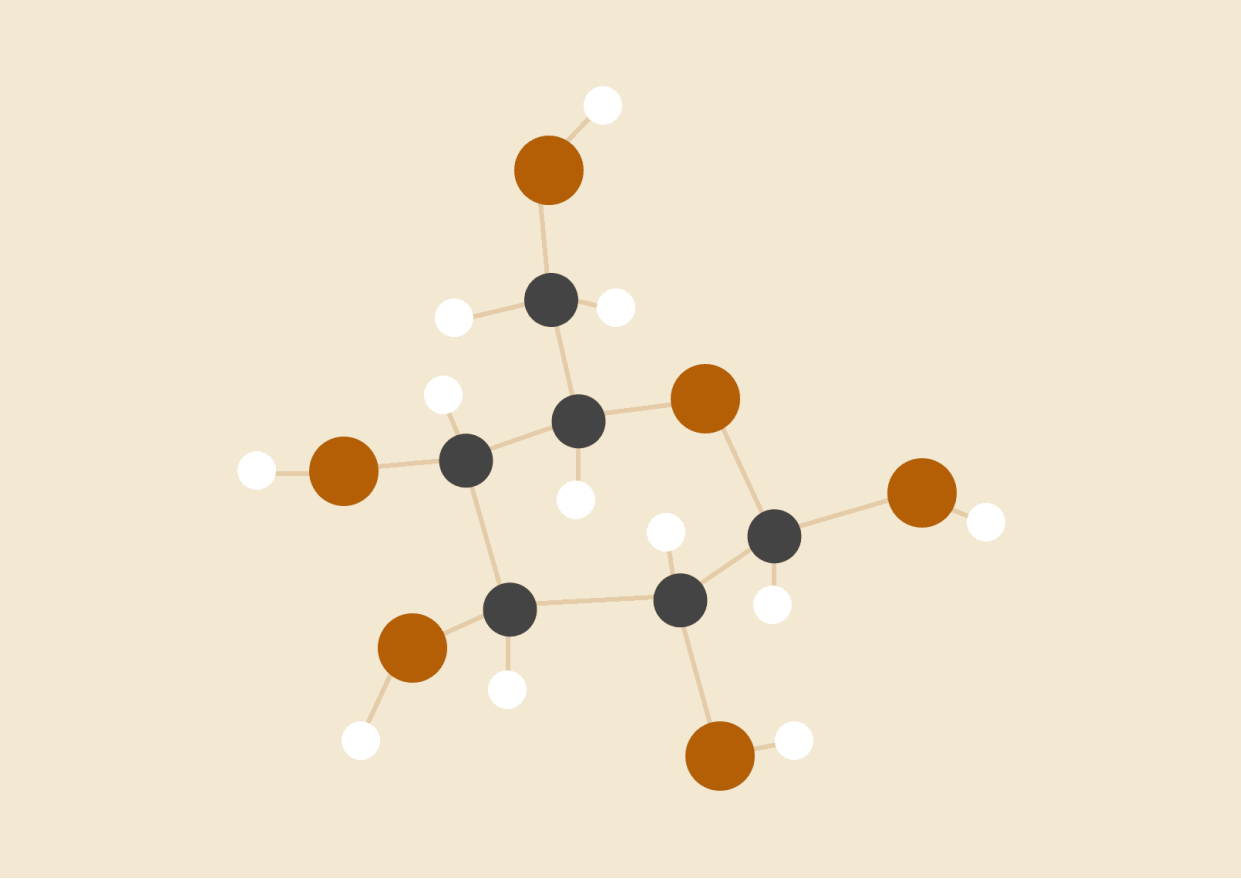
Trabajo Práctico Especial

2da Etapa

Garcia Reinhold, Arturo, Wibaux Germán | Programación III | 04/05/2018

Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas.

UNICEN



Introducción

En esta segunda parte del proyecto nos hemos propuesto a desarrollar la consigna brindada por la cátedra, en una continuación del trabajo práctico especial de la materia. Siguiendo con nuestro estudio de las estructuras de datos, en este caso nos toca analizar al grafo, las estructuras que lo componen y el funcionamiento que hemos de darle para poder explorarlo, manipularlo, obtener información del mismo y de las conexiones/aristas entre cada uno de sus nodos/vértices.

