Trabajo práctico especial.

Entrega Nº 2.

Programación 3 - TUDAI 2022
UNICEN

Fecha de entrega 01/07/2022

Alumno: Farinelli, Matías E.
matfarinelli@gmail.com

https://github.com/matfarinelli/TP-Especial-Prog-III-p2

Contenido

Introducción:

A lo largo del presente informe se pretende desarrollar la solución propuesta por el alumno y su implementación, a una problemática planteada por la cátedra. Se analizará cada servicio solicitado, buscando la optimización computacional aplicando los conocimientos vistos durante la cursada.

Problemática planteada:

En esta última etapa del trabajo se desea realizar un análisis de la utilización del buscador, por parte de los usuarios; puntualmente la relación entre los géneros ingresados en las distintas búsquedas. Se asume que la herramienta permite ingresar un conjunto de categorías a buscar, con lo cual la colección de libros resultantes contendrá sólo los libros que cumplen con todas las categorías ingresadas.

La cátedra proveerá como entrada al programa varios archivos .csv con los sucesivos géneros que se ingresaron en distintas búsquedas realizadas. A partir de esta información se desea construir un grafo, donde:

● Cada vértice representa un género que fue incluido en alguna búsqueda; y

● Un arco que comunica dos vértices indicará la cantidad de veces que luego de buscar el primer género inmediatamente a continuación se buscó por el segundo género.

Utilizando este grafo como estructura se deberán implementar los siguientes 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. Vamos a definir 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 consiguió una vinculación cerrada entre uno o más géneros que permitió volver al género de inicio.

Desarrollo:

Para dar solución a la problemática planteada se utilizó un Grafo Dirigido que implementa la interfaz Grafo, la cual nos provee de diferentes funcionalidades.

Este grafo se construye dinámicamente con la lectura de un dataset provisto por la cátedra. Dicho dataset contiene una cantidad de búsquedas secuenciales realizadas por un usuario, en el cual se comenzó buscando por “x” género y se finalizó en algún otro.

Una vez construido el grafo, podemos analizar las relaciones entre cada género, permitiendo la vinculación sucesiva o afinidad con otros géneros del dataset. Mediante los servicios implementados, podemos encontrar patrones y comportamientos de los usuarios en las búsquedas.

Clases involucradas:

1- “CSVReader”, su función principal es la lectura del dataset y fraccionamiento de las búsquedas para generar dinámicamente el grafo.
2- “Grafo Dirigido”, es la estructura que representa las relaciones entre los distintos géneros del dataset.
3- “Vértice”, son los géneros del grafo.
4- “Arco”, representa la relación entre los géneros. Tiene una etiqueta o ponderación que representa la cantidad de veces que esa vinculación entre géneros se repite en el análisis del dataset.
5- “Main”, se utilizó para probar los métodos realizados.
6- “Grafo”, interfaz que provee funcionalidades al grafo dirigido y no dirigido.

Implementación:

En primer lugar, analizaremos la clase “CSVReader” quien es la encargada de procesar los datasets recibidos y mediante invocaciones a las clases pertinentes generar dinámicamente el grafo a través de su procesamiento. Para esto, al recibir el dataset, fracciona mediante el método “split” los géneros de la búsqueda (cada línea del dataset es una búsqueda), agregando el vértice y generando los arcos entre sí y el inmediato posterior.

```
// para recorrer arreglo de la linea de busqueda
int i = 0;
while (i < generos.length) {

    grafo.agregarVertice(generos[i]);

    if (i + 1 < generos.length) {

        if (grafo.existeArco(generos[i], generos[i + 1])) {
            grafo.obtenerArco(generos[i], generos[i + 1]).incrementarValor();
        } else {
            grafo.agregarVertice(generos[i + 1]);
            grafo.agregarArco(generos[i], generos[i + 1]);
        }
    }
```

Con respecto a las estructuras elegidas, los vértices o géneros del Grafo se almacenan en un HashMap<String, Vertice>, la razón es que no se debe controlar la repetición de los vértices por la funcionalidad provista por la estructura. Solo se guardan géneros no repetidos.

```
public class GrafoDirigido<T> implements Grafo<T> {

    private HashMap<String, Vertice<T>> vertices;

    public GrafoDirigido() {
        this.vertices = new HashMap<String, Vertice<T>>();
    }
```

Posteriormente, el Grafo controla si existe el arco entre el género actual y el siguiente género del arreglo examinado, si existe si incrementa el peso o etiqueta del arco, lo cual será de vital importancia para la implementación de servicios posteriores. En caso que no exista, se agrega dicho arco al grafo.

