PAGE RANK

Fecha: 22-XII-2021

Participantes:
Xabier Gabiña
Javier Criado

Índice

1.Introducción

Para el laboratorio 4 hemos tenido que desarrollar el algoritmo pageRank(), el cual es un algoritmo para numerar la relevancia de un conjunto de datos. Este algoritmo fue desarrollado por Google y es el que usa en su navegador.

2.Diseño de clases

Hemos realizado 4 clases Main, Menu, Par y GraphHash.

La clase Main es una TAD que se encargara de llamar al método showMenu() de la clase Menu que inicia la aplicación.

La clase Menu es una MAE. El método showMenu() muestra un menú con diferentes opciones donde tendrás que elegir la opción correspondiente a la acción que quieras realizar.

La clase Par es una TAD compuesta por los atributos actor y pageRank. El método getName devuelve el nombre del actor y el método getPageRank devuelve el pageRank del actor.

La clase GraphHash es una MAE compuesta por un atributo de tipo HashMap<String , ArrayList<String>>. El método crearGrafo() se encargará de pedir al usuario el tamaño del grafo sobre el que quiere trabajar y entonces llamara al método cargarFichero para cargar los datos,.el método cargarFichero() se encargara de cargar los datos del fichero, el metodo print() imprimirá todos los elementos del grafo, el método add(String pClave, String pDato) añade los datos introducidos , el método estanConectados() llama al método estanConectados(String s1, String s2), el método estanConectados(String s1, String s2) devuelve un booleano indicando si ambos actores están conectados, el método listaConectados() devuelve los elementos que se encuentran entre los dos elementos introducidos, el método ListaConectados((String s1, String s2) devuelve los actores que hay entre los actores dados, el método pathFinder(String a1, String a2) devuelve el recorrido entre los dos actores dados, el método pageRank(boolean print) devuelve un HashMap<String, Double> con el valor de cada actor del Grafo, el método ordenarPorPageRank() devuelve un ArrayList de tipo Par donde la lista esta ordenada en función del valor de cada actor, el método ordenarPorPageRank(ArrayList<String> Actores) devuelve un ArrayList de tipo Par donde la lista esta ordenada en función del pageRank de cada actor.

3. Diseño e implementación de los métodos principales

3.1 crearGrafo()

//Pre:

Casos de prueba:

- Pedir un número valido
- Pedir un número no valido

Implementación:

Pedir el tamaño del grafo a crear

Cargar el fichero elegido

Coste: O(n) ya que es el encargado de ejecutar cargarFichero() que tiene coste O(n)

3.2 print()

//Pre:

Casos de prueba:

Implementación:

Imprime todos los elementos

Coste: O(m*n) siendo m el número de elementos que componen el HashMap y n el número de valores que contiene cada clave.

3.3 add()

//Pre:

Casos de prueba:

-Existe la clave dada

-No existe la clave dada

Implementación:

Si existe pClave

Conseguir su valor

Si el dato no es null

se añade

Coste: O(1) ya que añadir a un ArrayList tiene más o menos conste constante.

3.4 public boolean estanConectados()

//Pre:

Casos de prueba:

Implementación:

Pedir el primer elemento a buscar

Pedir el segundo elemento a buscar

Devolver el resultado de llamar al método estanConectados (s1, s2)

Coste: El coste será O(1) en el mejor de los casos y O(n) en el peor de los casos siendo n el número de elementos que tenga el HashMap.

3.5 public ArrayList<String> listaConectados()

//Pre:

Casos de prueba:

Implementación:

Pedir el primer elemento a buscar

Pedir el segundo elemento a buscar

Llamar al método listaConectados(1ºelem, 2º elem)

Si están conectados

Devolver los elementos que los conectan

Si no

Devolver una lista vacía

Coste: El coste será O(1) en el mejor de los casos y O(n) en el peor de los casos siendo n el número de elementos que tenga el HashMap.

3.6 public HashMap<String , Double> pageRank(boolean print)

//Pre:

Casos de prueba:

Implementación:

Mientras que diferencia sea mayor que el límite

Si iteración == 0

Dar mismo valor a todos los objetos

Sino

Para cada elemento del HashMap

Para cada uno de los enlaces

Calcular el sumatorio con el pagerank anterior y numero de enlaces del enlace

Actualizar el pageRank del elemento

Por cada elemento del HahMap

Actualizar la diferencia

Devolver el HashMap

Coste: O(2n*r) siendo n el numero de nodos del grafo y r la media de relaciones que tiene cada nodo del grafo.