La consigna nos pedía realizar un análisis de la utilización del buscador, implementado en la primera parte, por los usuarios. Partimos del supuesto que ahora nuestra herramienta permite ingresar un conjunto de categorías o géneros a buscar. Por lo que nosotros en esta segunda etapa partimos del análisis de las búsquedas realizadas por los usuarios. Lo que nos da la pauta que el trabajo se centra en saber discernir relaciones en un grafo, saber leer sus datos y trabajar con su estructura.

Se debía implementar tres servicios que ha de proveer la herramienta: Obtener los “n” géneros más buscados luego de buscar por el género “a”, obtener todos los géneros buscados luego de buscar por “a” y finalmente obtener los géneros afines al género “a”.

Podemos ver claramente que los servicios apuntan a obtener información de nuestra estructura, una vez que la hayamos generado a partir de la información volcada por los usuarios en sus búsquedas.

La aplicación real de estos servicios podría bien ser utilizada en cualquier web relacionada con la venta de libros, ya que por cada género en el que un usuario se interese, este último podrá ver sus géneros afines y cuáles son los géneros que la mayoría de la gente opta por leer luego de haber buscado o leído por el primer género.

Trataremos de ser lo más concisos y claros en nuestro desarrollo de cómo planteamos el problema, cómo pensamos su solución, y que estructuras elegimos para implementar la estructura, valga la redundancia, que contiene todos los datos sobre las búsquedas de los géneros. Presentaremos las justificaciones de cada una de nuestras elecciones y mostraremos los resultados que nos arrojaron, siendo éstos un respaldo de la explicación teórica.

También hemos de presentar un grafo dirigido que servirá de ejemplo y sobre el cual explicaremos el funcionamiento e implementación de cada uno de los servicios requeridos, junto con una breve explicación de los algoritmos para recorrer el grafo, detectar ciclos en el mismo (para obtener géneros afines a un género dado) y obtener información sobre los géneros adyacentes a un género dado.

Por cada servicio explicaremos también su tipo de retorno y por qué creímos conveniente el mismo, además de algunas cuestiones de eficiencia en lo que nosotros consideramos incluir como estructuras, analizando sus factores positivos y negativos.

Desarrollo

Estructuras elegidas en la implementación del Grafo

A continuación, pasaremos a analizar las estructuras que creímos convenientes para la implementación de nuestro grafo, desde un punto de vista eficiente tenido en cuenta para su construcción, recordando que la misma se hacía a través de la lectura de datos otorgados por la cátedra, junto con un punto de vista asociado al uso de las estructuras por parte de nuestros tres servicios.

Comenzando con la clase del GrafoDirigido, lo primero a tener en cuenta parte de la definición misma del grafo y es que éste es un conjunto de elementos llamados vértices o nodos unidos por enlaces llamados aristas o arcos. El grafo puede contener su estructura interna mediante una lista de adyacencias o una matriz de adyacencias. Aquí encontramos la primera dicotomía a resolver.

Partimos de pensar cuál estructura se adecúa mejor a la creación del grafo, siendo las mismas inserciones de géneros que no estén repetidos, junto con sus aristas, aumentando su peso o simplemente creándolas. Comenzamos pensando en la matriz de adyacencias, por un lado, nos ofrecía la ventaja de una recuperación rápida de las adyacencias ya que permite saber si existe una arista o no entre dos nodos (y su peso en este caso) con una complejidad de O (1), siendo también fácil de implementar y guardando la información que necesitamos de la arista. Por otro lado, tenemos como desventaja la complejidad espacial de la matriz, ya que se necesitan n^2 casilleros para representar un grafo de n nodos y como información aparte también sabemos que recorrer los vecinos de un nodo de esta manera va a tener una complejidad de O(n). Pero acá tenemos que tener en cuenta un detalle, al grafo le iremos agregando nodo por nodo y arista por arista, por lo que no conocemos el tamaño final del mismo con respecto a la cantidad de nodos, en este caso representar una matriz nos lleva al problema de tener que definirla con un espacio aleatorio y cuando éste se llene copiar todos los datos a una matriz nueva lo que sería bastante costoso O(n^2).

Lo que nos llevaría a hacer un chequeo cada vez que la matriz se llene y su posterior copia a una matriz más grande, realizando este procedimiento una cantidad desconocida de veces.