En la implementación propuesta, son los vértices quien conoce y administra los arcos hacia sus adyacentes y no el Grafo. El Grafo solo contiene vértices. El método agregarArco, le indican al género en cuestión que establezca un arco entre el mismo, y otro género indicado por parámetro. El vértice guarda los arcos en un HashMap por el mismo motivo que el grafo, evitar repeticiones y la búsqueda de optimización computacional.
Las complejidades de los métodos indicados son O(1) tanto para añadir, quitar o consultar el HashMap, ya sea de vértice o de arcos. Vértices en el grafo, arcos en el vértice.

Servicios:

- **Obtener los N géneros más buscados luego de buscar por el género A.:**

Para resolver dicha problemática se implementó un método que retorne un ArrayList<String> con los géneros afines a un género buscado. Se considera afinidad, cuando un usuario inmediatamente después de buscar un género continúa a otro, y esa tendencia se repite a lo largo del dataset estudiado. La tendencia la podemos extraer del peso de los arcos.

La complejidad del problema fue retornar los N géneros ordenados de mayor a menor, tomando como referencia el peso de sus arcos y debiendo recurrir a un ordenamiento.

En primer lugar, se solicita al género buscado todos sus arcos y mediante un iterador, se va guardando Arco por Arco en un ArrayList<Arco>. Cada arco tiene un género adyacente como destino y un peso, que representa las repeticiones de esa secuencia en el conjunto de datos.

Seguidamente, dicho arrayList es ordenado de mayor a menor. Esto se pudo lograr mediante la interfaz Comparator y Collections.sort, logrando una complejidad de O(n log n) según documentación oficial del método donde n es el número total de elementos en la matriz.

Como el servicio solicita que se retorne “N” cantidad de géneros, el método instancia un nuevo arrayList<String> “solución”. En este último array se almacenan “N ” cantidad arco mediante un “for”.

```java
public ArrayList<String> obtenerGenerosAfines(String generoBuscado, int n) {
    ArrayList<Arco> arcos = new ArrayList<>();

    Timer timer = new Timer();
    System.out.println("Tiempo: ");

    Iterator<Arco<T>> it = this.obtenerArcos(generoBuscado);
    while (it.hasNext()) {
        Arco arco = it.next();
        arcos.add(arco);
    }

    Collections.sort(arcos, new Comparator<Arco>() {
        @Override
        public int compare(Arco arco1, Arco arco2) {
            if (arco1.getValor() < arco2.getValor()) {
                return 1;
            } else if (arco1.getValor() > arco2.getValor()) {
                return -1;
            } else
                return 1;
        }
    });

    ArrayList<String> generosAfines = new ArrayList<>();

    int lenght = 0;

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

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

    System.out.println(timer.stop());
    return generosAfines;

}
```

Análisis desempeño algoritmo:

Se realizaron pruebas en 4 datasets con 20, 1.000, 100.000 y 1.000.000 de búsquedas respectivamente.

El rendimiento se mantuvo constante a pesar del diferente tamaño de las muestras analizadas indicando eficiencia en la implementación del método realizado.

