Trabajo Práctico Especial

Etapa 2

Programación 3

TUDAI

Facultad de Ciencias Exactas

Universidad Nacional del Centro de la Provincia de Buenos Aires

Integrantes:

Miguez, Jose Maria (josemiguez148@gmail.com)
Tubino, Bruno Gonzalo (bruno.tubino33@gmail.com)

Entrega:

6 de Junio del 2018

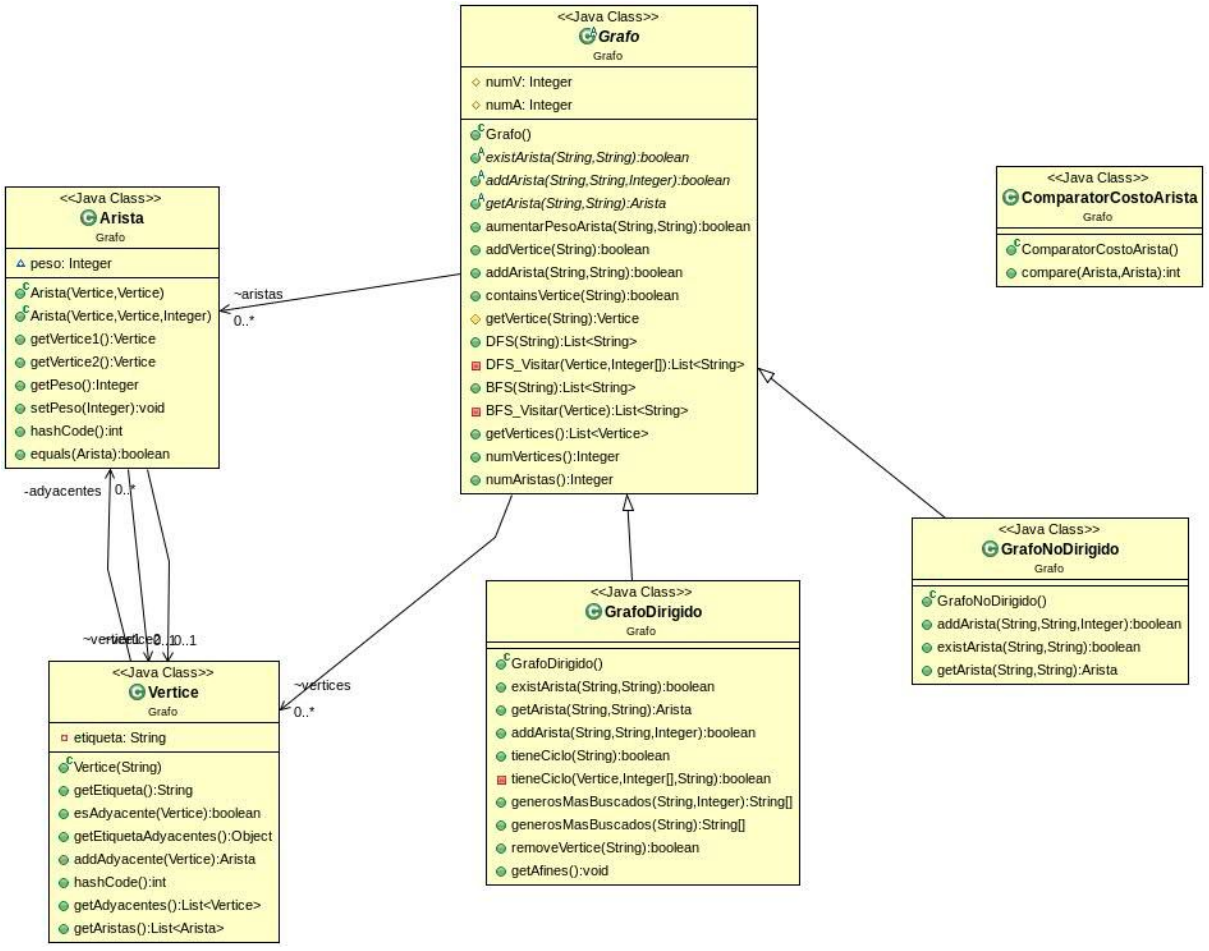
Introducción:

A continuación se detalla el desarrollo de la segunda etapa del Trabajo especial dado por la cátedra de Programación 3, el cual consiste en extender la funcionalidad de la primera entrega mediante el uso de un grafo que representa la relación entre las búsquedas permitiendo así obtener una estadística que refleje el comportamiento del usuario.

