PAGE RANK

Iñaki García, Ander Lekanda, Alex Beltrán

June 26, 2022

Contenido

Capítulo 1

Introducción

En esta tarea hemos implementado un algoritmo llamado PageRank, el cual era utilizado en Google anteriormente. Este algoritmo asigna de forma numérica la relevancia de páginas web indexados por un motor de busqueda. Este ejercicio es la continuación de las 3 tareas anteriores, por lo cual implementamos muchos de los método utilizados en esas tareas.(Ejemplo de PageRank figura 1.2).

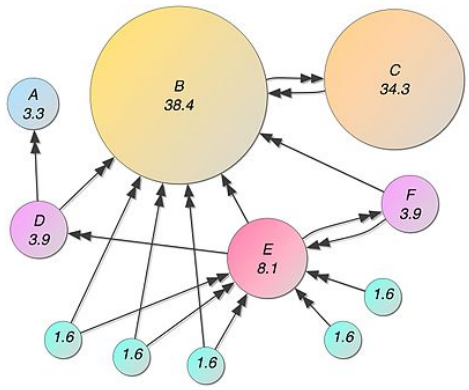

Figure 1.1: Ejemplo de algoritmo PageRank.

El cálculo de PageRank requiere varios pasos, llamados iteraciones, para ajustar el valor aproximado. Inicialmente, todos los nodos reciben la misma probabilidad (asociado a la dimensión del grafo). Se muestra el cálculo de la probabilidad de un nodo en la figura 2.1).

$$\forall\, A \in nodos\,, PR(A) = \frac{1}{N} \quad \text{donde } \mathbf{N} \text{ es el número de nodos del grafo.}$$

Figure 1.2: Calculo probabilidad de un nodo.

Como hemos mencionado antes, tenemos que iterar; por lo cual esa probabilidad varia como se muestra en la figura 1.3. La iteración se acaba cuando la suma de PRActual-PRAnterior (en valor absoluto) sea menor que 0.0001.

Explicación de la formula:

$$PR(A) = \frac{1-d}{N} + d * \sum_{i=1}^{n} \frac{PR(i)}{C(i)} \quad \text{donde:}$$

Figure 1.3: Ejemplo de algoritmo PageRank.

- **PR(A):** es el PageRank de la página A.
- **d:** (damping factor) es un factor de amortiguación que tiene un valor entre 0 y 1 (un valor típico es 0.85)
- **PR(i):** son los valores de PageRank que tienen cada una de las páginas i que enlazan a A.
- **C(i):** es el número total de enlaces salientes de la página i (sean o no hacia A).

Los objetivos de la tarea son los siguientes:

- **Tarea principal:** implementar un método que calcule el PageRank de un grafo. Ese grafo contiene páginas webs y están asociados a un identificador.
- **Tarea opcinal:** implementar un buscador que realiza la busqueada de páginas webs. Ese buscador devuelve una lista con páginas webs que coinciden con la palabra clave introducida por el usuario. Esa lista está ordenada según los PageRanks.

Capítulo 2

Diseño de las clases

En esta tarea hemos agregado una única clase nueva, Par. Esta clase es una clase auxiliar para el método buscar; el cual busca webs ordenadas mediante sus pageRank. Por otra parte, se han añadido nuevos métodos a la clase Grafo: pageRank y buscar. En la figura 2.1 está el diagrama de clases.