La próxima estructura a tener en cuenta para la representación interna de las adyacencias es la lista de adyacencias, la idea es la misma, guardamos para cada nodo o vértice una lista de pares (vértices con sus pesos) que indican el nodo vecino y el peso de la arista que los une. Las ventajas de esta implementación son por un lado la complejidad espacial lineal en el tamaño del grafo, ya que necesitaremos para un grafo de n nodos y a aristas una memoria aproximada O(a+n). Por el otro lado recorrer los adyacentes o vecinos de un vértice es O(n) donde n es la cantidad de adyacentes al vértice. Las desventajas son que ahora no es una complejidad lineal O (1) la que tiene la operación de chequear si dos nodos son adyacentes y también es más difícil de implementar y manipular que la matriz de adyacencias.

También en nuestro análisis hemos de tener en cuenta que la estructura que guarde los vértices ha de ser recorrida por cada entrada de un género, ya que deberemos revisar en todo nuestro grafo si el género ya está incluido, y esa búsqueda en el mayor volumen de datos se realizará varios millones de veces. Por lo que a priori parece ser fundamental que la estructura de almacenamiento de vértices sea una estructura que me permita reducir ese tiempo que en el caso de la lista de adyacencias y la matriz es O(n) siendo n la cantidad de vértices que tiene el grafo. Por esto se nos ocurrió pensar en una estructura trabajada anteriormente que es el árbol, el árbol binario en este caso me ayudaría a reducir ese tiempo de búsqueda que tengo por cada género que me llega. Recordando un poco, me permite mantener un orden alfabético y si se encuentra balanceado me reduce el costo de búsqueda a (log 2 n), siendo en el peor de los casos O(n) como las estructuras previamente analizadas.

Por lo que tenemos en cuenta para la inserción en la lista de adyacencias si utilizamos la interface List que nos provee java la inserción será de O (1), en el caso de la matriz ya describimos la situación previamente.

Decidimos entonces elegir una vez más el árbol para ordenar alfabéticamente los géneros y aprovechar así la ventaja que nos ofrece en la búsqueda de los mismos. Ya que como implementamos la creación del grafo, buscar si un género está presente es una operación que realizamos varias veces, sobre todo en los volúmenes de datos más extensos.

Eso corre para almacenar los vértices. Por lo que para guardar las aristas en el grafo usamos una lista normal que contenga instancias de la clase Arista, conteniendo esta última su nodo de origen, nodo destino y el peso. Ya que nos vimos imposibilitados, en el marco de nuestras capacidades e ideas del momento pensar en una estructura más eficiente para almacenar las aristas. Con respecto a las adyacencias, para facilitar su acceso, decidimos que los vértices tengan una lista de aristas, denominada adyacencias, esta lista tiene todas las aristas que tienen como nodo origen al nodo en cuestión y están estas aristas ordenadas, luego de generarse todo el grafo, de mayor a menor (para que luego las consultas que se hagan por el primer servicio recojan las “n” primeras adyacencias del género que se ingresó).

Para repasar el grafo resultante con las estructuras elegidas posee:

Una List<Aristas> que almacenan todas las aristas que tiene el grafo. (para facilitar la búsqueda en el momento de ver si una arista debe agregarse o ya existe y su peso debe ser aumentado).

Un árbol binario de búsqueda que almacena todos los vértices que tiene el grafo. Este tipo de aplicación no es para nada común ya que se suele implementar con una lista de adyacencias o una matriz de adyacencias, pero para este caso en particular lo creímos más conveniente, para las situaciones previamente analizadas, no encontrando una restricción teórica en contra, ni tampoco negando que las haya. Cada nodo del árbol contiene su hijo izquierdo, su hijo derecho y a su vez tiene como información el vértice.

- Impacto de las estructuras en los servicios

En el primer servicio donde hay que obtener los N géneros más buscados luego de buscar por el género A. Se resuelve simple ya que cada vértice tiene su lista de adyacente ordenada de mayor a menor por peso de cada una. Entonces no es más que retornar esa misma lista de un vértice dado. No tiene mayor complejidad que devolver los n adyacentes de un vértice por lo tanto es O (nro. de adyacentes). En este caso el impacto de una implementación de listas de listas para el grafo facilita devolver los adyacentes de cualquier vértice dado.