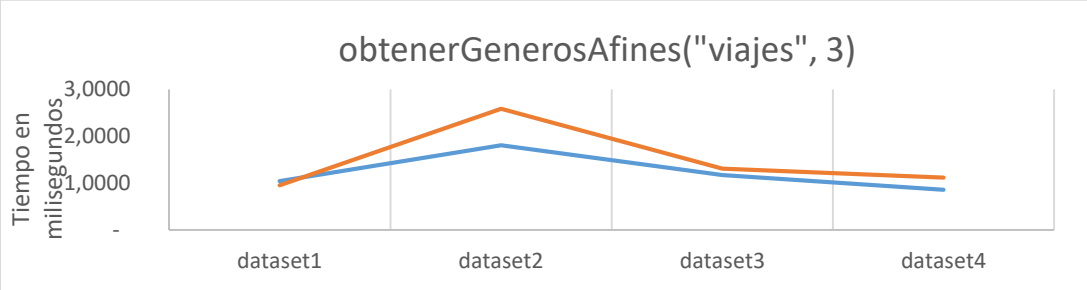
```
// pruebas
GrafoDirigido<String> grafo_d = new GrafoDirigido<String>();

CSVReader reader = new CSVReader(csvFile: "./dataset4.csv");
reader.cargarGrafo(grafo_d);

// servicio 1

System.out.println(grafo_d.obtenerGenerosAfines(generoBuscado: "viajes", n: 3));
```

obtenerGenerosAfines("viajes", 3)	dataset1	dataset2	dataset3	dataset4
	1,126527003	1,126537454	1,126545585	1,126556882



	Resultados obtenidos:
dataset1	marketing, humor, informática
dataset2	fotografía, relatos, policial
dataset3	música, investigación, leyendas
dataset4	servicios, humor, religión

- **A partir de un género A encontrar, en tiempo polinomial, la secuencia de géneros que más alto valor de búsqueda posee. Vamos a definir el valor de búsqueda de una secuencia como la suma de los arcos que la componen.**

Para resolver este problema se implementó un algoritmo Greedy, el cual por definición se utiliza para maximizar o minimizar una solución. Si bien no siempre llega a la mejor solución, puede otorgarnos una aproximación cercana a la mejor solución y que cumpla con el tiempo polinomial requerido. Un algoritmo backtracking podría otorgarnos la respuesta correcta para este enunciado, pero la complejidad de tal algoritmo, no cumple con el enunciado.

Respecto a la implementación del método, el universo o conjunto de candidatos eran todos los géneros disponibles que forman parte del grafo.

En primer lugar, se solicita los géneros adyacentes al género origen o inicial y se incorpora este género origen, al conjunto solución y se lo remueve del universo de candidatos.

```java
public ArrayList<String> obtenerMayorSecuenciaGeneros(String generoOrigen) {

    ArrayList<String> solucion = new ArrayList<>();
    ArrayList<String> generosCandidatos = new ArrayList<>(); // candidatos
    // // HashMap<String, Boolean> visitados = new HashMap<>();
    // ArrayList<String> visitados = new ArrayList<>();
    int sumaArcos = 0;

    Iterator<String> it = this.obtenerVertices(); // todos los generos
    while (it.hasNext()) {
        String genero = it.next();
        generosCandidatos.add(genero);
    }

    // situación que evita NULL point exception - por si el genero no existe
    if (generosCandidatos.contains(generoOrigen)) {

        while (!generosCandidatos.isEmpty() && generoOrigen != null) {
            // visitados.add(generoOrigen);
            solucion.add(generoOrigen);

            generosCandidatos.remove(generoOrigen);

            // para ver el recorrido entre arco y arco
            // System.out.println("Lista solucion:" + solucion.toString() + " " +
            // solucion.size());
            // System.out.println("Suma de pesos: " + sumaArcos);

            // si devuelve null, se acaba el ciclo
            Arco arcoAdyMayor = this.seleccionarArcoMayorPeso(generoOrigen, generosCandidatos);

            sumaArcos += arcoAdyMayor.getValor();

            generoOrigen = arcoAdyMayor.getVerticeDestino();

        }
    }

    return solucion;
}
```

La estrategia de selección propia del algoritmo Greedy, recibe los géneros candidatos y seleccionará el arco con mayor etiqueta. La condición es que esté género, debe ser adyacente al género origen.

Con el arco de mayor etiqueta seleccionado, contamos con la información de su género destino y el tamaño de su etiqueta. El género destino, ahora será el nuevo género origen y pasará a formar parte del conjunto solución y dejará de pertenecer al conjunto de candidatos.