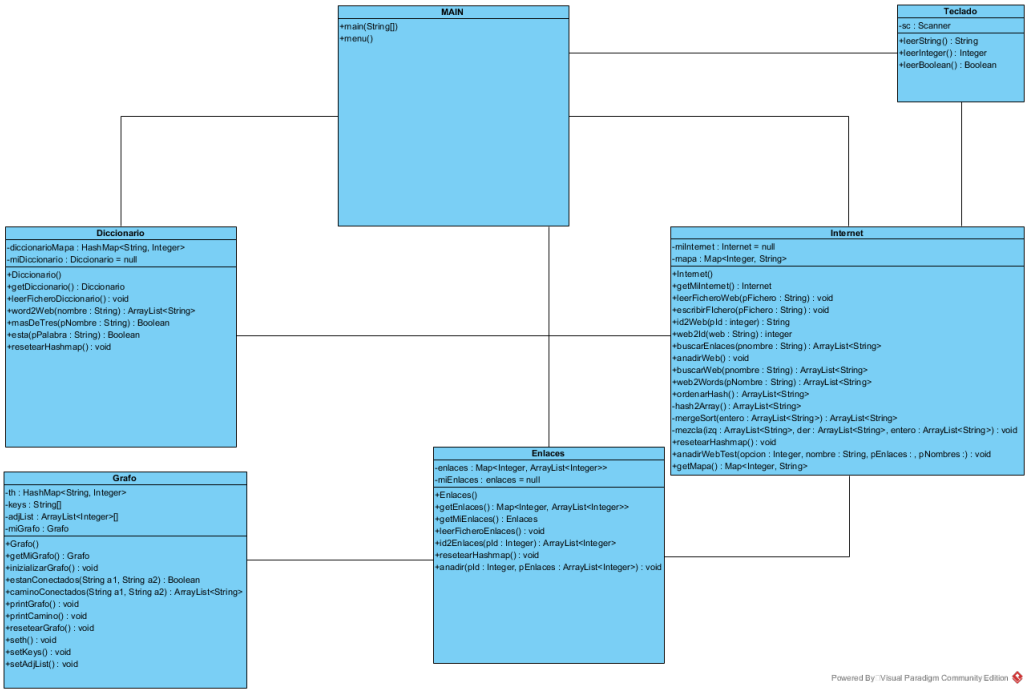

Figure 2.1: Diagrama de clases.

Capítulo 3

Descripción de las estructuras de datos principales

En esta tare hemos utilizado las siguientes estructuras de datos:

- **Array**: se utilizada un Array de Double para almacenar en cada iteración la parte de la formula PR(i)/C(i). Como unicamente se accede a este, no hemos visto conveniente utilizar otra estrucutura de datos.
- **ArrayList**: se han utilizado ArrayList para almacenar; en su mayoría, enlaces.
- **HashMap**: se utiliza el HashMap en el método pageRank() es el dato de salida. En ella se almacena el nombre de la web con su correspondiente PageRank.

Capítulo 4

Diseño e implementación de los métodos principales

En este apartado se presentarán los métodos principales, indicando por cada uno su especificación (precondiciones y postcondiciones), además de los casos de prueba planteados. Por cada método no trivial, se presentará la descripción del algoritmo implementado. Finalmente, se deberá presentar, de manera razonada, el cálculo del coste de cada algoritmo. Presentamos un ejemplo de una posible manera de presentar un método de ejemplo.

4.1 Método pageRank

```
public HashMap<String, Double> pageRank(){
// Precondicion: las estructuras de datos de la clase
    estan inicializadas.
// Postcondicion: devuelve un HashMap con las webs y sus
    respectivas pageRank.
```

Casos de prueba:

Las pruebas del método pageRank se efectuarán con ficheros de pruebas para poder agilizar el proceso, dado que realizarlas con los ficheros originales resultaría casi imposible.

Para probar el método se han definido 3 casos:

- **Fichero normal de 10 elementos, con caminos posibles e imposibles.**

  Como la automatización de los casos de pruebas del método pageRank resulta casi imposible, hemos hecho los cálculos a mano, para luego compararlos con lo que el resultado debería ser. Si son iguales o sólo los diferencia unos pocos decimales, lo damos por bueno (dado que hay que contar con el error humano). Por comodidad, primero se imprime la estructura de datos del pageRank entera, para después analizar las 11 webs del fichero una a una.

- **Fichero vacío**.

Para el caso del fichero vacío, simplemente nos aseguramos con un assertEquals de que el tamaño de la estructura de datos donde guardamos el hashmap sea 0, que nos confirma que no hay pageRanks, porque no hay webs.

- **Fichero de 1 elemento**.

  Esta última prueba se hace de forma idéntica a la primera, sólo que en este caso, solamente se calcula para el primer elemento. No es necesario hacer un assertEquals para el tamaño de la estructura de datos, porque la imprimimos antes, y podemos ver que sólo contiene un elemento.

Este es el código del algoritmo resultante:

```
%    //Inicializamos las variables
     mientras !acabar
         Iterator itr = pageRank.iterator() //Recorremos el
             HashMap de pageRank
         mientras itr.hashNext
             entrada = itr.sig().
             id = th.get(entra.getClave())
             enlaces = Enlaces.convertirIdaENlaces(id)
             si enlaces==null || enlaces.tamano==0 entonces
                 puntuacion.anadir[id] = 0.0
             si no
                 puntuacion.anadi[id] =
                     entrada.getValor/enlaces.tamano
                     //PR(i)/C(i)

         //Recorremos otra vez el HashMap, pero antes tenemos
             que inicializar
         //otra vez el iterador
         mientras itr.hasNext
             entrada2 = itr.sig()
             id = th.get(entrada.getClave())
             enlacesEntrantes =
                 Enlaces.obtenerEnlacesReferenciados(id)
             si enlacesEntrantes!=null entonces
                 para i=0; i<enlacesEntrantes.tamano(); i++
                     suma = suma +
                         (puntuacion[enlacesEntrantes.get(i)]
             si no
                 suma = 0.0
             //aplicamos la formula
             PR = ((1-0.85)/pageRank.tamano())+(0.85*suma))
             diferenciaActual =
                 PR-pageRank.get(entrada2.getClave)
             si diferenciaActual<0 entonces
                 //Aplicamos el valor absoluto
                 diferenciaActual = diferenciaActual*(-1)
             diferencia = diferencia + diferenciaActual
             pageRank.anadir(entrada2.getValor(),PR)
             suma = 0.0

         si diferencia <0.0001 entonces
```

```
            acabar = True
        si no
            diferencia = 0.0

    return pageRank
```

