# Informe del Trabajo Práctico Especial
# Programación 3 -TUDAI


*Segunda entrega*

## Fecha de entrega

01/07/2022

## Integrantes


- Celani Cañedo, Ana (anacelcan@gmail.com)
- Celani Cañedo, Mauro (maurocelcan@gmail.com)

# Índice

# Introducción

En éste informe realizado para la segunda etapa del TPE se lleva a cabo el análisis del proceso realizado para la implementación de un buscador el cual detalla distintas relaciones entre géneros literarios. El usuario podrá, a partir de un género ingresado, obtener un grafo dirigido con sus géneros afines, obtener una lista de los n géneros más buscados luego del género dado y obtener la secuencia de búsqueda más alta a partir del género dado.

A partir de varios archivos dados por la cátedra, los cuales contenían distintas secuencias de géneros, se generó un grafo dirigido como estructura de almacenamiento de estos géneros, el cual además permitió mostrar las distintas relaciones que se generaron entre estos géneros. De acuerdo con esto, los géneros quedaron representados por los vértices del grafo y las relaciones mediante los arcos, donde la etiqueta del mismo simboliza la cantidad de veces que luego de buscar el primer género, inmediatamente a continuación, se buscó por el segundo género.

Se llevó a cabo el desarrollo de los siguientes tres servicios:

- ➔ Obtener los N géneros más buscados luego de buscar por el género A.
- ➔ A partir de un género A encontrar, en tiempo polinomial, la secuencia de géneros que más alto valor de búsqueda posee. Se define el valor de búsqueda de una secuencia como la suma de los arcos que la componen.
- ➔ Obtener el grafo únicamente con los géneros afines a un género A. Es decir que, partiendo del género A, se encuentre una vinculación cerrada entre uno o más géneros la cual permita volver al género de inicio.

Para la implementación de dichos servicios se trabajó nuevamente con una clase Biblioteca, la cual contiene la estructura del grafo y los métodos que permiten al usuario hacer uso de los tres servicios descritos. Además de la clase Biblioteca se encontrará en el código también una clase GrafoDirigido el cual implementa cada uno de los métodos de la interfaz Grafo; una clase Arco, la cual contiene un vértice destino, vértice origen y una etiqueta; y por último una clase AdministradorArchivos, la cual se encarga de leer los archivos csv.

Dentro de la clase GrafoDirigido se encontrarán diferentes métodos, los cuales algunos son parte de la solución de los servicios. Estos métodos fueron realizados para el Práctico III, y no necesariamente todos los métodos son aplicados para la resolución del TPE pero le brindan funcionalidad al grafo.

Para corroborar que el grafo se estaba generando correctamente y que el servicio 3 estaba devolviendo el grafo que correspondía se utilizó la herramienta brindada por la cátedra (la cual genera un grafo a partir de cierto código) y un archivo csv pequeño generado por los participantes del grupo de trabajo.

A continuación se podrá ver el análisis del proceso de desarrollo de cada uno de los servicios.

# Desarrollo

## Clases principales

A continuación se detallan las clases mencionadas anteriormente.

- Clase ‘AdministradorArchivos’: esta clase es la encargada de la administración de los archivos .csv, es decir, la lectura de los mismos.

- Clase ‘Biblioteca’: en esta clase se encuentra el almacenamiento de los géneros, en forma de grafo, y un administrador de archivos. Esta clase, mediante el generarGrafoGeneros() y la lectura del archivo .csv, crea cada uno de los géneros, lo almacena en el grafo y luego crea las relaciones que se van dando entre los mismos. A medida que la secuencia de búsqueda entre dos géneros se empieza a repetir el peso del arco que une ambos vértices comienza a aumentar. Por otro lado, la clase dispone de otros seis métodos más, los cuales resuelven los tres servicios pedidos.

- Clase ‘GrafoDirigido’: esta clase implementa los métodos de la interfaz Grafo. Dentro de la misma se halla un HashMap llamado ‘map’ donde la clave es un String, representando cada uno de los géneros, y los valores son un ArrayList<Arco> representando la relación entre la clave (género) y el género que tiene como destino el Arco.

- Clase ‘Arco’: en esta clase, como ya se mencionó, hace referencia a la relación entre dos géneros, obtenida a partir de los archivos csv. En la misma se encuentran como atributos, el género que se buscó primero (vértice origen) y el género que se buscó a continuación del mismo (vértice destino); y una etiqueta representando la cantidad de veces que se produjo esta secuencia de búsqueda dentro del archivo csv.

## Implementación del Servicio I

El primer servicio del trabajo solicitaba la obtención de los N géneros más buscados luego un género dado por el usuario. Para poder definir cuándo un género es más buscado que otro a partir del género dado por el usuario, se debe mirar cada uno de los arcos que tienen como origen el género dado. En particular, lo que se debe observar del arco es el valor de la etiqueta. Si la etiqueta de un arco es mayor que la etiqueta de otro arco, entonces el género como destino del primer arco fue más buscado que el género destino del segundo arco.

A partir de esto, se reconoció la necesidad de obtener todos los arcos de un género dado. Para esta instancia se tenía implementado sobre la clase GrafoDirigido un método el cual devolvía un Iterator<Arco>. Sin embargo a medida que se avanzaba en el análisis del problema fue notoria la necesidad de tener los arcos ordenados por el valor de la etiqueta. A partir de esto se decidió redefinir el método que devolvía los arcos de manera tal que éste retornara un ArrayList<Arco> en vez de un Iterator<Arco>. Este cambio permitió ordenar los arcos mediante Collections.sort() por el valor de la etiqueta de manera descendente.

