Carpeta de Desarrollo

Belén Beltran
belubeltran@gmail.com

Pablo Ariel Rodriguez
prodriguez@fi.uba.ar

Federico Martin Rossi
federicomrossi@gmail.com

2do. Cuatrimestre 2014
75.52 Taller de Programación II
Facultad de Ingeniería, Universidad de Buenos Aires

Antes de empezar

Esta guía asume que el lector se encuentra familiarizado con el sistema operativo *Microsoft Windows* (versión XP y posteriores) y aquellos basados en GNU/Linux. Es decir, se considera que el usuario se encuentra mínimamente en condiciones de desenvolverse en el entorno en el cual se ejecuta la aplicación, razón por la cual no se explican temas como la utilización de cuadros de diálogo, asistentes o el exploradores de carpetas.

Objetivo

El objetivo principal de esta guía es ayudar y guiar al usuario en la instalación, así como también en la utilización de **Graferator** haciendo hincapié en las características y opciones más relevantes que se presentan en este.

Cómo leer esta guía

Se recomienda al usuario leer esta guía teniendo a la vista de su pantalla el software ejecutándose. Esto le permitirá poder realizar los pasos aquí detallados a medida que avanza en la lectura, logrando asimilar más fácilmente la disposición de las distintas opciones mencionadas y la forma en que estas deben ser utilizadas.

Organización

La guía está compuesta de capítulos que agrupan las distintas funcionalidades provistas por la aplicación y que se encuentran relacionadas entre sí. Las secciónes que conforman los capítulos detallan los pasos a seguir para poder hacer uso de dichas funcionalidades.

Contenido

Capítulo 1

Introducción

1.1 Descripción general

El presente documento se centra en el desarrollo de *Graferator*, una aplicación de computadora cuyo objetivo es llevar a cabo el estudio de la teoría de grafos con el fin de ser utilizado en la enseñanza y el aprendizaje de la respectiva temática.

Recordando que un grafo es un conjunto conformado por vértices o nodos unidos por enlaces conocidos como aristas o arcos, el software permitirá de manera gráfica la confección y manipulación de estos a través de una interfaz de usuario intuitiva y simple.

Los usuarios podrán hacer uso de dos modalidades de ejecución, a saber:

- **Modo aprendizaje**: resolución paso a paso con avance a solicitud del aprendiz (al siguiente paso o al resultado final).
- **Modo autoevaluación**: resolución paso a paso con elaboración guiada del resultado de cada paso a cargo del aprendiz. Confirmación de la corrección del resultado con opción de reintentar en caso de que fuere incorrecto, o ver el resultado correcto.

1.2 Funcionalidad

Graferactor constará de las funcionalidades básicas que se requieren para el armado de un grafo. Inicialmente se le permitirá al usuario la creación de un *Grafo Orientado* o un *Grafo No Orientado*.

Además, para cada una de estas opciones se deberá elegir si se desea comenzar a partir de un grafo vacío o si se prefiere generar un grafo aleatorio, especificando la cantidad de vértices y aristas a tomar en cuenta.

De esta manera, la aplicación permitirá:

- el alta o baja de vértices,
- el alta o baja de aristas o arcos,
- el borrado del grafo,
- la ponderación de arcos.

1.3 Algoritmos

Una vez armado el grafo deseado, Graferator permitirá a los usuarios aplicar sobre este una serie de algoritmos que ayudarán y aportarán al estudio de sus propiedades.

Como se ha adelantado previamente, cada algoritmo podrá ser ejecutado en dos modos distintos (modo aprendizaje y modo autoevaluación). Así, los usuarios no solamente comprenderán el grafo, sino que también adquirirán conocimientos acerca de cómo funcionan los algoritmos aplicables a estos.

De esta manera, habiendo confeccionado el grafo, se podrán efectuar los siguientes algoritmos:

- Recorrido en profundidad,
- Recorrido en anchura,
- Prueba de aciclidad.

Si el grafo es orientado, además será posible ejecutar los siguientes:

- Recorrido topológico en anchura y en profundidad (si es acícilico),
- Obtención de la cerradura transitiva,
- Obtención de componentes fuertemente conexas.