La sumatoria de las etiquetas, carece de valor si no se compara con la sumatoria de otras secuencias posibles. Se lo consideró en la implementación para darle valor numérico a la secuencia retornada. En un algoritmo backtracking, nos hubiese permitido comparar las posibles soluciones e ir seleccionando siempre la mayor.

```java
CSVReader reader = new CSVReader(csvFile: "./dataset2.csv");
reader.cargarGrafo(grafo_d);
grafo_d.obtenerMayorSecuenciaGeneros("deportes");
```

```java
public Arco seleccionarArcoMayorPeso(String generoOrigen, ArrayList<String> generosCandidatos) {
    Iterator<Arco<T>> it = this.obtenerArcos(generoOrigen);
    Arco arcoMayor = new Arco(verticeOrigen: null, verticeDestino: null, valor: 0); // al estar los arcos ordenados
    // sido recorrido por otro genero anteriormente

    while (it.hasNext()) {
        Arco arco = it.next();

        if (arco.getValor() > arcoMayor.getValor() && generosCandidatos.contains(arco.getVerticeDestino())) {
            arcoMayor = arco;
        }
    }

    // System.out.println("Origen " + generoOrigen + " / arco mayor: " +
    //       arcoMayor.getVerticeDestino() + " = " + arcoMayor.getValor());

    return arcoMayor;
}
```

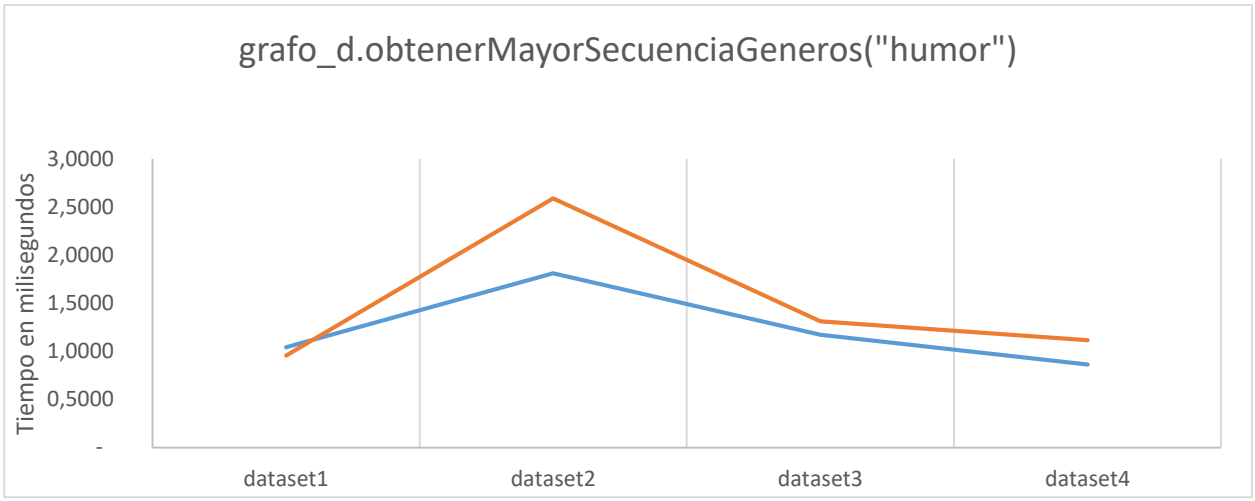
```
\TP\ Especial\ 2022\ -\ p2\\bin Main
Origen deportes / arco mayor: ensayo = 6
Origen ensayo / arco mayor: ciencia = 6
Origen ciencia / arco mayor: informÃ¡tica = 5
Origen informÃ¡tica / arco mayor: relatos = 6
Origen relatos / arco mayor: servicios = 6
Origen servicios / arco mayor: tecnologÃ a = 4
Origen tecnologÃ a / arco mayor: moda = 5
Origen moda / arco mayor: viajes = 4
Origen viajes / arco mayor: fotografÃ a = 5
Origen fotografÃ a / arco mayor: economÃ a = 7
Origen economÃ a / arco mayor: gastronomÃ a = 5
Origen gastronomÃ a / arco mayor: religiÃ³n = 5
Origen religiÃ³n / arco mayor: thriller = 4
Origen thriller / arco mayor: arte = 4
Origen arte / arco mayor: cultura = 5
Origen cultura / arco mayor: periodismo = 5
Origen periodismo / arco mayor: investigaciÃ³n = 4
Origen investigaciÃ³n / arco mayor: psicologÃ a = 6
Origen psicologÃ a / arco mayor: ficciÃ³n = 7