A partir de éste análisis se implementó sobre la clase Biblioteca un método, llamado generosMasBuscados, el cual recibe como parámetro el género (un String) y la cantidad de géneros que debe devolver (un int). Mediante el grafo definido en la clase Biblioteca, el cual almacena todos los géneros y sus relaciones, se hace un pedido de los arcos del género recibido como parámetro. Es decir, se le solicita al HashMap los valores que tienen como clave el género dado. Una vez obtenidos los arcos, dado que se requieren los N géneros más buscados, se ordenaron los arcos mediante Collections.sort(arcos). Por último, se definió que como solución de éste servicio se iba a retornar un ArrayList<String> con cada uno de los géneros correspondientes.

```
//SERVICIO 1

public ArrayList<String> generosMasBuscados(int n, String genero) {
    ArrayList<String> solucion = new ArrayList<>();
    int minimo = 0;
    if (this.generos.contieneVertice(genero)) {
        ArrayList<Arco> arcos = this.generos.obtenerArcos(genero);
        Collections.sort(arcos);

        if (n < arcos.size()) {
            minimo = n;
        } else {
            minimo = arcos.size();
        }

        for (int i = 0; i < minimo; i++) {
            solucion.add(arcos.get(i).getVerticeDestino());
        }
    }

    return solucion;
}
```

Método solución del Servicio I de la clase Biblioteca.

## Implementación del Servicio II

En el segundo servicio se busca a partir de un género A encontrar, en tiempo polinomial, la secuencia de géneros que más alto valor de búsqueda posee. Es necesario recordar que una secuencia de géneros de más alto valor es la cual la suma de los pesos de sus arcos sea la mayor.

Por otro lado, se solicitó que la búsqueda sea en tiempo polinomial, por lo que rápidamente se descartó la posibilidad de implementar la solución mediante la técnica de Backtracking. Esto llevó a pensar una solución mediante Greedy. De acuerdo a esta decisión lo primero que se pensó es en quienes conformarían el conjunto de candidatos y cuál sería la estrategia Greedy a usar.

Dado que cada una de las decisiones se toman en base a los pesos de los arcos, se eligió que el conjunto de candidatos estaría conformado por todos los arcos y que la estrategia Greedy sería ir eligiendo el arco con mayor peso y así avanzando a través del grafo por los vértices seleccionados. Sin embargo pareció sumamente costoso contar con absolutamente todos los arcos del grafo desde el comienzo, ya que cada vez que se quisiera elegir el arco con mayor peso de un cierto vértice se iban a tener que recorrer todos los arcos. En base a esto se optó por ir calculando el conjunto de candidatos a medida que se iba avanzando en los vértices.

Por otro lado, dado que se aplicó Greedy, fue necesario analizar la condición de corte de la iteración. Se consideraron para la condición de corte, por un lado, que el conjunto de candidatos no fuese vacío, dado que si este no tiene elementos no hay arcos que seleccionar; y por otro lado, que vértice al cual se le solicita el conjunto de candidatos no se haya visitado. Por lo tanto fue necesario contar con una estructura auxiliar en el método (HashMap colores) el cual contabiliza si el vértice ya fue tenido en cuenta o no.

El método que resuelve el servicio se hizo sobre la clase Biblioteca, secuenciaMayorValorDeBusqueda, el cual recibe como parámetro un género, el cual será el primer género de la secuencia solución. A partir de este género se solicitan los candidatos (arcos) del mismo, utilizando la misma función que en el servicio I, la cual devuelve un ArrayList<Arco>. El género se marca como “amarillo”, lo cual representa que ya fue tenido en cuenta, se agrega a la solución y se prosigue a seleccionar el próximo género.

Para la selección del próximo candidato se generó la función seleccionCandidato, la cual recibe como parámetro el ArrayList<Arco> calculado previamente. Esta función se encarga de ordenar los candidatos en base a su peso descendentemente utilizando nuevamente Collections.sort(). Una vez que se tiene los arcos ordenados se retorna el género destino que tiene el primer arco, el cual es el que tiene la etiqueta con mayor peso. Una vez obtenido el nuevo género se calculan los nuevos candidatos a partir del género obtenido de la selección. Esta iteración se repite hasta que alguna de las condiciones de corte no se cumple.

Una vez que se sale de la iteración, o porque el último género no tiene arcos que lo relacionen con otro género, o porque el próximo género ya se encuentra en la

solución, se agrega el último género por el cual pasó la iteración y se retorna la solución.

Para este segundo servicio también se decidió que el retorno sería un ArrayList<String> conteniendo la secuencia obtenida.

```
//SERVICIO 2

public ArrayList<String> secuenciaMayorValorDeBusqueda(String genero){
    ArrayList<String> secuencia = new ArrayList<>();

    if (this.generos.contieneVertice(genero)) {
        HashMap<String, String> colores = new HashMap<>();
        Iterator<String> vertices = this.generos.obtenerVertices();
        while (vertices.hasNext()) {
            String g = vertices.next();
            colores.put(g, "blanco");
        }

        String generoActual = genero;
        ArrayList<Arco> candidatos = this.generos.obtenerArcos(generoActual);

        while (!candidatos.isEmpty() && colores.get(generoActual).equals("blanco")) {

            secuencia.add(generoActual);
            colores.put(generoActual, "amarillo");
            generoActual = seleccionCandidato(candidatos);
            candidatos = this.generos.obtenerArcos(generoActual);

        }

        if (candidatos.isEmpty() && colores.get(generoActual).equals("blanco")) {
            secuencia.add(generoActual);
        }
    }

    return secuencia;
}

private String seleccionCandidato(ArrayList<Arco> candidatos) {
    //Ordenamos los candidatos descendentemente y retornamos el primer genero (el mas pesado)
    //Candidatos nunca puede ser vacio (chequeado en el while)
    Collections.sort(candidatos);
    return candidatos.get(0).getVerticeDestino();
}
```

Solución del Servicio II sobre la clase Biblioteca.