En el segundo servicio en donde hay que dar todos los géneros que fueron buscados luego de buscar por el género A. No fue más complejo que implementar un dfs dentro de nuestro grafo. En este caso la implementación del grafo también se comporta de manera favorable para hacer esta clase de recorridos.

En el tercer servicio pasa algo parecido que, en el segundo, directamente vamos a ver en la parte del análisis con gráficos y estadísticas el impacto real de las estructuras sobre los servicios.

• DECISIONES DE DISEÑO

- Herramienta

Para la implementación de la herramienta, seguimos la misma idea que en la primera parte del trabajo. Nuestra herramienta llamada LiteraryGenreAnalyzer está compuesta por un grafo dirigido y por un CSVReader. El funcionamiento de la misma es simple, en su creación, crea una instancia de la clase CSVReader, que recopila la información de los datasets y una instancia del grafo dirigido.

La herramienta comienza generando un grafo con el path que recibe del usuario, siendo el dataset del 1 al 4. Trae todas las líneas que le brinda el CsvReader con el path recibido de parámetro y por cada una extrae de a uno los géneros, fijándose si se encuentran o si no y luego añadiendo la arista correspondiente si no se encuentra o aumentando su peso si ya estaba. Al finalizar la creación del grafo, obtiene todos los vértices que lo componen y ordena sus aristas con sus adyacentes de mayor a menor teniendo en cuenta el peso.

Por el otro lado la herramienta tiene las funciones para los tres servicios a implementar que serán explicados más adelante, con un grafo de ejemplo.

La herramienta internamente es soportada por un grafo dirigido. Este grafo posee una estructura que es una lista de adyacencias más concretamente un ArrayList de tipo Arista. Donde Arista tiene un peso, un origen y un destino. Un abb (Árbol Binario Balanceado) de tipo Vértice llamado vértices donde se almacenan los vértices de manera ordenada. La clase Vértice posee un String info que le da el nombre,

una List de tipo Arista llamado adyacentes donde almacena los adyacentes a cada vértice y un estado que es utilizado para los recorridos dentro del grafo.

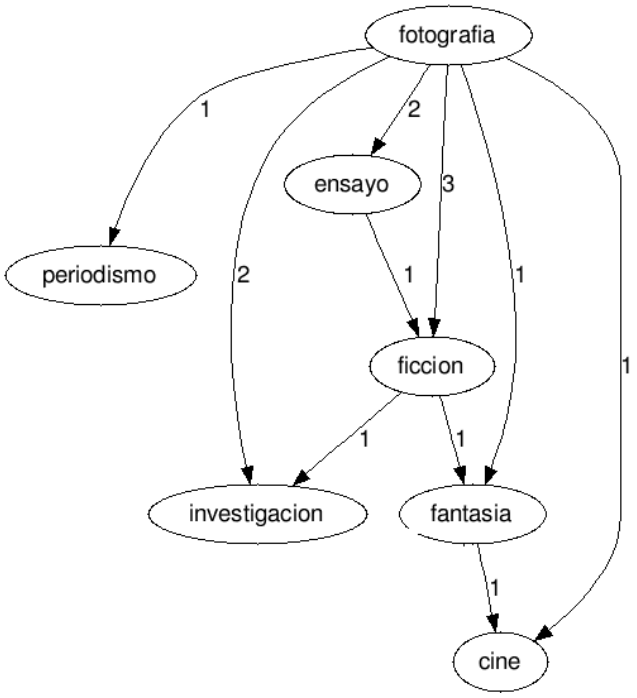
La clase GrafoDirigido tiene los métodos normales como el constructor. Los métodos addVertice y addArista que agregan un vértice en el árbol y una arista en la lista respectivamente. También tiene métodos que indican el número de aristas y también si ya existe una arista dada o no. Por el lado de los vértices nos da un vértice dado y los adyacentes a un vértice indicado. Y además de eso posee los métodos para realizar un dfs y encontrar ciclos dentro del mismo.

A grandes rasgos estas estructuras descriptas son las que implementamos para resolver la problemática planteada.