Origen ficciÃ³n / arco mayor: infantil = 6
Origen infantil / arco mayor: cine = 4
Origen cine / arco mayor: drama = 9
Origen drama / arco mayor: romance = 4
Origen romance / arco mayor: poesÃ a = 4
Origen poesÃ a / arco mayor: humor = 5
Origen humor / arco mayor: filosofÃ a = 5
Origen filosofÃ a / arco mayor: juegos = 5
Origen juegos / arco mayor: polÃ tica = 5
Origen polÃ tica / arco mayor: policial = 4
Origen policial / arco mayor: negocios = 3
Origen negocios / arco mayor: fantasÃ a = 2
Origen fantasÃ a / arco mayor: leyendas = 6
Origen leyendas / arco mayor: historia = 4
Origen historia / arco mayor: novela = 3
Origen novela / arco mayor: marketing = 5
Origen marketing / arco mayor: mÃºsica = 2
Origen mÃºsica / arco mayor: biografÃ a = 3
Origen biografÃ a / arco mayor: terror = 3
Origen terror / arco mayor: educaciÃ³n = 1
Origen educaciÃ³n / arco mayor: null = 0
```

```
Lista solucion:[deportes] 1
Suma de pesos: 0
Lista solucion:[deportes, ensayo] 2
Suma de pesos: 6
Lista solucion:[deportes, ensayo, ciencia] 3
Suma de pesos: 12
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica] 4
Suma de pesos: 17
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos] 5
Suma de pesos: 23
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios] 6
Suma de pesos: 29
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios, tecnologÃ a] 7
Suma de pesos: 33
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios, tecnologÃ a, moda] 8
Suma de pesos: 38
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios, tecnologÃ a, moda, viajes] 9
Suma de pesos: 42
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios, tecnologÃ a, moda, viajes, fotografÃ a] 10
Suma de pesos: 47
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios, tecnologÃ a, moda, viajes, fotografÃ a, economÃ a] 11
Suma de pesos: 54
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios, tecnologÃ a, moda, viajes, fotografÃ a, economÃ a, gastronomÃ a] 12
Suma de pesos: 59
Lista solucion:[deportes, ensayo, ciencia, informÃ¡tica, relatos, servicios, tecnologÃ a, moda, viajes, fotografÃ a, economÃ a, gastronomÃ a, religiÃ³n] 13
Suma de pesos: 64
```