Coste: en este algoritmo utilizamos dos bucles. Por una parte, se recorre un HashMap con un iterador; es decir, entero. En el bucle utilizamos el método id2Enlaces(i) que es constante ya que accede al *HashMap* de Enlaces; además, se hace un acceso a un Array que también es constante. Por lo cual, el coste de este bucle es: O(n+1+1) (siendo "n" los elementos del HashMap de pageRank).

Por otra parte, tenemos otro bucle que también se recorre el *HashMap* de pageRank. Se recorre dos veces esta estructura de datos ya que en el primer bucle calculamos los dividendos de los nodos y en el segundo se hace un sumatorio por cada nodo, lo cual no sería posible sin el primer bucle. Además, con el método referenciados() recorremos todos los enlaces del HashMap de Enlaces y hacemos el acceso a un Array. Por consiguiente, el coste de este algoritmo es el siguiente: O(m*(s+1)) (siendo "m" los elementos del *HashMap* de pageRank y siendo "s" los elementos del HashMap de Enlaces).

Dicho todo esto, podemos decir que el coste de este método es el siguiente: O(n*(1+1)+m*(s+1)) por lo cual sería **cuadrático**.

4.2 Método buscar

```
public ArrayList<Par> buscar(String palabraClave) {
// Precondicion: las estructuras de datos de la clase
    estan inicializadas.
// Postcondicion: devuelve un ArrayList de Par ordenado de
//  manera ascendente
```

Casos de prueba: Las pruebas del método buscar se efectuarán con ficheros de pruebas para poder agilizar el proceso, dado que realizarlas con los ficheros originales resultaría casi imposible.

Para probar el método se han definido 2 casos:

- **Fichero normal de 10 elementos y un solo Par en el ArrayList.**

  Como la automatización de los casos de pruebas requieren los pageRank los hemos calculado anteriormente. Lo que hacemos es crear dos ArrayList uno auxiliar con los datos que nosotros sabemos que son correctos y otro con los datos que calcula el sistema. Posteriormente calculamos su dimensión. Si es diferente salta el fail, ya que no tienen la misma dimensión y por lo tanto no es el mismo. Después mediante el método del iterado comprobamos uno a uno los pares que en esta caso solo es uno. Si nada sale mal el test estará acabado, y si no saltará una excepción.

- **Fichero normal de 10 elementos y mas de un Par en el ArrayList.**

No hace falta que volvamos a explicar todo el recorrido ya que es el mismo que el del apartado anterior. Lo único que cambia es que el while dará más de una vuelta.

Este es el código del algoritmo resultante:

```
%    //Inicializamos las variables
     ArrayList<Par> resultado = new ArrayList<>()
     pageRank() //Iniciamos el metodo para calcular los
         pageRank
     Iterator itr = pageRank.iterator()
     mientras itr.hasNext()
         entrada = itr.sig()
         si entrada.getValor().contiene(palabraClave) entonces
             resultado.anadir(nuevo Par(entrada.getClave(),
                 entrada.getValor()))

     Collections.ordenar(resultado, Par.ComparadorPorPageRank)
     //Metodo explicado despues
```

Hacemos uso de **Collections** ya que es una clase que hemos visto en clase. Además, tras consultarlo en la *API de Java*, hemos comprobado que el método .sort() ordena una lista de como lo hace el mergesort; método que está implementando en la tarea 1.

La ordenación por mezcla funciona dividiendo la lista sin clasificar en N listas secundarias, donde N es el número de elementos en la lista. Esto nos da N listas y cada una de ellas solo contiene un elemento. Un elemento siempre está ordenado, por lo tanto, tenemos N listas ordenadas. Ahora, con las N listas ordenadas, las mezclamos repetida

Coste: en este algoritmo se recorre el HashMap de pageRank con un iterador; por lo cual, se recorren todos sus elementos. De seguido, se ordena el ArrayList que se genera en el bucle anterior con el método mergesort, que su coste es O(n.log2 n)

Capítulo 5

Código

En este apartado aparece la implementación de la clase nueva Par y de los nuevos métodos de la clase Grafo. No aparecen muchas clases presentadas en este documento ya que se han explicado en los documentos previos. En este apartado también se explican las pruebas utilizadas.