Si se trata de un grafo orientado con aristas ponderadas:

- Algoritmo de caminos mínimos de Dijkstra con un mismo origen (requiriéndose que todas las ponderaciones no sean negativas),
- Algoritmo de caminos mínimos de Floyd entre todos los pares de nodos,
- Algoritmo de flujos máximos de Ford-Fulkerson.

Por último, si el grafo es no orientado:

- Algoritmo de árbol de expansión de coste mínimo.

1.4 Ejecución de algoritmos

Dispuesto el grafo sobre la zona de diseño y elegido el algoritmo junto con el modo en el cual se ejecutará el mismo, el usuario deberá dar comienzo a la simulación. Para esto la interfaz contará con una serie de controles con los cuales se indicará el inicio o el fin de la ejecución, así como también controles con los que se podrá desplazar a lo largo de esta.

A medida que se corre el algoritmo se podrá visualizar paso a paso la ejecución de este sobre el grafo mediante colores que facilitarán la comprensión del avance en el mismo.

En el momento en que se realiza la elección del algoritmo, habrá una ventana interna en la que se visualizará el pseudocódigo respectivo. En la ejecución se resaltarán una a una las líneas de las sentencias de manera tal que los usuarios realicen un seguimiento detallado de todo el proceso, siendo esto de complemento y apoyo en el aprendizaje del funcionamiento de los distintos algoritmos. A esto se le incluye el conteo del número de sentencias ejecutadas. De esta manera los usuarios tendrán la información necesaria para realizar distintas tareas de análisis, tales como el cálculo de la complejidad algorítmica, refactorización de los algoritmos, entre otros.

Cada paso podrá ser deshecho dado que no solo existirán controles para realizar el avance, sino también para llevar a cabo el retroceso en cualquier instancia del proceso.

Capítulo 2

Herramientas de Desarrollo

2.1 Lenguaje y entorno de programación

2.1.1 Lenguaje Java

2.1.2 WindowBuilder (GUI)

2.1.3 Eclipse IDE

2.1.4 Maven

2.2 Controlador de versiones

2.2.1 Git

2.2.2 GitHub

2.3 Herramientas para documentación

2.3.1 LaTeX

2.3.2 Draw.io

Capítulo 3

Diseño de la solución

3.1 Arquitectura

En el primer capítulo de este documento hemos detallado la funcionalidad con la que contara Graferator. Allí hemos podido ver que será necesaria una interfaz gráfica la cual será la mediadora entre el usuario y la aplicación.

Para contar con un entorno visual va a ser necesario que exista un motor o modelo que lo respalde. Esto es, un módulo que se encarge de manejar aquellos aspectos abstractos e internos que permitirán que el software funcione correctamente. Dado esto es que se cree conveniente utilizar como base en el desarrollo de la aplicación el *patrón de arquitectura Model-View-Controller* o mayormente conocido por sus siglas MVC.

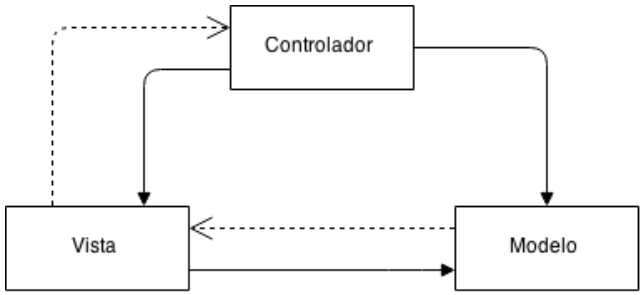

Imagen 3.1: *Diagrama del patrón de arquitectura MVC*

El patrón MVC hará posible separar los datos y la lógica de negocio de la interfaz de usuario (también conocida por las siglas GUI) y el módulo encargado de gestionar los eventos y las comunicaciones. Para esto, dicho patrón propone la construcción de tres módulos descentralizados: *Modelo*, *Vista* y *Controlador*. De esta manera, se establecen por un lado los componentes que hacen a la representación de la información y por otro lado se constituyen componentes para la interacción con el usuario. En la *Imagen 3.1* se muestra un diagrama representativo de este patrón.

