UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Solving With Graphs 2.0

Proyecto Final de Carrera

Ingeniería Informática

Autor: Jose Chamorro Molina

Director: Cristina Jordán Lluch

Valencia, 14 de febrero de 2014

Resumen

SWGraphs 2.0 es un programa para la ejecución de diferentes algoritmos sobre la teoría de grafos. Permite la creación, edición y análisis de grafos en modo gráfico o en modo matriz de adyacencia, así como la exportación de los grafos a ficheros .xml para su posterior utilización.

Este programa ha sido diseñado e implementado como realización del Proyecto Final de Carrera de 5º curso de Ingeniería Informática, por el alumno Jose Chamorro Molina, bajo la dirección y supervisión de Cristina Jordán Lluch.

SWGraphs 2.0 se ha desarrollado para las siguientes asignaturas de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia:

- ✓ Matemática Discreta, asignatura obligatoria de 1er curso del Grado en Ingeniería Informática.
- ✓ Grafos, modelos y aplicaciones, asignatura optativa de 4rto curso del Grado en Ingeniería Informática.

Palabras clave: grafo, nodo, arista, arco, cadena, algoritmo, ciclo, ciclo hamiltoniano, ciclo euleriano, camino, bucle, matriz, búsqueda en anchura, búsqueda en profundidad, árbol, árbol de expansión mínima, emparejamiento, red, flujo máximo de la red.

Agradecimientos

Antes de comenzar a exponer el proyecto realizado quiero agradecer su colaboración, directa o indirecta, a las siguientes personas y entidades sin las cuales no hubiera sido posible la realización del mismo.

- A Joan Moliner, mi padrino pedagógico y amigo, que me animó a terminar el segundo ciclo de Ingeniería Informática y gracias al cual empecé este proyecto.

- A Escola El Drac y a su director José Royo por haber hecho tan fácil la difícil tarea de compaginar mi labor docente en el centro con la asistencia a la Universidad sin poner ni un solo pero en todo el camino.

- A Cristina Jordán por ofrecerme la oportunidad de participar en este proyecto y dirigirlo con la única intención de obtener un producto de calidad para su utilización en las asignaturas del Departamento de Matemática Aplicada.

- A todos mis amigos y amigas por estar siempre cuando los necesitaba y que, sabiéndolo o no, me han servido como válvula de escape para desconectar y coger fuerzas para las siguientes horas dedicadas al proyecto.

- A mis padres y hermana por ser los pilares fundamentales de mi vida, ya que sin ellos nada de esto tendría sentido.

- Y sobre todo, a Ana, mi inseparable compañera, a la que quiero con toda mi alma y que, con su apoyo incondicional y cariño, me ha permitido terminar este proyecto para poder empezar otro el cual necesitará de todo nuestro esfuerzo, comprensión y cariño el resto de nuestras vidas.

Gracias a todos,

Jose Chamorro Molina

Tabla de contenidos

1. Introducción

1.1. Objetivo del proyecto

El objetivo fundamental del presente Proyecto Final de Carrera es proporcionar una aplicación a los alumnos de las asignaturas "Matemática Discreta", asignatura obligatoria de 1er curso del Grado en Ingeniería Informática y "Grafos, modelos y aplicaciones", asignatura optativa de 4rto curso del Grado en Ingeniería Informática, que les permita resolver, una vez modelados, los problemas propuestos en la realización de las prácticas de las asignaturas mencionadas, aplicando los algoritmos implementados que resuelven problemas de teoría de grafos.

La aplicación debe ser ejecutable en los laboratorios de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia, donde los alumnos realizan las prácticas de las anteriormente citadas asignaturas, así como en los ordenadores de las aulas de libre acceso para los alumnos de informática y en general en cualquier ordenador que tenga instalada la Máquina Virtual Java.

Dicha aplicación podrá ser lanzada sobre cualquier sistema operativo, con objeto de que no existan problemas de portabilidad para su uso en las máquinas que se encuentran disponibles actualmente en la Escuela Técnica Superior de Ingeniería Informática de la UPV.

SWGraphs 2.0 será un programa para la ejecución de diferentes algoritmos sobre la teoría de grafos. Debe permitir la creación, edición y análisis de grafos en modo gráfico o en modo matriz de adyacencia, así como la exportación de los grafos a ficheros .xml para su posterior utilización.

1.2. Alcance del proyecto

La implementación de la aplicación SWGraphs 2.0 permitirá su adopción por parte de los profesores para su utilización en las clases de laboratorio, con el objetivo de poder resolver problemas de interés con base real, que hasta el momento podían ser modelados y cuya resolución se hacía o bien a mano, o bien con Scilab a Mathematica, programas ambos de mayor complejidad de manejo.

