=true;
    }
    for(String w:examinados) resultado.put(w, resultado.get(w)+1.0);
}

for(String w:resultado.keySet())resultado.put(w, resultado.get(w)/numElem);

double tiempo=reloj.elapsedTime();
System.out.println("\nEL-MTODO-calcularRandomWalkRank
HA-TARDADO-"+String.format("%.4f", tiempo)+"
SEGUNDOS-EN-EJECUTARSE");

return resultado;
}
```

```
public HashMap<String, Double> calcularPageRank() {
    Stopwatch reloj=new Stopwatch();
    double d = 0.85; // damping factor
    double valorDamping=(1-d)/this.enlaces.size(); //(1-d)/N
    HashMap<String,Double> iteracionAnterior=new HashMap<String,Double>();
    HashMap<String,Double> resultado=new HashMap<String,Double>();
    for(String web:this.enlaces.keySet()) {
        iteracionAnterior.put(web, 1.0/this.enlaces.size());
        resultado.put(web, 0.0);
    }
    double sumatorioIteracion=1.0;
    while(sumatorioIteracion>0.0001) {
        sumatorioIteracion=0.0;
        for(String web:this.enlaces.keySet()) {
            double contrib=
            =iteracionAnterior.get(web)/this.enlaces.get(web).size();
            for(String saliente:this.enlaces.get(web)) {
                resultado.put(saliente,resultado.get(saliente)+contrib);
            }
        }
        for(String w:resultado.keySet()) {
            resultado.put(w, valorDamping+d*resultado.get(w));
            sumatorioIteracion+=Math.abs(iteracionAnterior.get(w)
            -resultado.get(w));
        }
        if(sumatorioIteracion>0.0001) {
            for(String web:resultado.keySet()) {
                iteracionAnterior.put(web, resultado.get(web));
                resultado.put(web, 0.0);
            }
        }
    }
    double tiempo=reloj.elapsedTime();
    System.out.println("\nEL METODO calcularPageRank HA TARDADO
----"+String.format("%.4f", tiempo)+" SEGUNDOS EN EJECUTARSE");
    return resultado;
}
```

```
public void imprimirLosDeMejorPageRank(
HashMap<String, Double> pMapa, int n) {

    // Usamos una cola de prioridad para mantener los n
    // elementos de mayor valor
    PriorityQueue<Map.Entry<String, Double>> pq = new PriorityQueue<>(
            Comparator.comparingDouble(Map.Entry::getValue));
    for (Map.Entry<String, Double> entry : pMapa.entrySet()) {
        if (pq.size() < n) {
            pq.add(entry);
        } else if (entry.getValue() > pq.peek().getValue()) {
            pq.poll();
            pq.add(entry);
        }
    }

    // Convertimos la cola de prioridad a una lista y
    // la ordenamos en orden descendente
    List<Map.Entry<String, Double>> result = new ArrayList<>(pq);
    result.sort((e1, e2) -> Double.compare(e2.getValue(), e1.getValue()));

    // Imprimimos los n elementos de mayor valor
    int i=0;
    for (Map.Entry<String, Double> entry : result) {
        System.out.println("La web en la posici n de importancia "+
        ++i+" es: " + entry.getKey() + ", con un valor de: " +
        String.format("%.6f", entry.getValue()));
    }
}
```

5.2 Clase Stopwatch

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

public double elapsedTime() {
    long now = System.nanoTime();
    return (now - start)/1000000000.0;
}
```