Para las pruebas, se usarán los ficheros creados por nosotros, con menos elementos que los ficheros originales, con tal de conseguir la mayor automatización posible. Se podrá acceder a una prueba completa con los ficheros originales desde el menu inicial del programa. Estos ficheros mas pequeños se usan para comprobar el buen fucionamiento de los métodos mediante JUnits Test Cases. Junto a la implementación de las pruebas, se proporciona una extensión de las pruebas presentadas en el Capítulo 4.

Todos los métodos no explicados aqui se consideran triviales para las pruebas,o han sido probados anterioromente a lo largo del proyecto global. Dicho de otra forma, sólo lo que tenga que ver con el laboratorio actual sera probado (siempre y cuando no sea trivial, como puese ser un print(x)).

5.1 Clase Grafo

```
public class Grafo {

private HashMap<String, Integer> th = new HashMap<String,
    Integer>();
private String[] keys;
private ArrayList<Integer>[] adjList;
private static Grafo miGrafo = new Grafo();
private HashMap<String, Double> pageRank = new HashMap<>();

public Grafo() {
}
```

```
public static Grafo getMiGrafo() {
    return miGrafo;
}

public void inicializarGrafo() {
    //Iniciliazamos las estructuras de datos.

    Enlaces enlaces = Enlaces.getMiEnlaces();
    adjList = new ArrayList[enlaces.tamano()];
    for(int i=0; i<adjList.length; i++) {
        adjList[i] = new ArrayList<Integer>();
        adjList[i] = enlaces.id2Enlaces(i);
    }

    Internet mapaWeb = Internet.getMiInternet();
    Iterator<Map.Entry<Integer, String>> itr =
       mapaWeb.iterator();
    String web = null;
    int id = 0;
    while(itr.hasNext()) {
        Map.Entry<Integer, String> entrada = itr.next();
        web = entrada.getValue();
        id = entrada.getKey();
        th.put(web, id);

   }

    keys = new String[th.size()];
    for(String k: th.keySet()) keys[th.get(k)] = k;
}

public boolean estanConectados(String a1, String a2) {
    Queue<Integer> porExaminar = new LinkedList<Integer>();
    boolean enc = false;
    if (th==null || keys==null || adjList == null){
        System.out.println("Grafo vacio");
    }else if ((th.size()==1 || keys.length==1 ||
        adjList.length == 1) && a1==a2){
        enc = true;
    }else if ((th.size()==1 || keys.length==1 ||
        adjList.length == 1) && a1!=a2){
        System.out.println("Grafo de un elemento");
    }else {
        int pos1 = th.get(a1);
        int pos2 = th.get(a2);
        int act = pos1;

        boolean[] examinados = new boolean[th.size()];

        if (a1.equals(a2)) enc = true;
        else {
            porExaminar.add(pos1);
            examinados[pos1] = false;
            while (!enc && !porExaminar.isEmpty()) {
```

```
                act = porExaminar.poll();
                if (!examinados[act]) { //si el que esta por
                    examinar no esta examinado
                    examinados[act] = true;
                    if (act == pos2) { //act es el ultimo
                        enc = true;
                    } else if (adjList[act] == null) {
                        continue;
                    } else {
                        for (int i = 0; i <
                            adjList[act].size(); i++) {
                            porExaminar.add(adjList[act].get(i));
                                //anadimos todos los enlaces
                        }
                    }
                }
            }
        }
    }
    return enc;
}

public ArrayList<String> caminoConectados(String a1, String
    a2){
    ArrayList<String> camino = new ArrayList<String>();
    Queue<Integer> porExaminar = new LinkedList<Integer>();
    boolean enc = false;

    if (th==null || keys==null || adjList == null){
        System.out.println("Grafo vacio");
    }else if ((th.size()==1 || keys.length==1 ||
        adjList.length == 1) && a1==a2){
        enc = true;
    }else if ((th.size()==1 || keys.length==1 ||
        adjList.length == 1) && a1!=a2){
        System.out.println("Grafo de un elemento");
    }else if (this.estanConectados(a1,a2)) {
        int pos1 = th.get(a1);
        int pos2 = th.get(a2);