- Funcionamiento de los servicios

  - Primer Servicio

El primer servicio que debía proveer la herramienta pedía obtener los “n” géneros más buscados luego de buscar por el género “a”.
En torno al problema tenemos que luego de buscar por un genero, al buscar otro se genera una conexión entre los mismos que da vida a la arista entre el primer genero buscado y el segundo, esta arista podia aumentar su peso a medida que la misma búsqueda se repetia. Por ende el problema implicitamente me esta pidiendo que devuelva los “n” mayores adyacentes al género dado, es decir los más buscados luego de buscar por “a”. Por ende como en nuestra estructura los vertices tenian una lista de aristas denominada “adyacentes” y esa lista se encontraba ordenada en orden decreciente de acuerdo a su peso, para usar el servicio solo deberemos pedirle al vértice/género en cuestión que nos devuelva sus “n” más grandes o en caso de que su tamaño de adyacentes sea menor a “n”, todos los que tenga.
Mostramos a continuación el grafo de ejemplo y el uso del servicio:
Entrada:



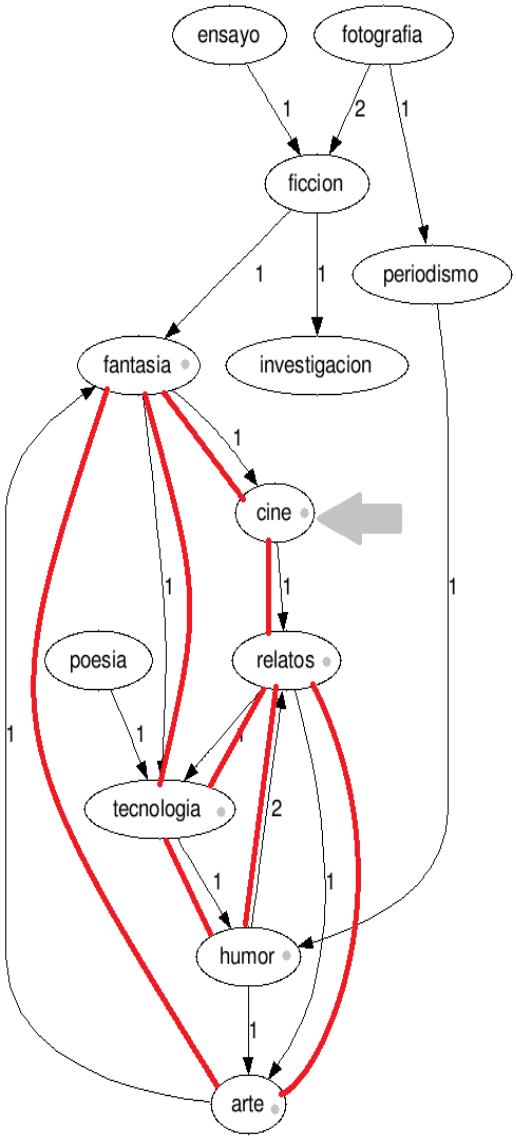
n = 3
género= fotografía
return tool.ServicioA(n, género)

Resultado arrojado:
[ficcion, investigacion, ensayo]

Como podemos ver el género fotografía tiene seis adyacentes, de esos seis, me tengo que quedar con los 3 más buscados, es decir los destinos de las aristas con mayor peso, como se encuentra ordenada mi lista de adyacencias devuelvo ficción (3), investigación (2) y ensayo (2).

  - Segundo Servicio

El segundo servicio nos pedía obtener todos los géneros que fueron buscados luego de buscar por el género A. Para extraer esta información del grafo rápidamente se nos hizo evidente que se podía realizar una búsqueda en profundidad y devolver todo lo buscado luego de género A, para esto implementamos el DFS, para cada adyacente al vértice dado como parámetro (al principio todos con estado en blanco) lo visitábamos (volviendo su estado a amarillo) y así con sus adyacentes, etc., etc. Al visitar todos sus adyacentes, volvíamos su estado en negro, en la vuelta recursiva al cambiar el estado del género dado en negro ya teníamos los géneros del sub-grafo resultante.
Mostramos a continuación el grafo de ejemplo y el uso del servicio:



Entrada:
género= cine
return servicioB(cine)