La totalidad de la aplicación será desarrollada en base a dicha arquitectura en gran medida por los grandes beneficios que significan poseer el modelo y la lógica separados de todo lo restante. Por esta razón, en los siguientes apartados se profundizará sobre cada módulo y se describirán los componentes que lo conforman como así también la forma en la que trabajan entre sí y se comunican con el exterior para lograr los objetivos planteados por el alcance del software.

3.2 Modelo

Como ya se ha adelantado, el *Modelo* es la representación de la información con la cual nuestro sistema operará. Por lo tanto, gestionará todos los accesos a los datos, tanto consultas como actualizaciones. Mediante solicitudes de la *Vista*, este módulo le enviará la información requerida para ser mostrada y visualizada por el usuario. Estas peticiones llegarán al modelo a través del módulo *Controlador*.

3.2.1 Componentes básicas

Primeramente, encontraremos que el modelo necesitará representar a los constituyentes básicos de los grafos. En la *Imagen 3.2* se puede observar un primer diagrama de clases UML en donde se encuentran las clases *Vertice* y *Arista*. Ambas implementarán a la interfaz *Selectable*, quien simplemente establece que tanto los objetos Arista como Vertice son seleccionables. A su vez, Arista hereda de clase que define a objetos con peso, de manera tal que las aristas podrán ser ponderadas cuando así lo requieran.

Por otro lado, la clase Vertice también implementa las interfaces *Serializable* y *Comparable*, las cuales le dan caracteristicas para poder ser serializados y poder ser comparados respectivamente.

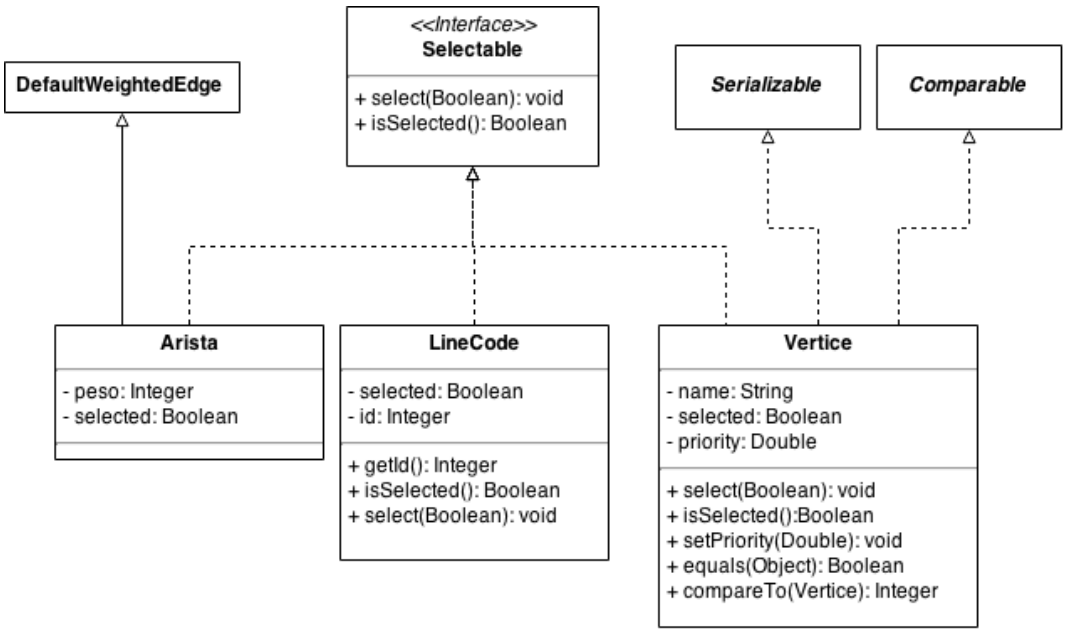

Imagen 3.2: *texto*
texto

3.2.2 Representación del Grafo

Conformadas las componentes básicas, necesitamos una entidad que modele al grafo. Esta clase la denominaremos *GraphModel*. Esta hará uso de un conjunto de objetos Vertice

relacionados entre sí por objetos Arista. En la *Imagen 3.3* se puede observar el diagrama de clases en donde se muestra como está conformado GraphModel.