        int act = pos1;

        boolean[] examinados = new boolean[th.size()];

        if (a1.equals(a2)) enc = true;
        else {
            porExaminar.add(pos1);
            examinados[pos1] = false;
            while (!enc && !porExaminar.isEmpty()) {
                act = porExaminar.poll();
                if (!examinados[act]) { //si el que esta por
                    examinar no esta examinado
                    examinados[act] = true;
                    camino.add(keys[act]); //falta caso de
                        que no existe
```

```
                    if (act == pos2) { //act es el ultimo
                        enc = true;
                    } else if (adjList[act] == null) {
                        continue;
                    } else {
                        for (int i = 0; i <
                            adjList[act].size(); i++) {
                            porExaminar.add(adjList[act].get(i));
                                //anadimos todos los enlaces
                        }
                    }
                }
            }
        }
    }
    int anterior = 0;
    int actual = 0;
    int tamano = camino.size()-1;
    for (int i=tamano; i>0; i--){
        anterior =
            Internet.getMiInternet().web2Id(camino.get(i-1));
        actual =
            Internet.getMiInternet().web2Id(camino.get(i));
        if (adjList[anterior]==null) { //anterior no esta en
            adjList
            camino.remove(camino.get(i-1));
        }else if (adjList[anterior]!=null &&
            !adjList[anterior].contains(actual)) {
            camino.remove(camino.get(i-1));
        }
    }
    return camino;
}

public void printGrafo(){
    System.out.println(this.th);
    for (int i = 0; i < adjList.length; i++) {
        System.out.println(i+" => "+adjList[i]);
    }
}

    public void printCamino(ArrayList<String> pCamino) {
        System.out.print("[ ");
        for( int i=0; i<pCamino.size(); i++) {
            System.out.print(pCamino.get(i)+" --> ");
        }
        System.out.print(" ]");
    }

   public void pruebaGrafo() {

        System.out.println("PRUEBA DEL GRAFO");
        System.out.println("Introduce un ID: ");
        this.inicializarGrafo();
```

```
        int id1 = Teclado.leerInteger();
        String web1 = keys[id1];
        System.out.println("ID :"+id1+"...URL: "+web1);
        System.out.println("Introduce un ID: ");
        int id2 = Teclado.leerInteger();
        String web2 = keys[id2];
        System.out.println("ID :"+id2+"...URL: "+web2);
        Stopwatch clock = new Stopwatch();
        ArrayList<String> array = this.caminoConectados(web1,
            web2);

        if (array.size()>0) {
            System.out.println("ESTAN CONECTADOS");
            this.printCamino(array);
        }else {
            System.out.println("NO ESTAN CONECTADOS");
        }
        System.out.println();
        System.out.println("Tiempo transcurrido:
            "+clock.elapsedTime());
    }

    public void resetearGrafo(){
        this.th.clear();
        this.keys = null;
        this.adjList = null;
        //System.out.println("SE HA RESETEADO: "+th.size());
    }

    public HashMap<String, Double> getPageRank(){
        return this.pageRank;
    }

    public void setTh(HashMap<String, Integer> th) {
        this.th = th;
    }

    public void setKeys(String[] keys) {
        this.keys = keys;
    }

    public void setAdjList(ArrayList<Integer>[] adjList) {
        this.adjList = adjList;
    }


public ArrayList<Par> buscar(String palabraClave) {
    ArrayList<Par> resultado = new ArrayList<>();
    this.pageRank();
    System.out.println("SE HA REALIZADO EL PAGE RANK");
    Set<Map.Entry<String, Double>> mapaEntrada =
        this.pageRank.entrySet();
    Iterator<Map.Entry<String, Double>> itr =
        mapaEntrada.iterator();
```

```
        int i = 0;
        while(itr.hasNext()) {  //Asignamos la cantidad que va a
            dividir cada web
            Map.Entry<String, Double> entrada = itr.next();
            if (entrada.getKey().contains(palabraClave)) {
                //Introducimos la palabra de manera ordenada en