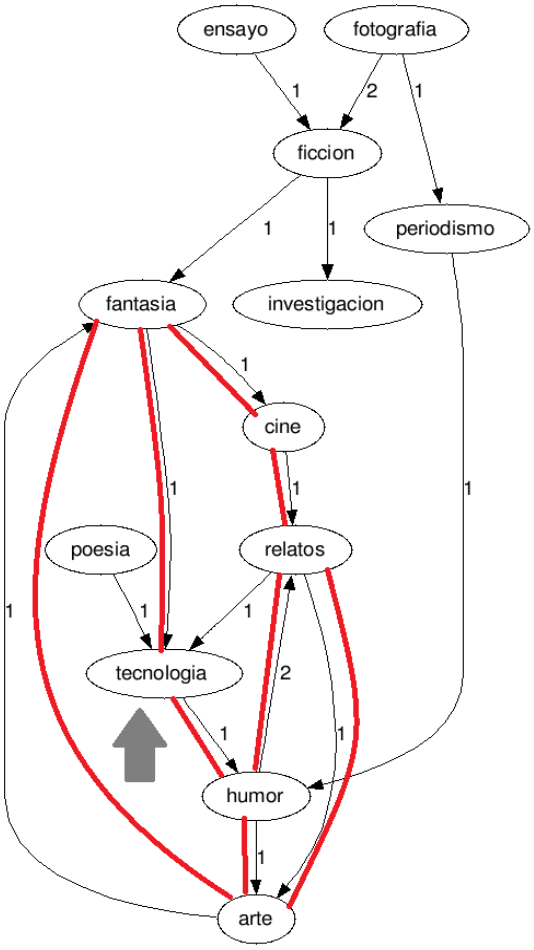
Resultado:
[fantasía, arte, humor, tecnología, relatos, cine]

- Tercer Servicio

Para el tercer servicio que debía proveer la herramienta se nos pedía obtener el grafo únicamente con los géneros afines, es decir, que se vinculan entre sí (pasando o no por otros géneros).
Aquí primero debimos definir lo que significaba "afín" y es que dos géneros son afines si uno era buscado después del otro y viceversa, habiendo o no otros géneros en el medio. Por lo que pudimos darnos cuenta que se nos pedía que detectemos los ciclos que se generaban a partir de un vértice dado. Nuestro grafo dirigido obviamente ha de ser cíclico por las múltiples conexiones/aristas que hay entre sus vértices, por

lo que se nos presentaba el reto de detectar un ciclo, por un lado, obtenerlo (guardando los géneros que participan) y continuar la búsqueda de ciclos, sin quedarnos solamente con el primero, ya que esto no cumpliría con la consigna en el momento que nos piden todos los “afines”.
Con respecto a la implementación, investigando un poco y siguiendo lo aprendido en la cátedra, sabemos que una de las aplicaciones del anteriormente mencionado DFS (depth first search) es usada para encontrar ciclos, ya que como vamos expandiendo cada uno de los nodos, en un momento nos chocamos por un nodo por el cual ya habíamos pasado, si éste nodo en cuestión es el mismo del cual partimos tenemos un ciclo. No quedándonos sólo con esto debemos seguir expandiendo la búsqueda por el resto de los nodos adyacentes al nodo donde nos quedamos en la llamada recursiva.

Mostramos a continuación el grafo de ejemplo y el uso del servicio:



Entrada:
genero= tecnología
return ServicioC(genero)

Resultado:
[tecnología, relatos, humor, cine, fantasía, arte]

Como podemos ver, no se quedó con el primer ciclo encontrado (tecnología, humor, relatos), sino que siguió buscando, pasando varias veces por géneros ya incluidos en el conjunto solución y no agregando repetidos.
Cabe destacar que para mayor entendimiento del resultado también incluimos en el conjunto solución el género que nos dan de parámetro, del cual partimos.

- ## Datos de las pruebas con los distintos datasets

En el primer dataset para 35 generos, en la búsqueda de generos se visitaron 654 nodos. Para el segundo servicio tuvimos 33 entradas recursivas. En el caso del tercer servicio fueron 32 entradas recursivas y 66 comparaciones.

