Aplicación para gestionar actores y películas

Fecha: 25-XI-2021

Participantes: Xabier Gabiña
Javier Criado

1. Índice

2. Introducción

Para el laboratorio 3 se nos ha pedido obtener un sistema que permita hacer eficientemente el método estanConectados(String a1, String a2) donde dodos 2 nombres nos indique si están conectados.

3. Diseño de clases

Hemos realizado 3 clases Main, Menu y GraphHash.

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

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.

4. Diseño e implementación de los métodos principales

4.1 public void crearGrafo()

//Pre:

Casos de prueba:

- Pedir un numero valido
- Pedir un numero 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)

4.2 public void cargarFichero()

//Pre:

Casos de prueba:

- Que no exista el fichero
- Que exista el fichero

Implementación:

Lee cada línea del fichero seleccionado y los separa por actores los cuales va incluyendo en un HashMap.

Coste: O(n) siendo n el numero de actores en el fichero.

4.3 public void 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 de valores que contiene cada clave.

4.4 public void add(String pClave, String pDato)

//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.

4.5 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.

4.6 public boolean estanConectados(String a1, String a2)

//Pre:

Casos de prueba:

Implementación:

Si el elemento a1 no se encuentra

Devolver false

Sino si el elemento 2 no se encuentra

Devolver false

Sino si los valores de a1 están vacíos o los valores de a2 están vacíos

Devolver false

Sino si el valor de s1 no contiene a a2

Mientras no encontrado y porExaminar no vacío

Si el actor es igual a a2

Devolver True

Sino Devolver falso

Sino devolver true

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.

4.7 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.

4.8 `public ArrayList<String> listaConectados(String a1, String a2)`

//Pre:

Casos de prueba:

-Introducir dos elementos iguales.

-Introducir dos elementos diferentes.

Implementación:

Devolver el resultado de llamar al método pathFinder (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.

4.9 `private ArrayList<String> pathFinder(String a1, String a2)`

//Pre:

Casos de prueba:

-Introducir dos elementos iguales.

-Introducir dos elementos diferentes.

Implementación:

Mientras no encontrado y lista sin explorar no vacía

Eliminar actor de sin explorar

Añadir actor a explorados

Si el actor es a2

Encontrado

Sino seguir buscando

Si encontrado

Devolver la lista de actores entre actores a1 y a2

Sino devolver 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.

5. Código

5.1 Main:

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

5.2 Menu:

```
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. Lista de conexion");
            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 8 :
                    GH.print();
                    break;
                case 9 :
                    exit = true;
                    break;
                default :
                    System.out.println("Introduce un numero valido\n");
            }
        }
    }
}
```

5.3 GraphHash:

```
public class GraphHash
{
    private HashMap<String, ArrayList<String>> g;
    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 3/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 3/src/files/lista.txt";
                break;
            case 1:
                Dir="Laboratorio 3/src/files/lista_20000.txt";
                break;
            case 2:
                Dir="Laboratorio 3/src/files/lista_50000.txt";
                break;
            case 3:
                Dir="Laboratorio 3/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");
        }

        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);
    }

    public 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<>();
    }

    public 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<>();
        }
    }
}
```

6. Conclusiones

Este laboratorio nos ha servido para poner un poco mas en practica lo ya aprendido con las estructuras de datos de listas, arrays, hashmaps…

Gracias a usar las estructuras de datos mas eficientes para cada solución hemos conseguido que incluso con la lista completa del laboratorio 1 en crear el grafo tenemos un tiempo de 23 segundos mientras que para hacer las búsquedas tanto el estanConectados como listaConectados tardan en hacer cualquier búsqueda de 1 a 2 segundos. Obviamente estos resultados pueden varias dependiendo del ordenador, en nuestro caso hemos usado un ordenador bastante potente por lo que es posible que los números empeoren.

En este proyecto hemos encontrado dos dificultades principalmente, la primera, relacionada con el tiempo de ejecución de estanConectados. Nuestro problema era el método contains de la clase Queue que tenia un coste lineal lo que hacia que nuestras búsquedas pasaran de apenas 1 segundo a 14 minutos de reloj. Para solucionarlo creamos un HashSet que contiene los mismos objetos que la Queue pero que permite usar contains en tiempo constante. La segunda dificultad se encontraba en el trabajo opcional ya que no teníamos muy claro como realizar este método. En primera instancia se nos ocurrió implementar Dijkstra para que nos devolviera el camino, pero no parecía la mejor solución al no tener pesos como tal en el grafo lo que complicaba el problema. Al final la solución se dio en clase usando backpoints. Para implementarlos hemos usado un HashMap que guarda las direcciones de llegadas de cada elemento y así poder recorrer el camino inverso una vez terminada la búsqueda.