3.7 public ArrayList<Par> ordenarPorPageRank()

//Pre:

Casos de prueba:

Implementación:

Recorrer todos los actores del HashMap

Meter los actores en una lista

Llamar al método ordenarPorPageRank() y pasar la lista como parámetro

Devolver el resultado del método ordenarPorPageRank(lista)

Coste: El coste de ordenar la lista mediante comparaciones (mergesort) es de $O(N*\log_2 N)$.

4. Código

4.1 Main:

```java
package org.eda.lab4;

public class Main {
    public static void main(String[] args) {
        Menu.getMenu().showMenu();
    }
}
```

4.2 Menu:

```
package org.eda.lab4;

import java.util.Scanner;
import java.util.concurrent.atomic.AtomicInteger;

public class Menu {
    //Attributes
    private static Menu miMenu=null;

    //Constructor
    private Menu(){}

    //Methods
    public static Menu getMenu()
    {
        if(miMenu==null)
        {
            miMenu=new Menu();
        }
        return miMenu;
    }

    public void showMenu()
    {
        Scanner sn=new Scanner(System.in);
        boolean exit=false;
        int opcion;
        GraphHash GH= GraphHash.getGraphHash();

        while(!exit)
        {

System.out.println("\n##################################################
");
            System.out.println("\t\t Menu Principal \n");
            System.out.println("Seleccione una de las siguientes
opciones:\n");
            System.out.println("0. Cargar los datos");
            System.out.println("1. Comprobar conexion");
            System.out.println("2. Obtener lista de conexion");
            System.out.println("3. Calcular PageRank");
            System.out.println("4. Obtener lista ordenada de
PageRank");
            System.out.println("8. Imprimir GraphHash");
            System.out.println("9. Finalizar Programa");
            System.out.print("---> ");
            opcion=sn.nextInt();

            switch (opcion) {
                case 0 :
                    GH.crearGrafo();
                    break;
                case 1 :
                    if(GH.estanConectados())
                        System.out.print("SI estan conectados");
                    else
                        System.out.print("NO estan conectados");
                    break;
                case 2:
                    GH.listaConectados().forEach(s ->
```

```
System.out.print("<"+s+"> "));
                    break;
                case 3:
                    GH.pageRank(true);
                    break;
                case 4:
                    AtomicInteger i = new AtomicInteger();
                    GH.ordenarPorPageRank().forEach(p ->
System.out.println(i.getAndIncrement() +"\t<"+p.getName()+" |
"+String.format("%.14f",p.getPageRank())+"> "));
                    break;
                case 8 :
                    GH.print();
                    break;
                case 9 :
                    exit = true;
                    break;
                default :
                    System.out.println("Introduce un numero
valido\n");
            }
        }
    }
}
```

4.3 Par:

```java
package org.eda.lab4;

public class Par {
    private String actor;
    private Double pageRank;

    public Par(String key, Double aDouble) {
        actor=key;
        pageRank=aDouble;
    }

    public String getName()
    {
        return actor;
    }

    public Double getPageRank()
    {
        return pageRank;
    }
}
```