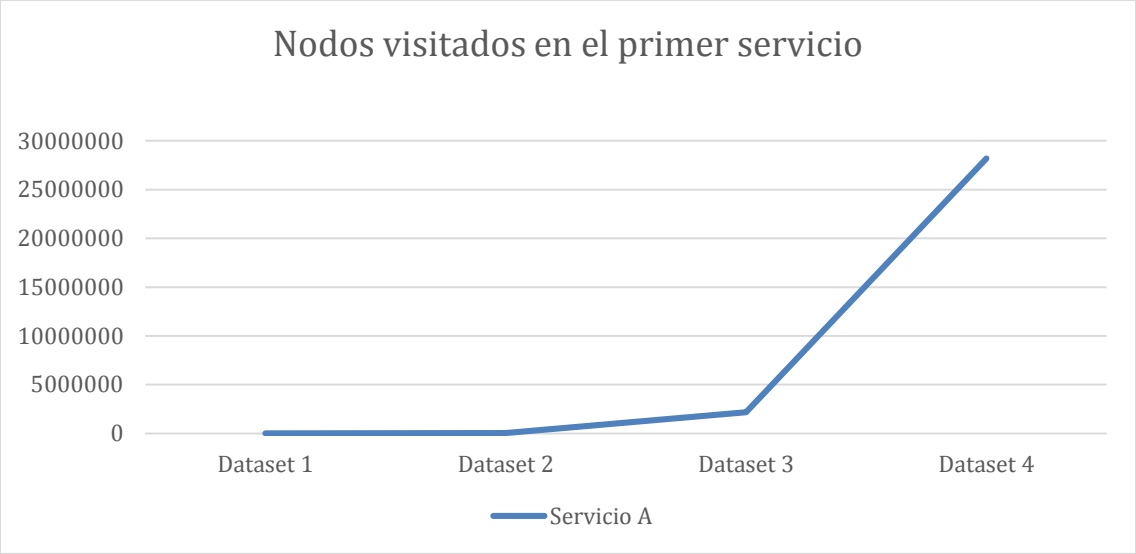
Para el segundo dataset con 40 generos para el primer servicio se visitaron 23013 nodos, en el segundo

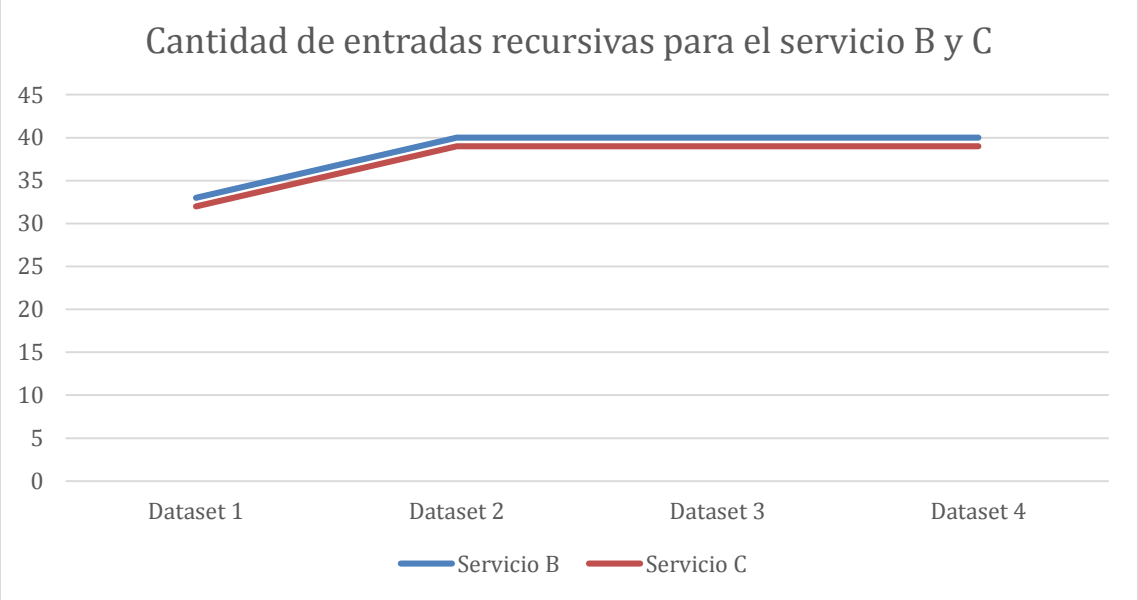
servicio se obtuvieron 40 entradas recursivas y en tercer servicio fueron 39 entradas recursivas y 1292 comparaciones.