Análisis desempeño algoritmo:

Se realizaron pruebas en 4 datasets con 20, 1.000, 100.000 y 1.000.000 de búsquedas respectivamente. 2 veces se realizaron las consultas por cada dataset.

grafo_d.obtenerMayorSecuenciaGeneros("humor")	dataset1	dataset2	dataset3	dataset4
	1,0422	1,8109	1,1733	0,8625
	0,9559	2,5870	1,3128	1,1165




```
Irma@USUARIO MINGW64 /e/Eclipe - Workspace/TP Especial 2022 - p2 (master)
$  cd e:\\Eclipe\ -\ Workspace\\TP\ Especial\ 2022\ -\ p2 ; /usr/bin/env C:\\Progr
am\ Files\\Java\\jdk1.8.0_111\\bin\\java.exe -cp E:\\Eclipe\ -\ Workspace\\TP\ Esp
ecial\ 2022\ -\ p2\\bin Main
Tiempo:
1.0422
[humor, informÃ¡tica, romance, ficciÃ³n, educaciÃ³n]

Irma@USUARIO MINGW64 /e/Eclipe - Workspace/TP Especial 2022 - p2 (master)
$  cd e:\\Eclipe\ -\ Workspace\\TP\ Especial\ 2022\ -\ p2 ; /usr/bin/env C:\\Progr
am\ Files\\Java\\jdk1.8.0_111\\bin\\java.exe -cp E:\\Eclipe\ -\ Workspace\\TP\ Esp
ecial\ 2022\ -\ p2\\bin Main
Tiempo:
1.8109
[humor, filosofÃ a, gastronomÃ a, religiÃ³n, deportes, ensayo, ciencia, informÃ¡ti
ca, relatos, servicios, tecnologÃ a, moda, viajes, fotografÃ a, economÃ a, poesÃ a
, infantil, arte, cultura, periodismo, investigaciÃ³n, psicologÃ a, ficciÃ³n, thri
ller, romance, mÃºsica, fantasÃ a, leyendas, historia, negocios, drama, polÃ tica,
 juegos, cine, novela, policial, biografÃ a, terror, marketing, educaciÃ³n]

Irma@USUARIO MINGW64 /e/Eclipe - Workspace/TP Especial 2022 - p2 (master)
$  cd e:\\Eclipe\ -\ Workspace\\TP\ Especial\ 2022\ -\ p2 ; /usr/bin/env C:\\Progr
am\ Files\\Java\\jdk1.8.0_111\\bin\\java.exe -cp E:\\Eclipe\ -\ Workspace\\TP\ Esp
ecial\ 2022\ -\ p2\\bin Main
Tiempo:
1.1733
[humor, mÃºsica, marketing, drama, servicios, religiÃ³n, leyendas, novela, thrille
r, poesÃ a, tecnologÃ a, infantil, ciencia, psicologÃ a, investigaciÃ³n, moda, neg
ocios, gastronomÃ a, fantasÃ a, educaciÃ³n, deportes, policial, periodismo, inform
Ã¡tica, biografÃ a, romance, juegos, relatos, fotografÃ a, economÃ a, polÃ tica, e
nsayo, cine, historia, arte, viajes, filosofÃ a, terror, ficciÃ³n, cultura]

Irma@USUARIO MINGW64 /e/Eclipe - Workspace/TP Especial 2022 - p2 (master)
$  cd e:\\Eclipe\ -\ Workspace\\TP\ Especial\ 2022\ -\ p2 ; /usr/bin/env C:\\Progr
am\ Files\\Java\\jdk1.8.0_111\\bin\\java.exe -cp E:\\Eclipe\ -\ Workspace\\TP\ Esp
ecial\ 2022\ -\ p2\\bin Main
Tiempo:
0.8625
[humor, leyendas, historia, cine, ciencia, polÃ tica, romance, gastronomÃ a, servi
cios, informÃ¡tica, psicologÃ a, moda, arte, negocios, terror, relatos, periodismo
, juegos, religiÃ³n, ensayo, marketing, drama, policial, ficciÃ³n, educaciÃ³n, inv
estigaciÃ³n, tecnologÃ a, deportes, thriller, fantasÃ a, biografÃ a, mÃºsica, infa
ntil, poesÃ a, viajes, cultura, filosofÃ a, economÃ a, fotografÃ a, novela]
```

DATASET 1

DATASET 2

DATASET 3

DATASET 4

El algoritmo mantiene buen funcionamiento. Al incrementar el tamaño de las muestras, el tiempo disminuye. Puede explicarse por la reducción del tamaño de posibles candidatos al seleccionar el arco de mayor tamaño. El costo computacional logrado es polinomial gracias al almoritmo Greedy.

- **Obtener el grafo únicamente con los géneros afines a un género A; es decir que, partiendo del género A, se consiguió una vinculación cerrada entre uno o más géneros que permitió volver al género de inicio.**

Para resolver la problemática se recurrió a un algoritmo recursivo de backtracking, de complejidad computacional alta y exponencial, ya que es la opción que permite conocer todas las alternativas posibles para el caso planteado.