## Implementación del Servicio III

El último servicio solicitado busca obtener un grafo con los géneros afines a un género dado. Es decir, partiendo de un género dado por el usuario, se obtenga una vinculación cerrada (ciclo) el cual permita volver al género inicial.

Para este último punto se planteó una solución utilizando la técnica de Backtracking, basándose en el método DFS visto en clase. Es importante aclarar que la resolución de este servicio retorna un único grafo el cual es el ciclo hallado con mayor longitud. Es decir, se pueden haber encontrado más de un ciclo, pero la solución que se decidió implementar es la que más géneros incluya.

Nuevamente es necesario generar tres estructuras auxiliares, un HashMap para marcar los los vértices a medida que se visitan, un ArrayList<String> para guardar los géneros a medida que se recorren y otro ArrayList<String> para llevar el ciclo de mayor longitud. Se decidió trabajar con ArrayList<> dado que era necesario hacer distintas operaciones sobre la estructura (como eliminar elementos, vaciar la estructura, preguntar por el tamaño, etc.) las cuales eran menos costosas en el ArrayList<>.

Para la implementación fueron necesarios tres métodos sobre la clase Biblioteca. El primero es el método público llamado obtenerGenerosAfines, el cual el usuario le pasa por parámetro el género inicial y éste define las estructuras auxiliares y hace uso de otros dos métodos privados (buscarVinculacionCerrada y generarGrafo) para retornar el grafo solución.

Una vez que el primer método llama a buscarVinculacionCerrada, comienza la recursividad de Backtracking con el fin de encontrar el ciclo. Cada vez que se entra en la recursividad lo primero que se hace es agregar el género con el cual se entró al ArrayList<String> auxiliar. Luego se chequea si el género no está visitado. En caso de estar visitado es porque se encontró con una vinculación cerrada, por lo que se procede a verificar si el género por el cual no se entró nuevamente en la recursividad efectivamente es el género inicial. De serlo, el ArrayList<String> auxiliar se copia al ArrayList<String> recorrido (el cual representa el ciclo mayor hallado hasta el momento) solo en el caso de que el tamaño del ArrayList<String> auxiliar sea mayor que del ArrayList<String> recorrido. Es decir, que el ciclo encontrado tenga mayor cantidad de géneros que el que se encuentra en el ArrayList<String> recorrido. En caso de que el género por el cual no se entró a la recursividad no sea el género inicial se remueve el género del ArrayList auxiliar y termina ese llamado a la función recursiva. Si el género no se encuentra visitado, entonces se marca con amarillo en el HashMap y se hace un llamado recursivo por cada uno de los vértices (géneros) adyacentes a él.

Una vez que se encuentra el ciclo más grande es necesario reconstruir el grafo. Para esto se implementó el último método generarGrafo, el cual recibe el ArrayList<String> recorrido y retorna el grafo reconstruido. Para ésto lo que se hace es repetir la manera en la que se genera el grafo el cual almacena todos los géneros.

```java
//SERVICIO 3

public GrafoDirigido obtenerGenerosAfines(String genero) {
    if (this.generos.contieneVertice(genero)) { //si contiene el genero

        HashMap<String, String> colores = new HashMap<>();
        ArrayList<String> recorrido = new ArrayList<>();
        ArrayList<String> aux = new ArrayList<>();

        Iterator<String> vertices = this.generos.obtenerVertices(); //Obtenemos vertices del grafo generos
        while (vertices.hasNext()) {
            String g = vertices.next();
            colores.put(g, "blanco");
        }

        this.buscarVinculacionCerrada(genero, aux, colores, recorrido);
        return this.generarGrafo(recorrido);

    }

    return null;
}
```

Método público que resuelve el Servicio III sobre la clase Biblioteca.