Estructuras utilizadas:

Grafo:

Implementación:



Para implementar esta estructura se definió una clase abstracta “Grafo” de la cual es extendida por “GrafoDirigido”(Grafo unidireccional) y “GrafoNoDirigido” (Grafo bidireccional). El modelado del grafo está compuesto por las clases “Vértice” y “Arista” (Detalladas más adelante) las cuales son contenidas en 2 ArrayList, una contiene los vértices y otra las aristas presentes en el el mismo, a su vez se lleva un conteo de la cantidad de Nodos y Aristas.

Vértice:

La clase Vértice está modelada por una etiqueta (String) única la cual lo define y una lista de Aristas adyacentes al mismo (Más adelante se detalla por qué se tomó esta decisión).

Aristas:

La clase Arista está modelada por un costo asociado (Integer) y 2 Vertices (Origen y destino)

Desarrollo del trabajo:

Discusiones:

- Este enunciado pide implementar un Grafo para tener un seguimiento de las búsquedas. Se decidió usar un grafo dirigido, ya que si se busca un género A y después otro B , no es lo mismo que si se busca en el sentido inverso.

- Para implementar el grafo mencionado anteriormente se utiliza una lista de adyacencia , está tiene un tamaño dinámico y un menor uso en memoria comparada con la matriz de adyacencia , que tiene un tamaño estático.
  Se contempló la idea de utilizar una matriz, pero esta presenta problemas a la hora de hacerla dinámica, ya que se debe redimensionar cada vez que esta completa creando una matriz nueva y volcando todos los datos que se tienen hasta el momento, este es un proceso muy costoso para la memoria y ya no la consideramos una opción viable. En resumen se debía utilizar un grafo con un tamaño estático - N de vértices o una lista dinámica de adyacencias, se decidió optar por el dinamismo.

Servicios y comportamiento:

- Se 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 incluído 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.

Para resolver esto se añadieron los métodos CSVReader.listadegeneros(); y CSVReader.setComportamiento(Grafo);

CSVReader.listadegeneros();

Este método recorre todas las líneas del archivo .csv haciendo un split para luego copiar cada género a un conjunto solución (sin repetidos) que es finalmente retornado.

listadegeneros(); es un método auxiliar para posteriormente crear un Vértice por cada género existente en el conjunto solución.

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

  GrafoDirigido.generosMasBuscados(String, Integer);

  En este método se pide la cantidad N de géneros más buscados inmediatamente luego de buscar género (A).

  Se verifica que el género A exista en el grafo , en caso de que exista, se pide la lista de adyacencia que posteriormente es ordenada en orden descendente utilizando un “Comparator<Arista>” el cual compara en base al peso asociado de las aristas presentes en el grafo , luego utilizando una sentencia “for” con i=0; i<N se obtienen los N-Vértices más buscados inmediatamente después que se busco al género A.

- Obtener todos los géneros que fueron buscados luego de buscar por el género A.

  En este método se pide la cantidad total de géneros más buscados inmediatamente luego de buscar género (A).

  Se verifica que el género A exista en el grafo, en caso de existir se pide el size() de su lista de adyacencias para luego llamar al método “generosMasBuscados(String s, Integer n);” donde n es igual a la cantidad de adyacentes, de esta manera se obtienen todos los géneros buscados inmediatamente después de A.

- Obtener el grafo únicamente con los géneros afines, es decir, que se vinculan entre sí .