Se recurre al método denominado “backtracking” el cual pretende recorrer todos los arcos de los géneros adyacentes al género origen, que mediante diversos arcos adyacentes, lo hagan retornar al mismo origen. Es decir, se debe recorrer todos los caminos posibles que lleguen al origen formando un ciclo. En cada ciclo encontrado, se generará un grafo.

```java
public GrafoDirigido<T> cicloGenerosAfines(String origen_destino) {
    GrafoDirigido<T> solucion = new GrafoDirigido<T>();
    HashSet<String> visitados = new HashSet<>(); // me permite que no se repitan los generos visitados

    solucion.agregarVertice(origen_destino);

    String actual = "Inicio";

    this.backtracking(origen_destino, actual, solucion, visitados);

    return solucion;
}
```

```java
private void backtracking(String origen_destino, String actual, GrafoDirigido<T> solucion,
        HashSet<String> visitados) {

    if (actual.equals(origen_destino)) { // situación corte, hay un ciclo
        // System.out.println("CICLO ENCONTRADO! por " + actual);
        // // System.out.println("Generos visitados: " + visitados);
        // System.out.println("-");

        // solo para control! - borrar!
        Iterator<Arco> it = solucion.obtenerTodoslosArcos();
        while (it.hasNext()) {
            Arco arco = it.next();
            arco.getDatos();
        }
        return;

    } else {
        // situación inicial, para evitar el if de corte
        if (actual.equals("Inicio")) {
            actual = origen_destino;
        }
        // Obtengo array de arcos del generoRecibido
        ArrayList<Arco<T>> arcosGenero = this.obtenerArcosArray(actual);

        // chequeo solo de revisar arcos no visitados
        for (Arco arco : arcosGenero) {
            // si no visite los arcos de ese genero actual u origen
            if (!visitados.contains(arco.getVerticeDestino())) {

                visitados.add(arco.getVerticeDestino());
                solucion.agregarVertice(arco.getVerticeDestino());
                solucion.agregarArco(actual, arco.getVerticeDestino());

                // System.out.println("ACTUAL: " + actual + " / " + visitados.toString());
                // System.out.println("-");
                this.backtracking(origen_destino, arco.getVerticeDestino(), solucion, visitados);

                solucion.borrarVertice(arco.getVerticeDestino());
                solucion.borrarArco(actual, arco.getVerticeDestino());
                visitados.remove(arco.getVerticeDestino());
            }
        }
        return;
    }
}
```

Se estableció como situación de corte cuando el género adyacente recibido sea igual al género que fue génesis de todo el proceso.

Mientras no haya corte, el algoritmo generará nuevos candidatos del género actual y recorrerá cada uno de sus arcos. Si ese arco no fue explorado, se lo añadirá al conjunto solución y se ejecutará nuevamente el backtracking para recorrer los arcos, de ese nuevo género no explorado.
Al finalizar, la solución se deshacerá, borrando el vértice y el arco agregado que no pudieron llegar a destino.

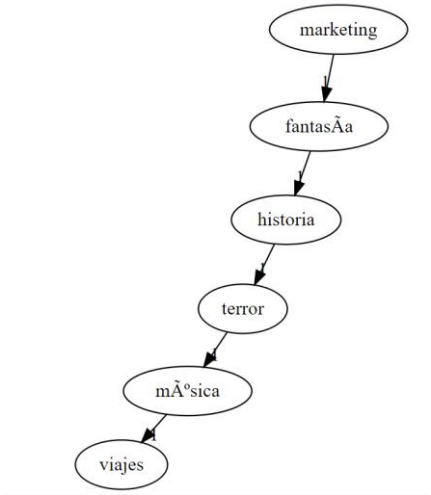
Análisis desempeño algoritmo:

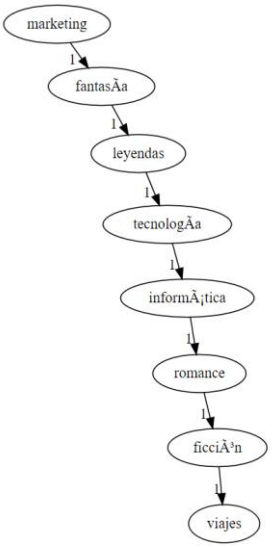
Solo pudo realizarse un análisis sobre dataset1, los siguientes datasets resultaron muy complejos de analizar computacionalmente y luego de varios minutos el algoritmo sigue sin retornar respuestas, producto del volumen exponencial de arcos.