4.4 GraphHash:

```
package org.eda.lab4;

import java.io.FileReader;
import java.io.IOException;
import java.util.*;

public class GraphHash
{
    private HashMap<String, ArrayList<String>> g;
    private HashMap<String, Double> HS_pageRank=new HashMap<>();
    private static GraphHash miGH=null;

    private GraphHash() {g=new HashMap<>();}

    public static GraphHash getGraphHash()
    {
        if(miGH==null)
        {
            miGH= new GraphHash();
        }
        return miGH;
    }

    public void crearGrafo()
    {
        Scanner sn=new Scanner(System.in);
        int opcion;
        String Dir="Laboratorio 4/src/files/lista.txt";
        System.out.println("\nSeleccione una lista sobre la que
trabajar:");
        System.out.println("0. Lista de 10 elementos");
        System.out.println("1. Lista de 20.000 elementos");
        System.out.println("2. Lista de 50.000 elementos");
        System.out.println("3. Lista completa");
        System.out.println("4. Otra lista (Meter direccion a mano)");
        System.out.print("---> ");
        opcion=sn.nextInt();
        sn.nextLine();
        switch (opcion) {
            case 0:
                Dir="Laboratorio 4/src/files/lista.txt";
                break;
            case 1:
                Dir="Laboratorio 4/src/files/lista_20000.txt";
                break;
            case 2:
                Dir="Laboratorio 4/src/files/lista_50000.txt";
                break;
            case 3:
                Dir="Laboratorio 4/src/files/lista_completa.txt";
                break;
            case 4:
                System.out.println("Introducce una direccion valida");
                Dir=sn.nextLine();
                break;
            default:
                System.out.println("Solo numeros del 0 al 4\n");
        }
        g.clear();
        cargarFichero(Dir);
```

```
    }

    private void cargarFichero(String Dir)
    {
        long statTime=System.nanoTime();
        try
        {
            Scanner entrada = new Scanner(new FileReader(Dir));
            String linea;
            while (entrada.hasNext())
            {
                linea=entrada.nextLine();

                String [] sub1 = linea.split(" --->>>"+"\\s+");
                String [] sub2 = sub1[1].split(" #####"+"\\s+");

                for (String s : sub2) {
                    for (String value : sub2) {
                        if (s.compareTo(value) != 0)
                            GraphHash.getGraphHash().add(s, value);
                        else
                            GraphHash.getGraphHash().add(s,"");
                    }
                }
            }
            entrada.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        long endTime=System.nanoTime();
        System.out.println(((endTime-statTime)/1000000000)+" segundos
a tardado en ejecutarse");
    }

    public void print()
    {
        int i=1;
        for(String s:g.keySet())
        {
            System.out.print("Element:"+ i++ + "\t" + s + " --> ");
            for(String k: g.get(s))
            {
                System.out.print(k+ " ### ");
            }
            System.out.println();
        }
    }

    public void add(String pClave, String pDato)
    {
        ArrayList<String> lDatos;
        if(g.containsKey(pClave))
        {
            lDatos = g.get(pClave);
        }
        else
        {
            lDatos = new ArrayList<>();
        }
```

```
        if(pDato.compareTo("")!=0)
            lDatos.add(pDato);
        g.put(pClave, lDatos);

    }

    public boolean estanConectados()
    {
        System.out.println("Introduce el nombre del primer actor");
        Scanner sn=new Scanner(System.in);
        String a1=sn.nextLine();
        System.out.println("Introduce el nombre del segundo actor");
        String a2=sn.nextLine();
        return estanConectados(a1,a2);
    }

    private boolean estanConectados(String a1, String a2)
    {
        if(!g.containsKey(a1))
        {
            System.out.println(a1+" no esta en la base de datos");
            return false;
        }
        else
        {
            if(!g.containsKey(a2))
            {
                System.out.println(a2+" no esta en la base de datos");
                return false;
            }
            else
            {
                long statTime=System.nanoTime();
                if(g.get(a1).isEmpty() || g.get(a2).isEmpty())
                {
                    long endTime=System.nanoTime();
                    System.out.println(((endTime-
statTime)/1000000000)+" segundos a tardado en comprobar conexion");
                    return false;
                }
                else if(!g.get(a1).contains(a2))
                {
                    Queue<String> sinExplorar = new
LinkedList<>(g.get(a1));
                    HashSet<String> HSsinExplorar = new
HashSet<>(g.get(a1));
                    HashSet<String> Explorados = new HashSet<>();
                    boolean enc = false;
                    Explorados.add(a1);
                    while (!HSsinExplorar.isEmpty() && !enc) {
                        String unActor = sinExplorar.remove();
                        HSsinExplorar.remove(unActor);
                        Explorados.add(unActor);
                        if (unActor.compareTo(a2) == 0) {
                            enc = true;
                        } else {
                            g.get(unActor).forEach(s ->
                            {
                                if (!Explorados.contains(s) &&
!HSsinExplorar.contains(s)) {
                                    sinExplorar.add(s);
```

```
                                    HSsinExplorar.add(s);
                                }
                            });
                        }
                    }
                    long endTime=System.nanoTime();
                    System.out.println(((endTime-
statTime)/1000000000)+" segundos a tardado en comprobar conexion");
                    return enc;
                }
                else
                {