                    el ArrayList
                resultado.add(new Par(entrada.getKey(),
                    entrada.getValue()));
            }
        }

        //Ordenamos el ArrayList de Par
        Collections.sort(resultado,Par.Comparators.PR);
        for (int j = 0; j < resultado.size(); j++) {
        System.out.println(resultado.get(j).web+" ->
            "+resultado.get(j).pageRank);
        }
        return resultado;

    }

    public HashMap<String, Double> pageRank(){
        // Post: el resultado es el valor del algoritmo PageRank
            para cada web
        //de la lista de webs

        //Ya estan inicializados loos pageRank de las webs.

        boolean acabar = false;
        Double PR;
        Double division = 0.0;
        Double suma = 0.0;
        Double[] puntuacion = new Double[this.pageRank.size()];
        Double diff = 0.0;
        Double diffAct;
        ArrayList<Integer> enlacesEntrantes = new ArrayList<>();
        ArrayList<Integer> enlaces = new ArrayList<>();
        int d2;
        int id;
        Double d1;
        while (!acabar) {
            //Recorremos las webs hasta que la diferencia sea
                menor que el umbral
            Set<Map.Entry<String, Double>> mapaEntrada =
                this.pageRank.entrySet();
            Iterator<Map.Entry<String, Double>> itr =
                mapaEntrada.iterator();
            while(itr.hasNext()) {  //Asignamos la cantidad que
                va a dividir cada web
                Map.Entry<String, Double> entrada = itr.next();
                id = th.get(entrada.getKey());
                enlaces = Enlaces.getMiEnlaces().id2Enlaces(id);
                    //obtenemos los enlaces
```

```
        //Double d1 =
            pageRank.get(this.id2Web(this.web2Id(entrada.getKey())));
        //enlacesEntrantes =
            Enlaces.getMiEnlaces().referenciados(th.get(entrada.getKey()));
        if (enlaces==null || enlaces.size()==0 ){
            puntuacion[id] = 0.0;
        }else{
            puntuacion[id] = (double)
                entrada.getValue()/enlaces.size();
        }
    }

    //Recorremos el HashMap de pageRank y obtenemos los
        enlaces referenciados
    Set<Map.Entry<String,Double>> mapaEntradaPR =
        pageRank.entrySet();
    Iterator<Map.Entry<String, Double>> itrPR =
        mapaEntradaPR.iterator();
    while(itrPR.hasNext()) {
        Map.Entry<String, Double> entradaPR =
            itrPR.next();
        id = th.get(entradaPR.getKey());
        //Buscamos sus enlaces y obtenemos la suma de
            dividendo
        enlacesEntrantes =
            Enlaces.getMiEnlaces().referenciados(id);
        if (enlacesEntrantes!=null) {
            for (int i = 0; i < enlacesEntrantes.size();
                i++) {
                suma = suma +
                    (puntuacion[enlacesEntrantes.get(i)]);
                    //obtenemos el sumatorio PR(i)/C(i)
            }
        }else{
            suma = 0.0;
        }
        PR =
            ((1-0.85)/this.pageRank.size())+(0.85*(suma));
            //Calculamos el PR del actual
        diffAct =
            PR-this.pageRank.get(entradaPR.getKey());
        if (diffAct<0){ //valor absoluto
            diffAct = diffAct*(-1);
        }
        diff = diff + diffAct; //introducimos la
            diferencia
        pageRank.put(entradaPR.getKey(),PR);
        suma = 0.0;
    }
    if ((diff < 0.0001)) {
        acabar = true;
    }else{
        diff = 0.0;
    }
```

```
        }

        return pageRank;
    }

    public void imprimirHash() {
        this.pageRank();
        Set<Map.Entry<String,Double>> mapaEntrada =
            this.pageRank.entrySet();
        Iterator<Map.Entry<String,Double>> itr =
            mapaEntrada.iterator();
        int i = 0;
        while(itr.hasNext()) {  //Asignamos la cantidad que va a
            dividir cada web
            Map.Entry<String,Double> entrada = itr.next();
            System.out.println(i);
            System.out.print(entrada.getValue());
            System.out.print(" -> ");
            System.out.println(entrada.getKey());
        }
    }
```