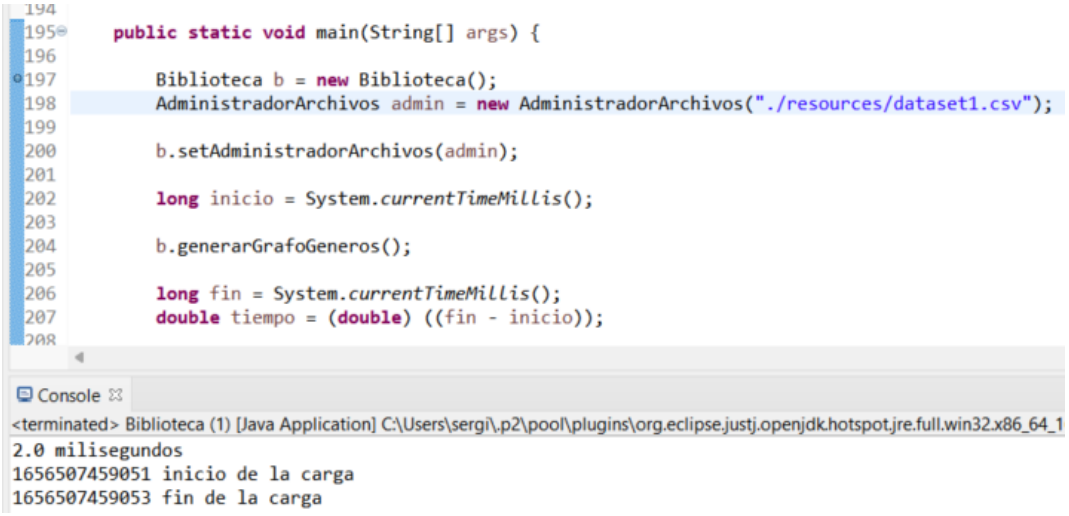
```java
private void buscarVinculacionCerrada(String genero, ArrayList<String> aux, HashMap<String, String> colores, ArrayList<String> recorrido) {
    aux.add(genero);
    if (colores.get(genero).equals("blanco")) { //si no esta visitado
        colores.put(genero, "amarillo");
        Iterator<String> adyacentes = this.generos.obtenerAdyacentes(genero);
        while (adyacentes.hasNext()) {
            String vertice = adyacentes.next();
            this.buscarVinculacionCerrada(vertice, aux, colores, recorrido);
        }

        colores.put(genero, "negro");
    }
    if (genero.equals(aux.get(0))) {
        if (recorrido.isEmpty()) {
            recorrido.addAll(aux);
        } else if (recorrido.size() < aux.size())  {
            recorrido.clear();
            recorrido.addAll(aux);
        }
    }
    aux.remove(aux.size() - 1);
}

private GrafoDirigido generarGrafo(ArrayList<String> recorrido) {
    GrafoDirigido grafo = new GrafoDirigido();
    for (int i = 0; i < recorrido.size(); i++) {
        if (i < recorrido.size() - 1) {
            grafo.agregarVertice(recorrido.get(i));
        }
        if (i > 0) {
            Integer etiqueta = this.generos.obtenerArco(recorrido.get(i - 1), recorrido.get(i)).getEtiqueta();
            grafo.agregarArco(recorrido.get(i - 1), recorrido.get(i), etiqueta);
        }
    }
    return grafo;
}
```

Métodos privados que resuelven el Servicio III sobre la clase Biblioteca.

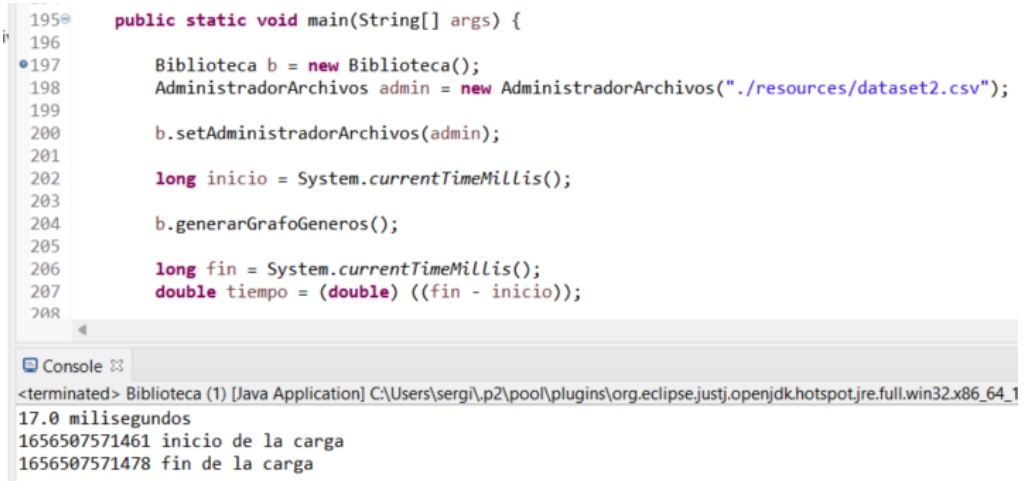
## Pruebas de ejecución

Una vez implementados los tres servicios solicitados se llevaron a cabo pruebas de ejecución con los distintos archivos .cvs.

A continuación se pueden observar las distintas ejecuciones de la carga de géneros en los diferentes dataset en la cual se genera el grafo de almacenamiento. Es notable cómo a medida que la cantidad de líneas sobre el csv crece el tiempo de ejecución también lo hace.




Tiempo de generación del grafo dataset1.




Tiempo de generación del grafo dataset2.

```
195    public static void main(String[] args) {
196
197        Biblioteca b = new Biblioteca();
198        AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset3.csv");
199
200        b.setAdministradorArchivos(admin);
201
202        long inicio = System.currentTimeMillis();
203
204        b.generarGrafoGeneros();
205
206        long fin = System.currentTimeMillis();
207        double tiempo = (double) ((fin - inicio));
```

Console

```
<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
285.0 milisegundos
1656507593692 inicio de la carga
1656507593977 fin de la carga
```

Tiempo de generación del grafo dataset3.