                    long endTime=System.nanoTime();
                    System.out.println(((endTime-
statTime)/1000000000)+" segundos a tardado en comprobar conexion");
                    return true;
                }
            }
        }
    }

    public ArrayList<String> listaConectados()
    {
        System.out.println("Introduce el nombre del primer actor");
        Scanner sn=new Scanner(System.in);
        String a1=sn.nextLine();
        System.out.println("Introduce el nombre del segundo actor");
        String a2=sn.nextLine();
        if(estanConectados(a1,a2))
            return listaConectados(a1,a2);
        else
            return new ArrayList<>();
    }

    private ArrayList<String> listaConectados(String a1, String a2)
    {
        return pathFinder(a1,a2);
    }

    private ArrayList<String> pathFinder(String a1, String a2)
    {
        long statTime=System.nanoTime();

        Queue<String> sinExplorar = new LinkedList<>(g.get(a1));
        HashSet<String> HSsinExplorar = new HashSet<>(g.get(a1));
        HashSet<String> Explorados = new HashSet<>();
        HashMap<String, String> backpointers=new HashMap<>();
        String unActor=null;
        boolean enc = false;

        Explorados.add(a1);
        backpointers.put(a1,null);
        while (!HSsinExplorar.isEmpty() && !enc)
        {
            unActor = sinExplorar.remove();
            HSsinExplorar.remove(unActor);
            Explorados.add(unActor);
            if (unActor.compareTo(a2) == 0)
            {
                enc = true;
            } else
```

```
            {
                String finalUnActor = unActor;
                g.get(unActor).forEach(s ->
                {
                    if (!Explorados.contains(s) &&
!HSsinExplorar.contains(s)) {
                        sinExplorar.add(s);
                        HSsinExplorar.add(s);
                        backpointers.put(s, finalUnActor);
                    }
                });
            }
        }
        if(enc)
        {
            ArrayList<String> lista = new ArrayList<>();

            while (unActor!=null)
            {
                lista.add(unActor);
                unActor=backpointers.get(unActor);
            }
            lista.add(a1);
            Collections.reverse(lista);
            long endTime=System.nanoTime();
            System.out.println(((endTime-statTime)/1000000000)+"
segundos a tardado en encontrar el camino");
            return lista;
        }
        else
        {
            long endTime=System.nanoTime();
            System.out.println(((endTime-statTime)/1000000000)+"
segundos a tardado en encontrar el camino");
            return new ArrayList<>();
        }
    }

    public HashMap<String, Double> pageRank(boolean print)
    {
        HashMap<String, Double> itr_prev=new HashMap<>();
        HashMap<String, Double> itr_act=new HashMap<>();
        double limite=0.0001;
        double diff=1;
        int itr=0;
        double d=0.85;
        double N = g.size();
        long statTime=System.nanoTime();
        while(diff > limite)
        {
            if(itr==0)  //En la primera iteracion damos el mismo valor
a todos los elementos
            {
                for(Map.Entry<String, ArrayList<String>>
entry:g.entrySet())
                {
                    itr_act.put(entry.getKey(), 1/N);
                }
            }
            else
            {
```

```
                itr_prev.putAll(itr_act);

                for(Map.Entry<String, ArrayList<String>>
entry:g.entrySet())
                {
                    double sum=0;
                    for(String s:entry.getValue())
                    {
                        sum+=itr_prev.get(s)/g.get(s).size();
                    }

                    itr_act.put(entry.getKey(), ((1-d)/N)+d * sum);
                }
                diff=0;
                for(Map.Entry<String,ArrayList<String>>
entry:g.entrySet())
                {
                    Double dif=(Math.abs(itr_prev.get(entry.getKey())-
itr_act.get(entry.getKey())));
                    diff+=dif;
                }
            }

            long endTime=System.nanoTime();
            if(print)

System.out.println("\titeracion:\t"+itr+"\t\tdiff:\t"+String.format("%
.14f",diff)+"\t\ttime:\t"+((endTime-statTime)/1000000000)+"s");
            itr++;
        }

        HS_pageRank.putAll(itr_act);
        return itr_act;
    }

    public ArrayList<Par> ordenarPorPageRank()
    {
        long statTime=System.nanoTime();
        ArrayList<String> actores=new ArrayList<>();
        for(Map.Entry<String,ArrayList<String>> entry:g.entrySet())
        {
            actores.add(entry.getKey());
        }
        ArrayList<Par> parejas = ordenarPorPageRank(actores);
        long endTime=System.nanoTime();
        System.out.println(((endTime-statTime)/1000000000)+" segundos
a tardado en ordenar la lista\n");
        return parejas;
    }

    private ArrayList<Par> ordenarPorPageRank(ArrayList<String>
actores)
    {
        if(HS_pageRank.isEmpty()) {
            HS_pageRank = pageRank(false);
        }
        ArrayList<Par> act=new ArrayList<>();
        for(String s:actores)
        {
            Par par=new Par(s, HS_pageRank.get(s));
            act.add(par);
```

```
        }

act.sort(Comparator.comparingDouble(Par::getPageRank).reversed());
        return act;
    }
}
```

5.Conclusiones

Este laboratorio nos ha servido sobre todo para aprender como a a partir de una formula matemática como era la del PageRank a desarrollar un algoritmo a la vez que utilizamos unas estructuras de datos eficientes y capaces de cumplir su función correctamente en el algoritmo.

El principal reto que hemos encontrado a sido el de pasar de una expresión matemática a un programa ya que no es fácil a veces entender como obtener los datos que nos pide la expresión de las estructuras de datos y trabajarlos correctamente. También fue un reto la parte de ordenar los Pares ya no eran simplemente números, aunque el método sort de la propia clase ArrayList y la clase Comparator facilitaron mucho esta tarea.