5.2 Clase Par

```
%import java.util.Comparator;

public class Par implements Comparable<Par>{


    String web;
    Double pageRank;

    public Par(String pWeb, Double pPage) {
        this.web = pWeb;
        this.pageRank = pPage;
    }

    @Override
    public int compareTo(Par o) {
        return Comparators.PR.compare(this,o);
    }

    public static class Comparators {

        public static Comparator<Par> PR = new
            Comparator<Par>() {
            @Override
            public int compare(Par o1, Par o2) {
                return o1.pageRank.compareTo(o2.pageRank);
            }
        };
    }
}
```

5.3 Clase pruebaGrafo

```
%public class TestGrafo {

    Grafo grafo;
    String w0,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10;
    HashMap<String, Integer> thAux;
    String[] keysAux;
    ArrayList<Integer>[] adjListAux;

    @Before
    public void setUp() throws Exception {
        MAIN.leerFicherosPruebas("indexGrafo","pldGrafo");
        grafo = Grafo.getMiGrafo();
        grafo.inicializarGrafo();
        //grafo.printGrafo();
        w0 = "web0";
        w1 = "web1";
        w2 = "web2";
        w3 = "web3";
        w4 = "web4";
        w5 = "web5";
        w6 = "web6";
        w7 = "web7";
        w8 = "web8";
        w9 = "web9";
        w10 = "web10";
    }

    @After
    public void tearDown() throws Exception {
        grafo.resetearGrafo();
    }



    @Test
    public void estanConectados() {
        System.out.println("PRUEBA 1: Grafo con 10 nodos.
            Caminos posibles, imposibles y desde la raiz
            hasta los ultimos elementos.");
        assertTrue(grafo.estanConectados(w0,w7));
        assertTrue(grafo.estanConectados(w7,w4));
        assertTrue(grafo.estanConectados(w1,w4));
        assertTrue(grafo.estanConectados(w0,w3));
        assertTrue(grafo.estanConectados(w0,w6));
        assertTrue(grafo.estanConectados(w0,w5));

        assertFalse(grafo.estanConectados(w1,w7));
        assertFalse(grafo.estanConectados(w7,w3));
        assertFalse(grafo.estanConectados(w6,w5));

        System.out.println("PRUEBA 2: Misma distancia
            distinto camino");
```

```
        assertTrue(grafo.estanConectados(w0,w4));

        System.out.println("PRUEBA 3: Grafo vacio");
        grafo.resetearGrafo();
        MAIN.leerFicherosPruebas("indexvacioGrafo","pldvacioGrafo");
        assertFalse(grafo.estanConectados(w1,w2));

        System.out.println("PRUEBA 4: un elemento");
        grafo.resetearGrafo();
        thAux = new HashMap<String, Integer>();
        thAux.put(w0,0);
        keysAux = new String[1];
        keysAux[0] = "web0";
        adjListAux = new ArrayList[1];
        adjListAux[0] = null;
        grafo.setAdjList(adjListAux);
        grafo.setKeys(keysAux);
        grafo.setTh(thAux);

        assertFalse(grafo.estanConectados(w0,w1));

        System.out.println("PRUEBA 5: Un solo nodo y se
            apunta a si mismo");

        adjListAux[0] = new ArrayList<Integer>();
        adjListAux[0].add(0);
        assertTrue(grafo.estanConectados(w0,w0));

    }

    @Test
    public void caminoConectados() {
        ArrayList<String> a = new ArrayList<String>();
        a.add(w0);
        a.add(w1);
        a.add(w2);
        a.add(w3);
        ArrayList<String> b = new ArrayList<String>();
        b.add(w0);
        b.add(w1);
        b.add(w2);
        b.add(w4);
        b.add(w6);
        ArrayList<String> c = new ArrayList<String>();
        c.add(w0);
        c.add(w7);
        c.add(w8);
        c.add(w9);
        c.add(w10);
        ArrayList<String> d = new ArrayList<String>();
        d.add(w0);
        d.add(w1);
        d.add(w2);
        d.add(w4);
```

```
        ArrayList<String> e = new ArrayList<String>();
        e.add(w0);

        System.out.println("PRUEBA 1: Grafo con 10 nodos.
            Caminos posibles, imposibles y desde la raiz
            hasta los ultimos elementos.");
        assertEquals(a,grafo.caminoConectados(w0,w3));
        assertEquals(b,grafo.caminoConectados(w0,w6));
        assertEquals(c,grafo.caminoConectados(w0,w10));

        System.out.println("PRUEBA 2: Misma distancia
            distinto camino");
        assertEquals(d,grafo.caminoConectados(w0,w4));

        System.out.println("PRUEBA 3: Grafo vacio");
        grafo.resetearGrafo();
        MAIN.leerFicherosPruebas("indexvacioGrafo","pldvacioGrafo");
        assertEquals(0,grafo.caminoConectados(w1,w2).size());

        System.out.println("PRUEBA 4: un elemento");
        grafo.resetearGrafo();
        thAux = new HashMap<String, Integer>();
        thAux.put(w0,0);
        keysAux = new String[1];
        keysAux[0] = "web0";
        adjListAux = new ArrayList[1];
        adjListAux[0] = null;
        grafo.setAdjList(adjListAux);
        grafo.setKeys(keysAux);
        grafo.setTh(thAux);

        assertEquals(0, grafo.caminoConectados(w0,w1).size());
    }

    @Test
    public void pageRank() {
        HashMap<String, Double> pageRank = new HashMap<>();
        pageRank = grafo.getPageRank();

        System.out.println("Pruebas para el metodo page rank.
            Los numeros correspodientes a");
        System.out.println("Cada uno de los page ranks se ha
            cAcalculado a mano, debido a la imposibilidad de
            automatizacion.");

        System.out.println("Grafo de diez elementos");
        System.out.println("Primero se imprimiran todas a la
            vez, y luego se podran ir analizando una a una");

        grafo.imprimirHash();

        System.out.println("El page rank de la web 0 es:
            0.01363636363636364");
        System.out.println("Y deberia ser o estar cerca de:
```

```