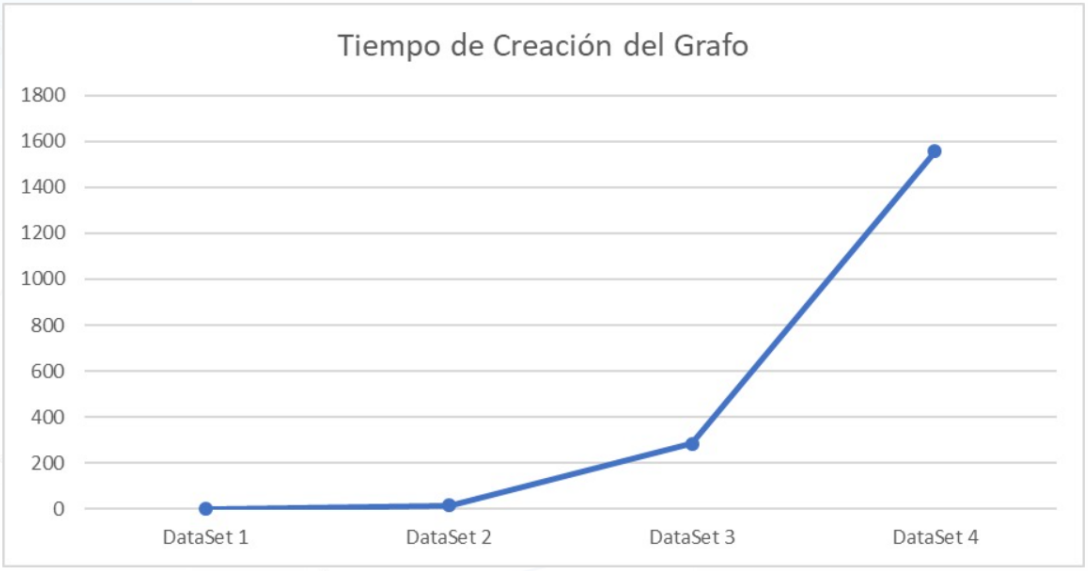
```
195    public static void main(String[] args) {
196
197        Biblioteca b = new Biblioteca();
198        AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset4.csv");
199
200        b.setAdministradorArchivos(admin);
201
202        long inicio = System.currentTimeMillis();
203
204        b.generarGrafoGeneros();
205
206        long fin = System.currentTimeMillis();
207        double tiempo = (double) ((fin - inicio));
```

Console

```
<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
1557.0 milisegundos
1656507627185 inicio de la carga
1656507628742 fin de la carga
```

Tiempo de generación del grafo dataset4.

En la primera imagen se evidencia como con el archivo dataset1, el cual tiene menor cantidad de géneros y relaciones entre los mismos, es el que menor tiempo tarda en generar el grafo. En la última imagen se evidencia nuevamente la relación entre el tamaño del csv y el tiempo de generación del grafo.




Tiempo de creación del grafo en los distintos datasets.

También se hizo un análisis del tiempo de ejecución de cada uno de los servicios con los distintos archivos csv.

```
195    public static void main(String[] args) {
196
197        Biblioteca b = new Biblioteca();
198        AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset1.csv");
199
200        b.setAdministradorArchivos(admin);
201
202
203        b.generarGrafoGeneros();
204
205        long inicio = System.currentTimeMillis();
206
207        System.out.println(b.generosMasBuscados(3, "servicios"));
208
209        long fin = System.currentTimeMillis();
210        double tiempo = (double) ((fin - inicio));
211
212        System.out.println(tiempo + " milisegundos");
213        System.out.println(inicio + " inicio de la carga");
214        System.out.println(fin + " fin de la carga");
```

```
Console
<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
[periodismo, tecnología, negocios]
0.0 milisegundos
1656509305593 inicio de la carga
1656509305593 fin de la carga
```

Primer Servicio para N = 3 y género = “servicios” para el dataset1.

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

    Biblioteca b = new Biblioteca();
    AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset2.csv");

    b.setAdministradorArchivos(admin);


    b.generarGrafoGeneros();

    long inicio = System.currentTimeMillis();

    System.out.println(b.generosMasBuscados(3, "servicios"));

    long fin = System.currentTimeMillis();
    double tiempo = (double) ((fin - inicio));

    System.out.println(tiempo + " milisegundos");
    System.out.println(inicio + " inicio de la carga");
    System.out.println(fin + " fin de la carga");
```

Console

<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_

```
[economía, educación, tecnología]
0.0 milisegundos
1656509249708 inicio de la carga
1656509249708 fin de la carga
```

Primer servicio para N = 3 y género = “servicios” usando dataset2.

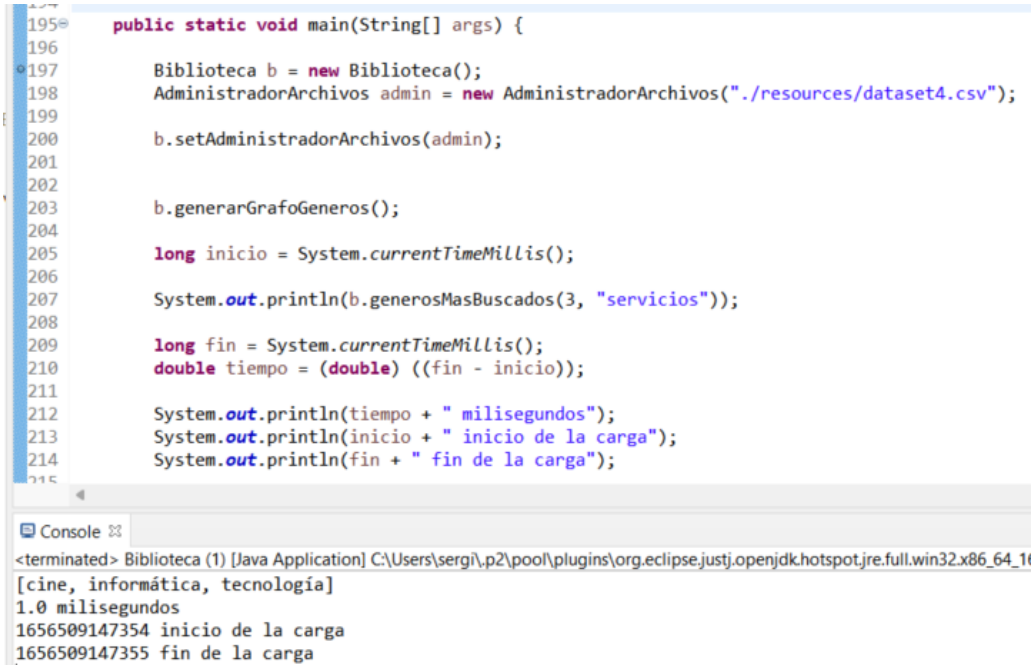
```java
public static void main(String[] args) {