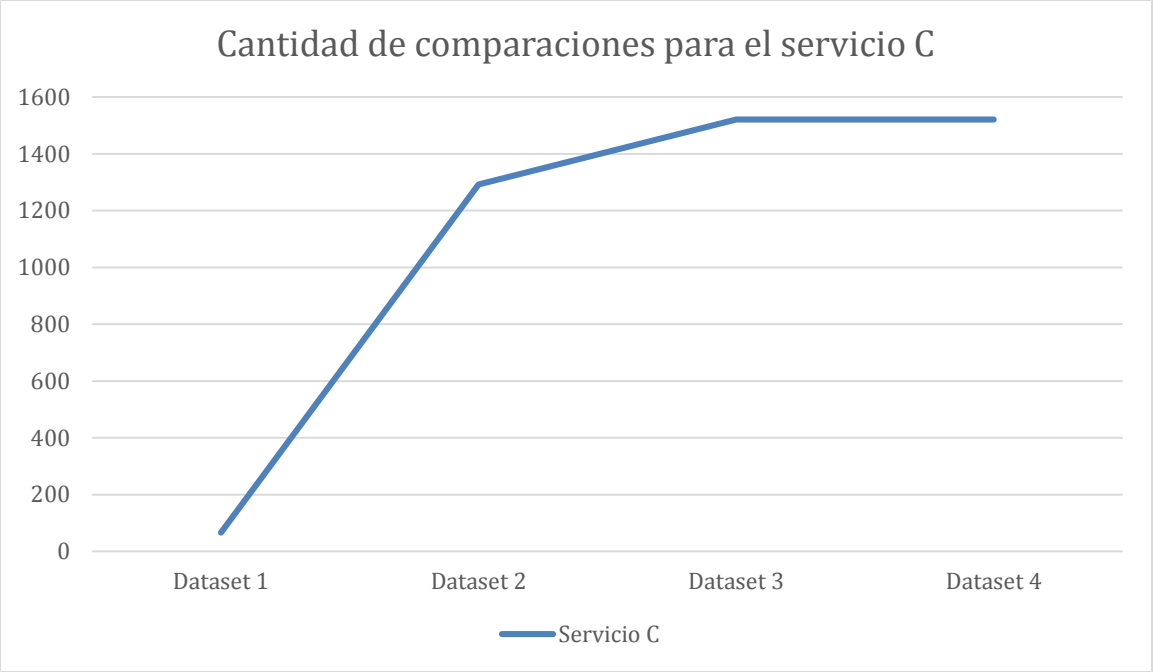
En el tercer dataset es muy similar al segundo salvo en 2 datos. En la búsqueda la cantidad de nodos visitados fue de 2155622 y en el tercer servicio la cantidad de comparaciones fueron 1521. Después todos los datos son exactamente iguales.

Para el cuarto y último dataset las estadísticas variaron con respecto a los anteriores. Para el primer servicio siendo también 40 generos la cantidad de nodos visitados fue de 28192692. Las entradas recursivas para el segundo y tercer servicio fueron de 40 y 39 respectivamente. Y para el tercer servicio la cantidad de comparaciones fue de 1521.

Cabe destacar que para esta prueba en los 3 servicios se utilizó el género cine tanto para buscar sus adyacentes como para el dfs y encontrar sus afines.







- ## CONCLUSIÓN

Con todo lo expuesto anteriormente y comparando con implementaciones experimentadas de manera individual por cada integrante del grupo. Podemos concluir en que la elección de una estructura que no es utilizada muy habitualmente, resulto arrojar resultados coherentes en cuanto a las entradas de datos y cantidad de acceso que manejo cada algoritmo implementado. Si hacemos hincapié directamente en los numero arrojados en la prueba con el género Cine, se puede exponer que cada número es coherente y que solamente tiene un crecimiento exponencial cuando debe recorrer varios nodos por la cantidad de entrada de datos, lo que resulta lógico, y en algoritmos donde no tiene que importar la cantidad de entrada de datos ya que para encontrar los géneros que se recorrieron después del buscado o sus afines la lógica del algoritmo es similar. Nuestras especulaciones al principio resultaron bastante acertadas en cuanto a los resultados que arrojaron las distintas pruebas.