5.3 Clase Prueba

```
public static void main(String[] args) {

    ListaWebs listaWebs1=new ListaWebs(11);
    Web webA=new Web(1, "A");
    Web webB=new Web(2, "B");
    Web webC=new Web(3, "C");
    Web webD=new Web(4, "D");
    Web webE=new Web(5, "E");
    Web webF=new Web(6, "F");
    Web webI=new Web(7, "I");
    Web webJ=new Web(8, "J");
    Web webK=new Web(9, "K");
    Web webL=new Web(10, "L");
    Web webM=new Web(11, "M");

    webB.anadirSaliente(webC);
    webC.anadirSaliente(webB);
    webD.anadirSaliente(webA);
    webD.anadirSaliente(webB);
    webE.anadirSaliente(webB);
    webE.anadirSaliente(webD);
    webE.anadirSaliente(webF);
    webF.anadirSaliente(webB);
    webF.anadirSaliente(webE);
    webI.anadirSaliente(webB);
    webI.anadirSaliente(webE);
    webJ.anadirSaliente(webB);
    webJ.anadirSaliente(webE);
    webK.anadirSaliente(webB);
    webK.anadirSaliente(webE);
    webL.anadirSaliente(webE);
    webM.anadirSaliente(webE);

    listaWebs1.anadirWeb(webA);
    listaWebs1.anadirWeb(webB);
```

```
        listaWebs1.anadirWeb(webC);
        listaWebs1.anadirWeb(webD);
        listaWebs1.anadirWeb(webE);
        listaWebs1.anadirWeb(webF);
        listaWebs1.anadirWeb(webI);
        listaWebs1.anadirWeb(webJ);
        listaWebs1.anadirWeb(webK);
        listaWebs1.anadirWeb(webL);
        listaWebs1.anadirWeb(webM);

        //PRUEBA CON EL GRAFO QUE APARECE DE EJEMPLO EN EL ENUNCIADO

        Graph grafo1=new Graph(listaWebs1);
        HashMap<String, Double> res1=grafo1.calcularRandomWalkRank(10000);
        System.out.println("UTILIZANDO EL ALGORITMO RANDOM
    ----WALK RANK EN EL GRAFO UTILIZADO COMO EJEMPLO EN EL
    ----ENUNCIADO, LOS VALORES OBTENIDOS SON: ");
        grafo1.imprimirLosDeMejorPageRank(res1, 20);

        HashMap<String, Double> res2=grafo1.calcularPageRank();
        System.out.println("UTILIZANDO EL ALGORITMO PAGE
    ----RANK EN EL GRAFO UTILIZADO COMO EJEMPLO EN EL
    ----ENUNCIADO, LOS VALORES OBTENIDOS SON: ");
        grafo1.imprimirLosDeMejorPageRank(res2, 20);

        //PRUEBA CON LA LISTA DE WEBS

        ListaWebs listaWebs2=new ListaWebs();
        listaWebs2.cargarLista("datuak-2024-2025/index-2024-25");
        listaWebs2.cargarSalientes("datuak-2024-2025/pld-arcs-1-N-2024-25");

        Graph grafo2=new Graph(listaWebs2);

        HashMap<String, Double> res3=grafo2.calcularRandomWalkRank(10000);
        System.out.println("UTILIZANDO EL ALGORITMO
    ----RANDOM WALK RANK EN EL GRAFO DE WEBS, LOS
    ----VALORES OBTENIDOS SON: ");
        grafo2.imprimirLosDeMejorPageRank(res3, 20);

        HashMap<String, Double> res4=grafo2.calcularPageRank();
        System.out.println("UTILIZANDO EL ALGORITMO
    ----PAGE RANK EN EL GRAFO DE WEBS,
    ----LOS VALORES OBTENIDOS SON: ");
        grafo2.imprimirLosDeMejorPageRank(res4, 20);

    }
```

Apartado 6

Conclusiones

Hemos desarrollado los algoritmos solicitados, asegurándonos de que cumplen con los requisitos y generan resultados claros y coherentes. Gracias a técnicas como Random Walk Rank y PageRank, hemos podido medir de forma precisa la importancia de los nodos en un grafo. Además, los resultados han confirmado que estos métodos son útiles para organizar y priorizar información en contextos como redes o buscadores, demostrando la utilidad práctica de los conceptos trabajados.