Haciendo uso de la herramienta GraphvizOnline, se pudo realizar el control de los grafos propuestos como solución por el algoritmo.

```
-
poesÃ a -> thriller[  label = 1 ];
filosofÃ a -> psicologÃ a[  label = 1 ];
romance -> negocios[  label = 1 ];
historia -> terror[  label = 1 ];
terror -> mÃºsica[  label = 1 ];
mÃºsica -> viajes[  label = 1 ];
fantasÃ a -> leyendas[  label = 1 ];
thriller -> arte[  label = 1 ];
negocios -> historia[  label = 1 ];
arte -> romance[  label = 1 ];
marketing -> fantasÃ a[  label = 1 ];
tecnologÃ a -> periodismo[  label = 1 ];
psicologÃ a -> poesÃ a[  label = 1 ];
leyendas -> tecnologÃ a[  label = 1 ];
periodismo -> filosofÃ a[  label = 1 ];
-
poesÃ a -> drama[  label = 1 ];
filosofÃ a -> psicologÃ a[  label = 1 ];
fantasÃ a -> leyendas[  label = 1 ];
marketing -> fantasÃ a[  label = 1 ];
drama -> viajes[  label = 1 ];
cine -> poesÃ a[  label = 1 ];
tecnologÃ a -> periodismo[  label = 1 ];
psicologÃ a -> cine[  label = 1 ];
leyendas -> tecnologÃ a[  label = 1 ];
periodismo -> filosofÃ a[  label = 1 ];
-
```

```
historia -> terror[  label = 1 ];
terror -> mÃºsica[  label = 1 ];
mÃºsica -> viajes[  label = 1 ];
fantasÃ a -> historia[  label = 1 ];
marketing -> fantasÃ a[  label = 1 ];
-
romance -> ficciÃ³n[  label = 1 ];
ficciÃ³n -> viajes[  label = 1 ];
fantasÃ a -> leyendas[  label = 1 ];
informÃ¡tica -> romance[  label = 1 ];
marketing -> fantasÃ a[  label = 1 ];
tecnologÃ a -> informÃ¡tica[  label = 1 ];
leyendas -> tecnologÃ a[  label = 1 ];
-
investigaciÃ³n -> thriller[  label = 1 ];
poesÃ a -> drama[  label = 1 ];
romance -> negocios[  label = 1 ];
historia -> investigaciÃ³n[  label = 1 ];
fantasÃ a -> leyendas[  label = 1 ];
informÃ¡tica -> romance[  label = 1 ];
thriller -> juegos[  label = 1 ];
negocios -> historia[  label = 1 ];
juegos -> ensayo[  label = 1 ];
ensayo -> psicologÃ a[  label = 1 ];
marketing -> fantasÃ a[  label = 1 ];
drama -> viajes[  label = 1 ];
tecnologÃ a -> informÃ¡tica[  label = 1 ];
psicologÃ a -> poesÃ a[  label = 1 ];
leyendas -> tecnologÃ a[  label = 1 ];
```





Conclusión:

A lo largo de la materia pudimos entender la importancia de la planificación en el uso de las estructuras a la hora del desarrollo de software. Los datasets brindados sirvieron para el análisis y cuantificación de la mejora en los procesos, al implementar una, u otra estructura.

Resultó muy importante, el uso y conocimiento de diferentes técnicas en la construcción de algoritmos, como, por ejemplo, la "poda" en el caso del backtracking para desechar candidatos o soluciones temporales que insumirían tiempo computacional y no necesitan ser analizados por ya no ser aptos al procesarlos. También lo resultó el algoritmo Greedy, el cual nos brinda una solución aceptable y muchas veces óptima, a un costo mucho menor al que resultaría de evaluar todas las condiciones.

Lo aprendido tiene multiples utilidades, pero en cuanto a datos:

Según "*El informe de 2019, 'Data Never Sleeps 7.0', afirma que este año habrá 40 veces más bytes de datos que estrellas hay en el universo observable. En el gráfico de 2018, se afirmaba que "se crean más de 2,5 quintillones de bytes de datos todos los días" y que se estima que cada persona en la Tierra genera 1,7MB de datos por segundo. Somos 7.75 billones."*

Fte: https://www.20minutos.es/noticia/4368243/0/cada-persona-en-la-tierra-genera-1-7mb-de-datos-por-segundo-que-se-puede-hacer-con-toda-esa-informacion/

Con lo indicado, si no se conocen y aplican las estructuras adecuadas, habrá datos que nunca podrán ser procesados.

Matías Farinelli
34.296.907