    0.013636");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 1 es:
    0.01363636363636364");
System.out.println("Y deberia ser o estar cerca de:
    0.013636");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 2
    es:0.02522727272727273");
System.out.println("Y deberia ser o estar cerca de:
    0.025228");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 3 es:
    0.02435795454545455");
System.out.println("Y deberia ser o estar cerca de:
    0.024436");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 4 es:
    0.03926676136363637");
System.out.println("Y deberia ser o estar cerca de:
    0.039267");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 5 es:
    0.030324737215909094");
System.out.println("Y deberia ser o estar cerca de:
    0.030320");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 6 es:
    0.030324737215909094");
System.out.println("Y deberia ser o estar cerca de:
    0.03029");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 7
    es:0.01363636363636364");
System.out.println("Y deberia ser o estar cerca de:
    0.01363");
assertTrue(true); //Cambiar a false si no fuciona,
    true si funciona.

System.out.println("El page rank de la web 8
```

```
            es:0.02522727272727273");
        System.out.println("Y deberia ser o estar cerca de:
            0.02523");
        assertTrue(true); //Cambiar a false si no fuciona,
            true si funciona.

        System.out.println("El page rank de la web 9
            es:0.03507954545454546 ");
        System.out.println("Y deberia ser o estar cerca de:
            0.03508");
        assertTrue(true); //Cambiar a false si no fuciona,
            true si funciona.

        System.out.println("El page rank de la web 10
            es:0.03507954545454546 ");
        System.out.println("Y deberia ser o estar cerca de:
            0.03508");
        assertTrue(true); //Cambiar a false si no fuciona,
            true si funciona.

        System.out.println("PRUEBA 2: Grafo vacio");
        grafo.resetearGrafo();
        MAIN.leerFicherosPruebas("indexvacioGrafo","pldvacioGrafo");
        grafo = new Grafo();
        grafo = grafo.getMiGrafo();
        grafo.inicializarGrafo();
        pageRank = null;
        pageRank = grafo.getPageRank();
        System.out.println("Aseguramos mediante un
            assertEquals que el hashmap de los page rank esta
            vacio.");
        assertEquals(0, pageRank.size());

        System.out.println("PRUEBA 3: Grafo de un elemento.");
        MAIN.leerFicherosPruebas("indexUnElemento","pldGrafoUnElemento");
        grafo = new Grafo();
        grafo = grafo.getMiGrafo();
        grafo.inicializarGrafo();
        pageRank = null;
        pageRank = grafo.pageRank();
        grafo.imprimirHash();
        System.out.println("Deberia dar o estar cerca de:
            0.15, y da: 0.15000000000000002");

    }

    @Test
    public void testBuscar() {
        w0 = "web0";
        w1 = "web1";
        w3 = "web3";
        w10 = "web10";
        Par p0 = new Par (w0,0.01363636363636364);
        Par p1 = new Par (w1,0.019431818181818186);
```

```
Par p3 = new Par (w3,0.026451562500000005);
Par p10 = new Par (w10,0.0303247372159091);


Integer cont;

Integer l1,l2;

ArrayList<Par> grafoaux = grafo.buscar(w0);
ArrayList<Par> grafotmp0 = new ArrayList<Par>();
ArrayList<Par> grafotmp1 = new ArrayList<Par>();
ArrayList<Par> grafotmp3 = new ArrayList<Par>();
grafotmp0.add(p0);
cont = grafoaux.size();
Boolean seguir = true;
l1=grafoaux.size();
l2=grafotmp0.size();

if (l1!=l2) {
    fail("Casca");
}
else {
    while (cont>1 && seguir) {
        if
            (grafoaux.get(cont-1).pageRank.equals((grafotmp0.get(cont-1))
        .pageRank)&&(grafoaux.get(cont-1).web.equals(grafotmp0.
        get(cont-1).web))) {}
        else {
            fail("Casca");
            seguir = false;
        }
        cont--;
    }

}
grafoaux = grafo.buscar(w1);
grafotmp1.add(p1);
grafotmp1.add(p10);
cont = grafoaux.size();
seguir = true;
l1=grafoaux.size();
l2=grafotmp1.size();
if (l1!=l2) {
    fail("Casca");
}
else {
    while (cont>1 && seguir) {
        if
            (grafoaux.get(cont-1).pageRank.equals((grafotmp1.get(cont-1))
        .pageRank)&&(grafoaux.get(cont-1).web.equals(grafotmp1.
        get(cont-1).web))) {}
        else {
            fail("Casca");
            seguir = false;
```

```
                }
                cont--;
            }

        }

        grafoaux = grafo.buscar(w3);
        grafotmp3.add(p3);;
        cont = grafoaux.size();
        seguir = true;
        l1=grafoaux.size();
        l2=grafotmp3.size();
        if (l1!=l2) {
            fail("Casca");
        }
        else {
            while (cont>1 && seguir) {
                if
                    (grafoaux.get(cont-1).pageRank.equals((grafotmp3.get(cont-1))
                .pageRank)&&(grafoaux.get(cont-1).web.equals(grafotmp3.
                get(cont-1).web))) {}
                else {
                    fail("Casca");
                    seguir = false;
                }
                cont--;
            }

        }


    }
}
```

Capítulo 6

Conclusiones

Con este trabajo hemos aprendido; aunque de manera reducida, como funcionaba Google años atrás. Nos ha parecido interesante y nos ha abierto los ojos sobre las grandes cualidades que tienen los grafos.

En cuanto a la resolución del trabajo, al principio de la tarea hemos tenido un contratiempo respecto a como recuperar el PageRank anterior (para el calculo de la diferencia). Nos imos cuenta de que con el uso de un HashMap almacendando el pagerank arreglabamos ese problema. Ese HashMap se rellena cuando se leen los ficheros, y se le asigna a cada web su valor por defecto: 0.25.