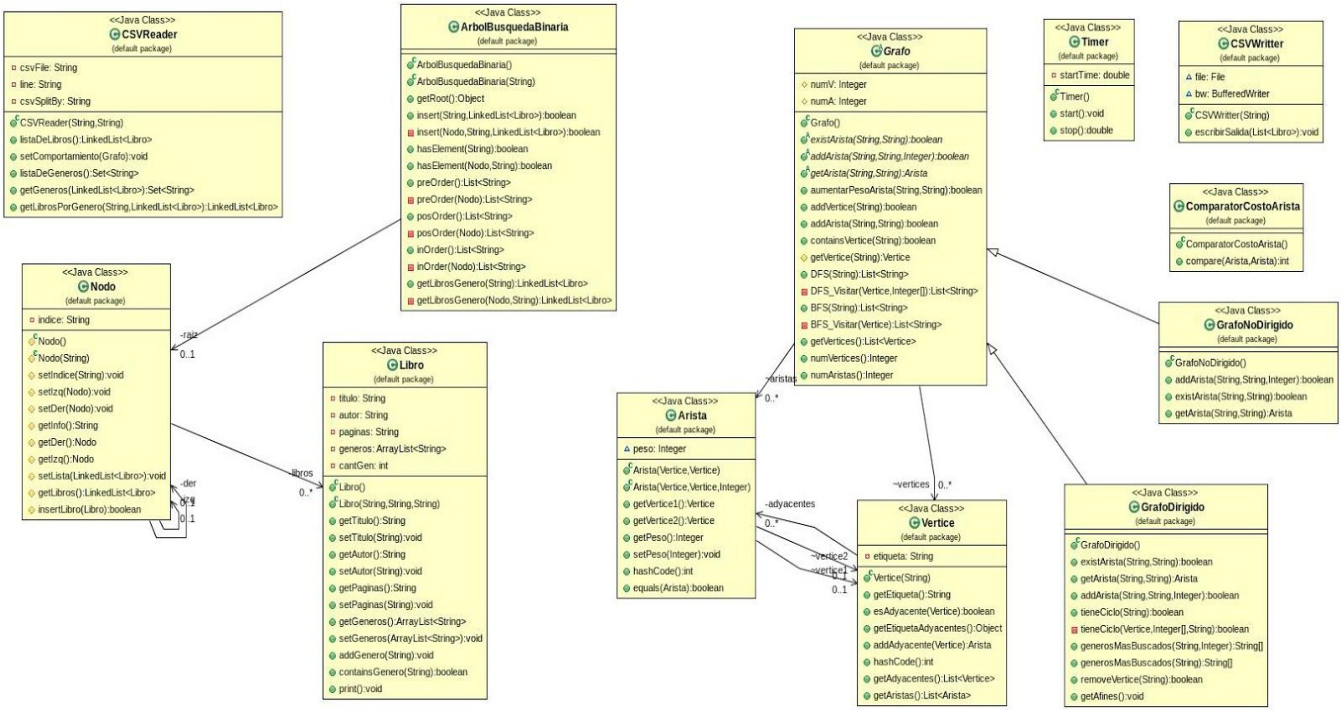
  En este método se pide que el grafo tenga solo los géneros que son afines con algún elemento , para esto se eliminan todos los géneros que no estén en algún ciclo de búsqueda.

  Para resolverlo se recorre la lista de vértices verificando 1 por 1 si forman parte de algún ciclo (si tienen afinidad) , en caso contrario el Vértice y sus Aristas correspondientes son borrados del grafo, obteniendo como resultado final un el grafo sólo con los géneros afines.

Pruebas con diferentes volúmenes de datos

Dataset1	Dataset2	Dataset3	Dataset4
24,76 milisegundos	127,93 milisegundos	4400,28 milisegundos	40337,63 milisegundos

La tabla anterior muestra un promedio (5 pruebas) del tiempo que tarda en modelar el grafo con diferentes volúmenes de datos.



En la figura anterior se puede observar el diagrama de clase del Programa completo (Entrega 1 y Entrega 2).

Conclusión

Con este trabajo logramos llevar a cabo una implementación de Grafo con datos del mismo tipo que están relacionados entre sí . Logramos obtener un conocimiento sobre cómo manejar este tipo de datos, ya sea como guardarlos en una estructura y/o como brindar los servicios necesarios para que el servicio sea mayormente eficiente y cumpla con los objetivos.