    Biblioteca b = new Biblioteca();
    AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset3.csv");

    b.setAdministradorArchivos(admin);


    b.generarGrafoGeneros();

    long inicio = System.currentTimeMillis();

    System.out.println(b.generosMasBuscados(3, "servicios"));

    long fin = System.currentTimeMillis();
    double tiempo = (double) ((fin - inicio));

    System.out.println(tiempo + " milisegundos");
    System.out.println(inicio + " inicio de la carga");
    System.out.println(fin + " fin de la carga");
```

Console

<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1

```
[humor, religión, deportes]
1.0 milisegundos
1656509211847 inicio de la carga
1656509211848 fin de la carga
```

Primer Servicio para N = 3 y género = “servicios” para el dataset3.



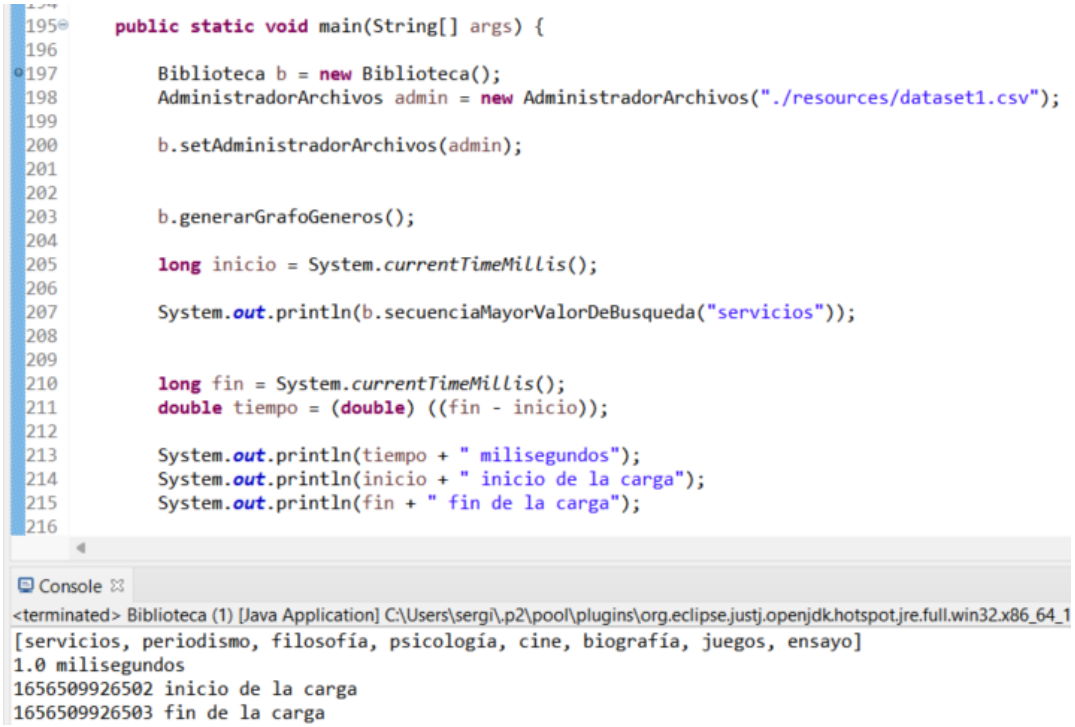
```
    public static void main(String[] args) {

        Biblioteca b = new Biblioteca();
        AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset4.csv");

        b.setAdministradorArchivos(admin);


        b.generarGrafoGeneros();

        long inicio = System.currentTimeMillis();

        System.out.println(b.generosMasBuscados(3, "servicios"));

        long fin = System.currentTimeMillis();
        double tiempo = (double) ((fin - inicio));

        System.out.println(tiempo + " milisegundos");
        System.out.println(inicio + " inicio de la carga");
        System.out.println(fin + " fin de la carga");
```

```
Console
<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1
[cine, informática, tecnología]
1.0 milisegundos
1656509147354 inicio de la carga
1656509147355 fin de la carga
```

Primer Servicio para N = 3 y género = “servicios” para el dataset4.

A partir de las cuatro imágenes, en las que únicamente se contabiliza el tiempo de ejecución del servicio I se ve muy poca diferencia de tiempo entre cada una de las búsquedas a pesar de las diferentes dimensiones de los grafos. En los primeros dos casos la búsqueda es tan rápida que devuelve como tiempo de búsqueda 0 milisegundos.

Para el segundo servicio también se hicieron pruebas de ejecución con los cuatro archivos utilizando como género inicial “servicios”.



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

        Biblioteca b = new Biblioteca();
        AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset1.csv");

        b.setAdministradorArchivos(admin);


        b.generarGrafoGeneros();

        long inicio = System.currentTimeMillis();

        System.out.println(b.secuenciaMayorValorDeBusqueda("servicios"));


        long fin = System.currentTimeMillis();
        double tiempo = (double) ((fin - inicio));

        System.out.println(tiempo + " milisegundos");
        System.out.println(inicio + " inicio de la carga");
        System.out.println(fin + " fin de la carga");
```

```
Console
<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1
[servicios, periodismo, filosofía, psicología, cine, biografía, juegos, ensayo]
1.0 milisegundos
1656509926502 inicio de la carga
1656509926503 fin de la carga
```

Segundo servicio para género = “servicios” dataset1.