La aplicación debe ejecutar los siguientes algoritmos:

- BFS (búsqueda en anchura)

- DFS (búsqueda en profundidad)

- Dijkstra (camino más corto)

- Kruskal (árbol de expansión mínima)

- Edmonds (2) (emparejamientos)

- Hierholzer (ciclo euleriano)

- Cartero Chino

- Ford-Fulkerson (flujo máxima de la red)

- Ciclos hamiltonianos (basado en árbol generador, voraz y por fuerza bruta)

Deben ser programados los algoritmos anteriores, a excepción del de Edmonds (2) y del Cartero Chino programados con anterioridad en otro proyecto final de carrera y de los que se ha reutilizado su código.

La herramienta de desarrollo que se ha escogido es Eclipse Ganymede versión 3.4.2 y el lenguaje de programación JAVA. Aunque hasta ahora el Departamento de Matemática Aplicada utilizaba el software Mathematica y Scilab para las prácticas de la asignatura, se ofrece esta aplicación JAVA por ser más amigable para los usuarios, en cuanto a interfaz gráfica se refiere, no requerir ningún tipo de licencia de uso y tener todas las ventajas de un software a medida.

El método de realización previsto consiste en diseñar una interfaz básica sobre la que probar los algoritmos, comenzando por los pasos básicos en el tratamiento de grafos (definición del grafo mediante dibujo de vértices y aristas, etiquetado de aristas, etiquetado de vértices…) y continuando por resolver, previa modelización, los distintos problemas de teoría de grafos propuestos en las prácticas de las asignaturas.

Tras comprobar el correcto funcionamiento de todos los algoritmos implementados se incluyen funcionalidades adicionales, como la ejecución por pasos en dos de ellos, vistas de la construcción de los grafos en los resultados, etiquetado mediante matriz de adyacencia, guardar y abrir los grafos en formato XML, guardar el grafo resultado, definir una imagen de fondo en el panel de dibujo de grafos, etc.

1.3. Planificación inicial

El proyecto comienza a principio del curso académico 2013/14 como una propuesta de Cristina Jordán para implementar un software de calidad que ofreciera una mejora sustancial en la realización de las prácticas de su asignatura.

La mayor parte del tiempo dedicado a la realización del presente proyecto se va a invertir en el análisis de los algoritmos y al diseño de estructuras de datos que permitan representar en un lenguaje orientado a objetos las estructuras propias de un grafo, así como la de los algoritmos a implementar.

El programa a desarrollar deberá cumplir las especificaciones de requisitos funcionales y no funcionales que se detallan en el análisis del proyecto.

La fecha de comienzo del proyecto se fija en enero de 2013. La fecha de terminación prevista es el mes de febrero del año 2014, empleando en su desarrollo un total de 300 horas.

2. Análisis

2.1. Especificación de requisitos

Requerimientos funcionales

Los requerimientos funcionales son los que se encargan de definir lo que la herramienta software debe hacer. Definen los alcances del sistema en cuanto a las acciones que debe realizar, y en cuanto a la transferencia de datos entre todas las diferentes funciones del sistema. Son descripciones de los servicios que el sistema debe proporcionar.

En el caso de este proyecto requerimientos funcionales son los siguientes:

- El usario podrá dibujar los grafos necesarios para poder resolver los problemas, previamente modelizados, sobre teoria de grafos. Si desea dibujar un grafo no dirigido, sólo podrá dibujar aristas, no arcos. Si desea dibujar un grafo dirigido, sólo podrá dibujar arcos (aparece una flecha en lugar de una línea) y no podrá dibujar aristas. Existirá un panel situado bajo el panel de dibujo, donde se especificará si se desea dibujar un grafo dirigido o no dirigido, así como si se desea dibujar un nodo o un arco/arista.

- Una vez el grafo esté dibujado, el usuario podrá cambiar el tipo de grafo, es decir, podrá pasar de grafo no dirigido a grafo dirigido y viceversa. En el caso de pasar de grafo no dirigido a grafo dirigido, la aplicación preguntará al usuario si desea mantener los pesos de las aristas o ver el grafo como matriz de adyacencia. En el caso contrario, la aplicación advertirá al usuario que se perderán todos los pesos y se cambiarán a 1's (matriz de adyacencia).

- Se podrá cambiar el nombre de los nodos, así como los pesos de las aristas.

- Se deben poder guardar los grafos dibujados en archivos XML, lo que permitirá acceder y abrir los grafos en cualquier otro momento.