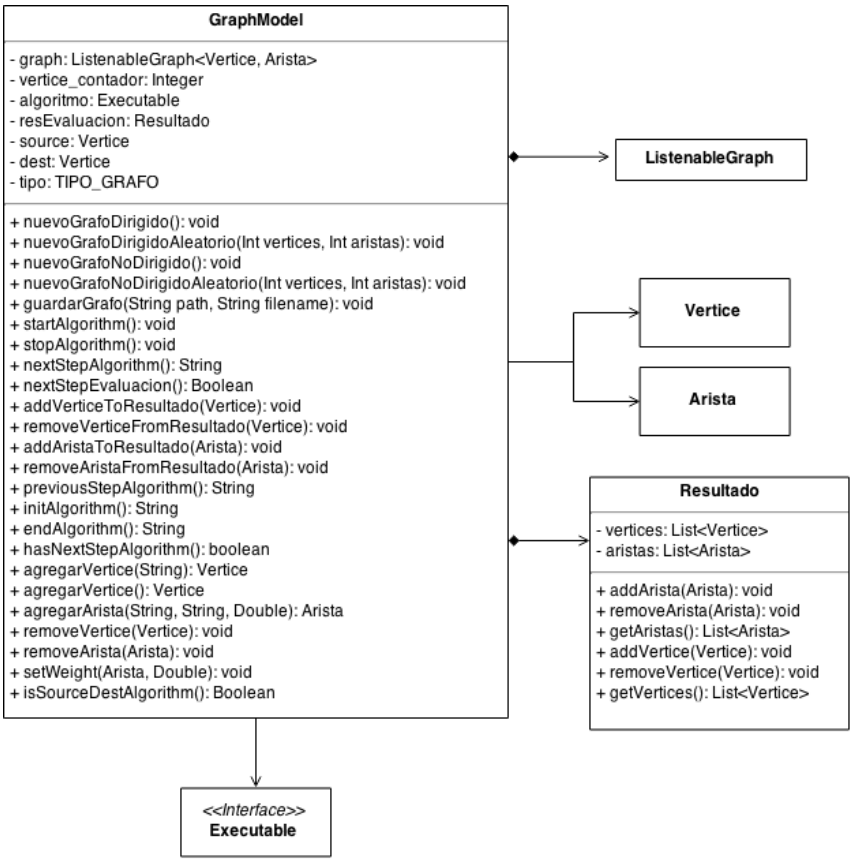

Imagen 3.3: *texto*
texto

3.2.3 Representación de los Algoritmos

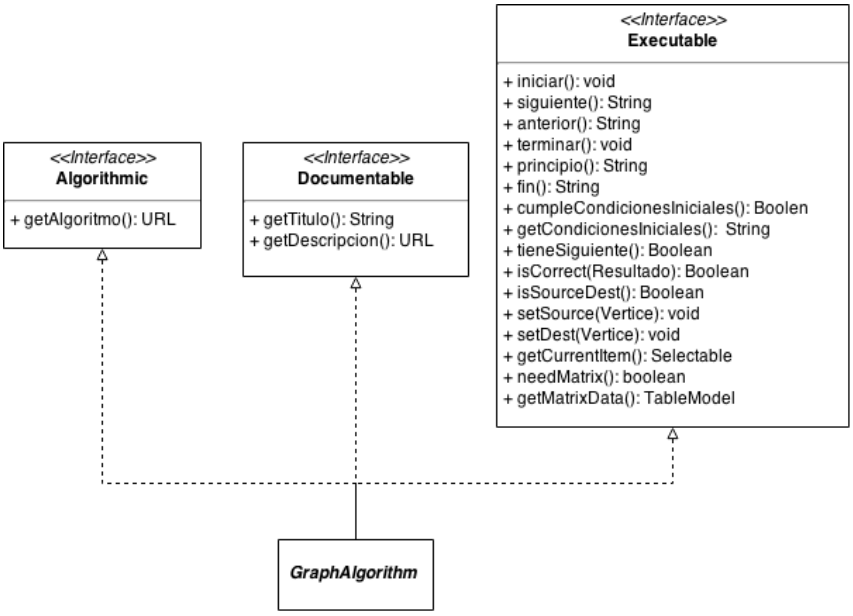

Imagen 3.4: *texto*