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

    Biblioteca b = new Biblioteca();
    AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset2.csv");

    b.setAdministradorArchivos(admin);


    b.generarGrafoGeneros();

    long inicio = System.currentTimeMillis();

    System.out.println(b.secuenciaMayorValorDeBusqueda("servicios"));


    long fin = System.currentTimeMillis();
    double tiempo = (double) ((fin - inicio));

    System.out.println(tiempo + " milisegundos");
    System.out.println(inicio + " inicio de la carga");
    System.out.println(fin + " fin de la carga");
```

```
Console
<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin\jav
[servicios, economía, poesía, infantil, cine, drama, ficción, thriller, arte, ensayo, ciencia, negocios, romance, relatos]
1.0 milisegundos
1656510039340 inicio de la carga
1656510039341 fin de la carga
```

Segundo servicio para género = “servicios” dataset 2.

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

    Biblioteca b = new Biblioteca();
    AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset3.csv");

    b.setAdministradorArchivos(admin);


    b.generarGrafoGeneros();

    long inicio = System.currentTimeMillis();

    System.out.println(b.secuenciaMayorValorDeBusqueda("servicios"));


    long fin = System.currentTimeMillis();
    double tiempo = (double) ((fin - inicio));

    System.out.println(tiempo + " milisegundos");
    System.out.println(inicio + " inicio de la carga");
    System.out.println(fin + " fin de la carga");
```

```
Console
<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
[servicios, humor, música, marketing, drama]
1.0 milisegundos
1656510139361 inicio de la carga
1656510139362 fin de la carga
```

Segundo servicio para género = “servicios” dataset3.

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

    Biblioteca b = new Biblioteca();
    AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset4.csv");

    b.setAdministradorArchivos(admin);


    b.generarGrafoGeneros();

    long inicio = System.currentTimeMillis();

    System.out.println(b.secuenciaMayorValorDeBusqueda("servicios"));


    long fin = System.currentTimeMillis();
    double tiempo = (double) ((fin - inicio));

    System.out.println(tiempo + " milisegundos");
    System.out.println(inicio + " inicio de la carga");
    System.out.println(fin + " fin de la carga");

```

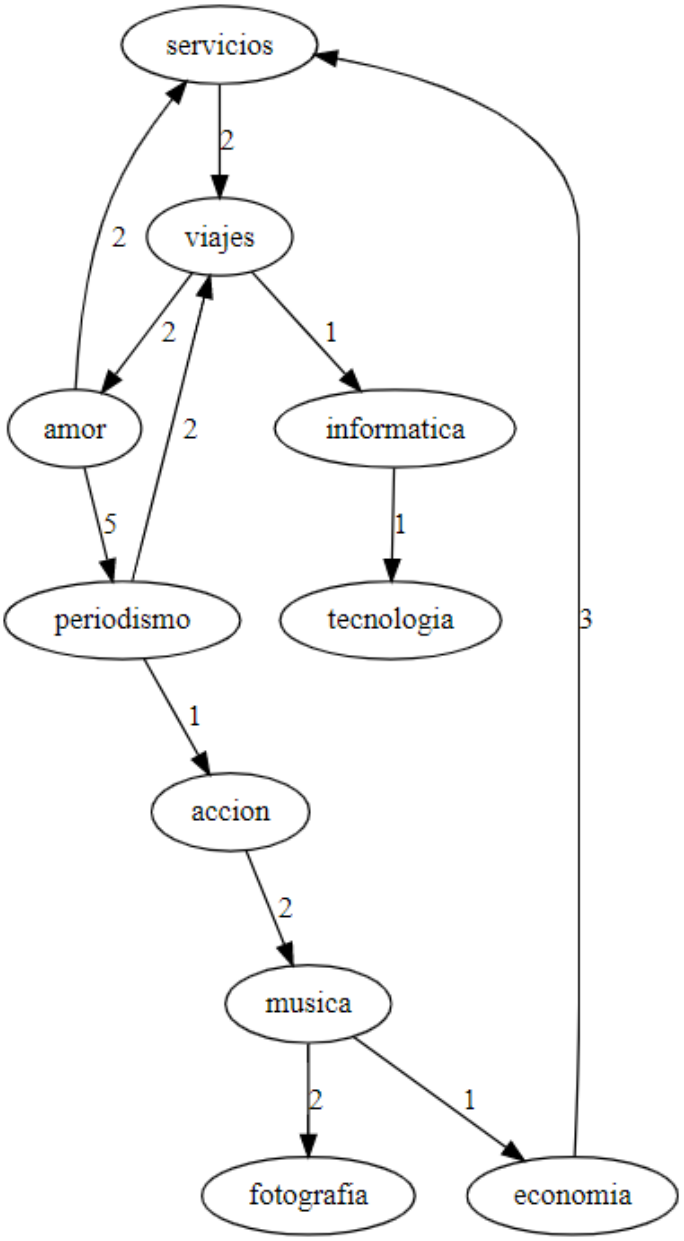
Console

<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.

```
[servicios, cine, ciencia, política, romance, gastronomía]
1.0 milisegundos
1656510167549 inicio de la carga
1656510167550 fin de la carga
```

Segundo servicio para género = “servicios” dataset4.

Durante las pruebas de este servicio surgieron preguntas debido a que los tiempos de ejecución eran muy similares a pesar de la diferencia de datos entre los distintos dataset. Para corroborar que se estuviera generando correctamente la secuencia con mayor valor de búsqueda para los dataset 3, 4 y 5 era imposible debido a la cantidad de datos. Por lo tanto se decidió corroborar lo hecho mediante el archivo dataset5 generado por el propio grupo de trabajo. Este archivo contiene pocos géneros pero varias relaciones entre ellos. También se uso la herramienta brindada por la cátedra para generar el grafo. Como resultado de este nuevo archivo se obtuvo el siguiente grafo.




Una vez generado este grafo, lo cual sirvió también para chequear que la estructura de almacenamiento se estuviera generando como debía, se procedió a calcular a mano la secuencia con mayor valor de búsqueda para el género “servicios”.

Dado que se había definido que la secuencia no debía tener géneros repetidos y si una vez que tomaba una decisión se encontraba con un género el cual ya se encontraba en ArrayList solución, entonces debía terminar con la ejecución. En base a esto como resultado del seguimiento del grafo se debía obtener el ArrayList [servicios, viajes, amor, periodismo] y así fue.

```
194
195    public static void main(String[] args) {
196
197        Biblioteca b = new Biblioteca();
198        AdministradorArchivos admin = new AdministradorArchivos("./resources/dataset5.csv");
199
200        b.setAdministradorArchivos(admin);
201
202
203        b.generarGrafoGeneros();
204
205        long inicio = System.currentTimeMillis();
206
207        System.out.println(b.secuenciaMayorValorDeBusqueda("servicios"));
208
209
210        long fin = System.currentTimeMillis();
211        double tiempo = (double) ((fin - inicio));
212
213        System.out.println(tiempo + " milisegundos");
214        System.out.println(inicio + " inicio de la carga");
215        System.out.println(fin + " fin de la carga");
```

Console

<terminated> Biblioteca (1) [Java Application] C:\Users\sergi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1

```
[servicios, viajes, amor, periodismo]
1.0 milisegundos
1656511162487 inicio de la carga
1656511162488 fin de la carga
```

Servicio II para el dataset 5 y género = “servicios”.

```
Generos
amor,servicios
amor,servicios
economia,servicios,viajes,informatica,tecnologia
economia,servicios,viajes,amor
economia,servicios
amor,periodismo,accion,musica,fotografia
musica,economia
accion,musica,fotografia
periodismo,viajes,amor,periodismo
amor,periodismo
amor,periodismo
amor,periodismo,viajes
```

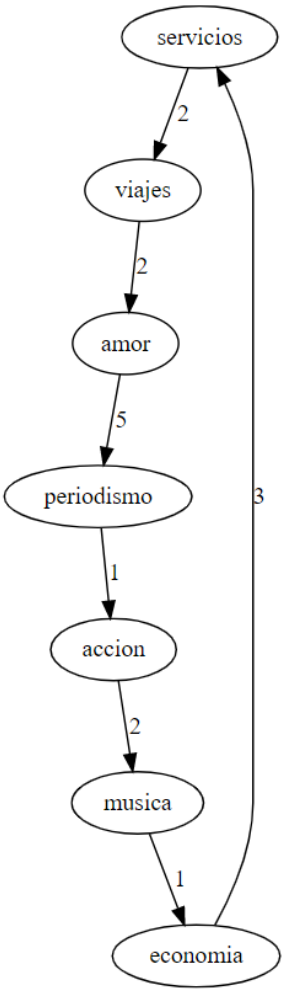
Archivo dataset5.

En base a estos resultados y el debug hecho se determinó que los tiempos de las secuencia de búsqueda en los dataset y el ArrayList resultado es muy pequeño (a comparación del contenido del archivo) dado que estos mismos contienen una gran cantidad de relaciones entre los vértices lo cual genera rápidamente un ciclo.

Por último se hicieron pruebas para el Servicio III. Para este punto también se usó el género “servicios” con los distintos dataset. En el caso del primer dataset, obtener la mayor vinculación cerrada tardó 0 milisegundos. Para el dataset2,

dataset3 y dataset4 la ejecución del servició dos llevó 2 milisegundos. Nuevamente a pesar de que la cantidad de datos en los archivos aumenta, el tiempo de ejecución del servicio III se mantuvo. En los diferentes grafos obtenidos como resultado de cada una de las ejecuciones no varía mucho la cantidad de vértices que tiene cada uno de ellos. De aquí la poca diferencia en el tiempo de ejecución.

Para este punto también se hicieron pruebas con la herramienta que genera el grafo de manera gráfica y se utilizó nuevamente el dataset5. Se sabía que el tiempo de ejecución no iba a ser notable pero si iba a permitir corroborar los resultados. Dado el mismo grafo de la página 16, se buscó entonces la mayor vinculación cerrada. El resultado obtenido mediante la ejecución del servicio III y la herramienta gráfica se corresponden con un resultado correcto.




Ciclo para el dataset5 y género = “servicios”.

# Conclusiones

En base a todo lo analizado durante el desarrollo de la primera y segunda instancia del TPE se puede ver una correcta elección de las estructuras de almacenamiento disminuye el tiempo de ejecución de distintos servicios a pesar de la gran cantidad de datos. Al inicio del trabajo se creía que si el archivo tenía una mayor cantidad de datos, entonces el tiempo de ejecución de las pruebas también debía aumentar. Sin embargo, esta suposición sólo se vio reflejada en la creación del grafo. Luego el tiempo de ejecución pasó a depender de la implementación del problema en sí y lo que se buscaba.

Hay que destacar la necesidad y la facilidad que brindan herramientas gráficas a la hora de corroborar las soluciones. A pesar de que no con todos los archivos se podía chequear el grafo obtenido, teniendo la posibilidad de generar un archivo propio permitía dar una mayor confianza al código implementado.