texto

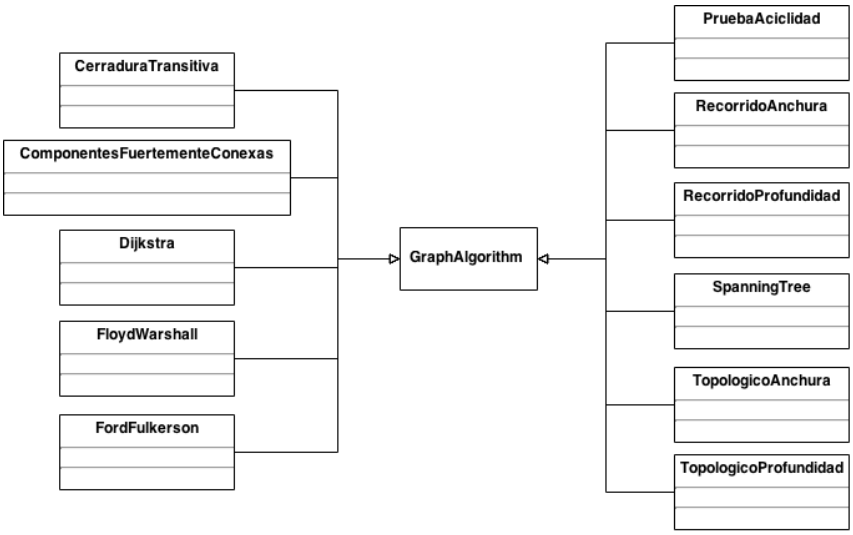

Imagen 3.5: *texto*
texto

3.3 Vista

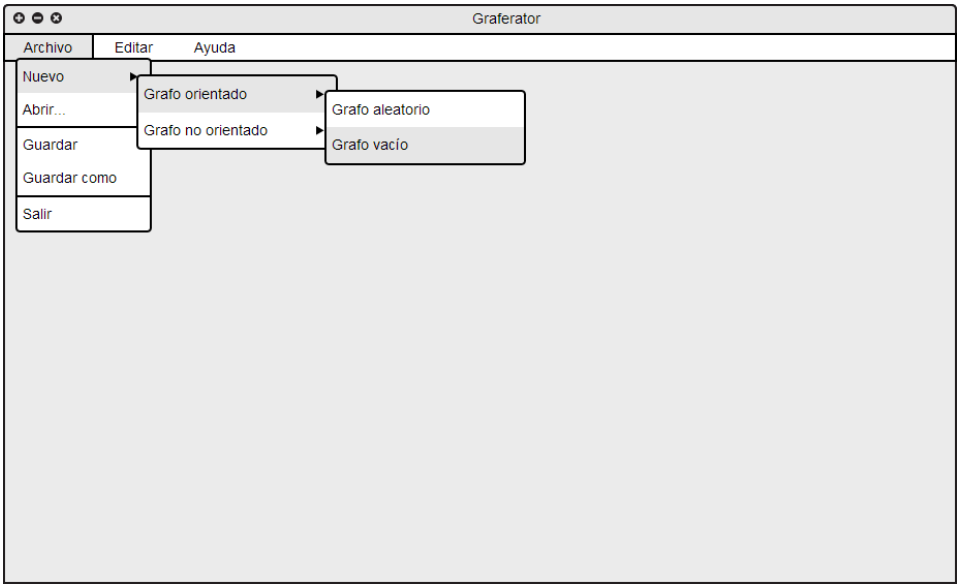

Imagen 3.6: *texto*
texto

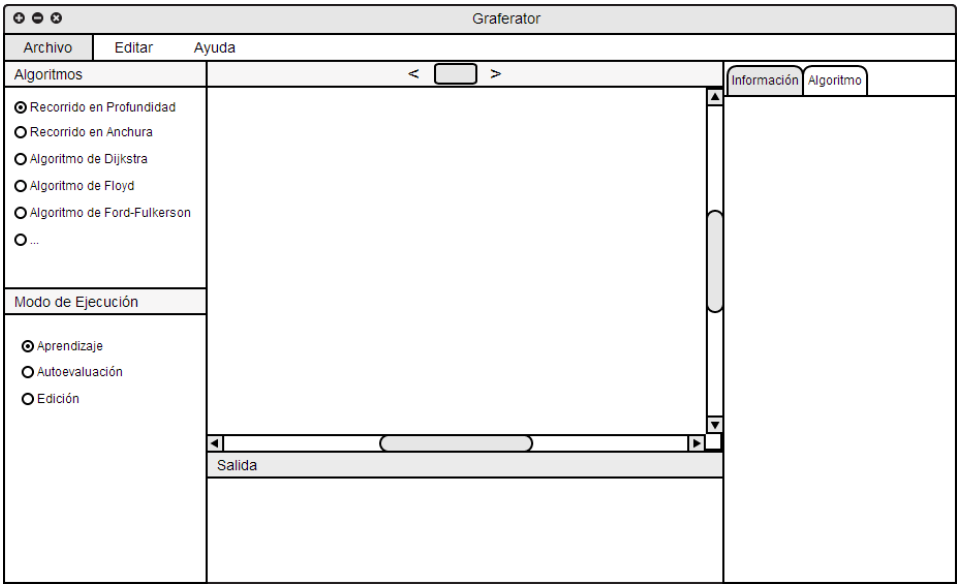

Imagen 3.7: *texto*
texto

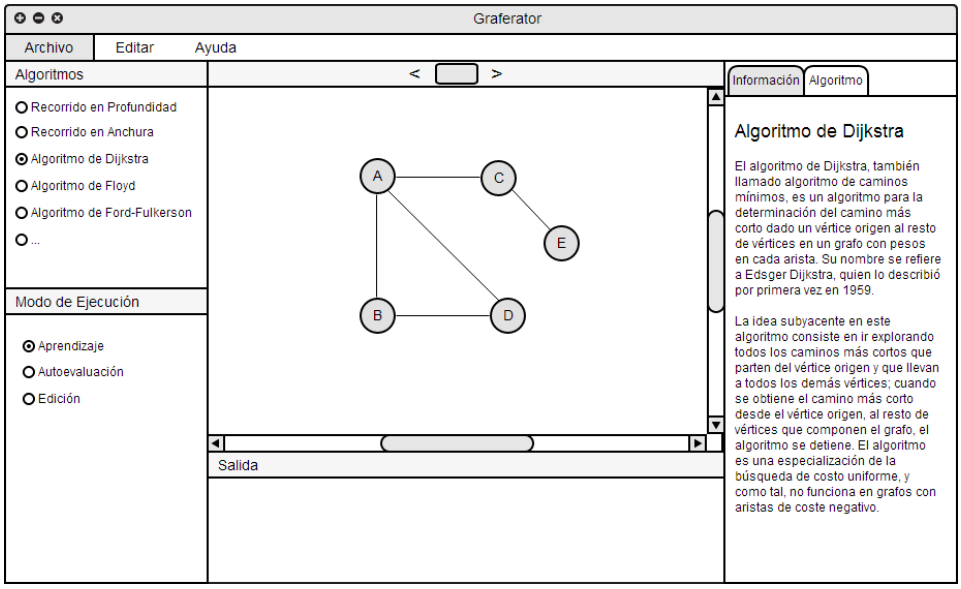

Imagen 3.8: *texto*
texto

3.4 Controlador

Capítulo 4

Distribución de tareas