- Se deberán poder ejecutar los siguientes algoritmos: BFS (búsqueda en anchura), DFS (búsqueda en profundidad), Dijkstra (camino más corto), Kruskal (árbol de expansión mínima), Edmonds (2) (emparejamientos), Hierholzer (ciclo euleriano), Cartero Chino, Ford-Fulkerson (flujo máxima de la red) y varios algoritmos de ciclos hamiltonianos (basado en árbol generador, voraz y por fuerza bruta).

- Los algoritmos Edmonds (2) y Cartero Chino deberán poder ejecutarse paso a paso.

- Se debe permitir al usuario guardar el grafo resultado de la ejecución de algunos algoritmos en un archivo XML, lo que permitirá acceder al grafo en otro momento.

- El resultado tras ejecutar cualquiera de los algoritmos disponibles se mostrará de forma clara y precisa en un panel de resultados.

- Se facilitarán una serie de grafos de ejemplo para ejecutar los algoritmos y entender su funcionamiento.

- Existirá una opción para defenir el fondo del panel de grafos la cual permitirá al usario escoger la imagen de fondo que le pueda ser útil. Esta utilidad permitirá dibujar el grafo sobre un fondo que represente el problema real a analizar mediante la teoría de grafos. Asimismo, existirá una opción que eliminará la imagen de fondo que se haya establecido con anterioridad.

- Se podrá obtener el código para Mathematica 4.0 (con el módulo grafos.m) del grafo de forma que un grafo dibujado en esta aplicación podrá portarse a Mathematica simplemente copiando y pegando el código obtenido.

- Para los grafos no dirigidos se deberá mostrar el grado de cada uno de los vértices del grafo. Para los grafos dirigidos, se mostrará el grado de entrada y el grado de salida de cada uno de los vértices del grafo.

- Se debe permitir obtener el grafo subyacente del grafo dibujado en el panel de grafos.

- La opción conexión del grafo, tendrá 3 submenús donde el usuario podrá preguntar si el grafo dibujado es conexo, en el caso de grafos no dirigidos, o si el grafo es fuertemente conexo o débilmente conexo, para el caso de los grafos dirigidos.

- Se permitirá al usuario generar un grafo completo: Esta opción permitirá al usuario especificar el número de vértices para generar un grafo completo. Si ya existiera un grafo dibujado, previa consulta al usuario, eliminará cualquier grafo que hubiera dibujado anteriormente en el panel.

- Se podrá abrir un cuadro de diálogo en el que se mostrarán las diferentes matrices del grafo (matriz de adyacencia, de pesos, de multiplicidades, ...). Si no hay ningún grafo dibujado, se preguntará el número de nodos que tendrá el grafo a dibujar y permitirá crear el grafo a partir de los pesos que introduzca el usuario en la matriz de pesos.

- El usuario tendrá la opción de limpiar por completo el panel de grafos, eliminando todos los nodos y aristas, así como, el panel de información.

- Existirá un manual completo de la aplicación disponible desde la misma interfaz de usario. Además abrá contenido teórico de todos los algoritmos programados que facilitará al alumno (usuario de la aplicación) entender la funcionalidad interna del software.

Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que definen lo que la herramienta de software debe tener en cuanto a apariencia, sensación, operabilidad y mantenimiento.

También pueden ser restricciones o atributos de los servicios y funciones ofertadas por el sistema, se refieren a las propiedades emergentes (atributos) del sistema como la

fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, la capacidad de los dispositivos de entrada/salida, ajuste a estándares, etc.

Los requisitos no funcionales para SWGraphs 2.0 son:

La aplicación se dividirá en las siguientes secciones que posteriormente se detallaran.

- Barra de menús
- Barra de botones
- Paneles
- Barra de estado

Barra de menús

La barra de menús debe contener 5 menús (Archivo, Grafo, Algoritmos, Ejemplos y Ayuda)

El menú Archivo debe tener 5 opciones: Abrir y guardar grafos, Definir y borrar un fondo para el panel de grafos y por último la opción de salir del programa

El menú Grafo debe tener las siguientes opciones: listar el grado de los vértices, obtener el grafo subyacente, obtener la información respecto a la concectividad del grafo, generar un grafo completo, obtener la matriz de adyacencia para trabajar con ella y por último obtener el código del grafo para Mathematica 4.0.

El menú Algoritmos se utilizará para escoger los algoritmos que se podrán ejecutar directamente o por pasos desde otras opciones de este mismo menú o desde las opciones de la barra de botones.

El menú de Ejemplos tendrá como función mostrar grafos ya dibujados para ejecutar y comprobar los algoritmos de la aplicación.

En el menú Ayuda debe aparecer el manual de usuario así como todo el contenido teórico necesario para uso de la aplicación.

Barra de botones

En la barra de botones para el manejo del programa se deberá incluir los siguientes botones:

- Abrir
- Guardar
- Ejecutar
- Por pasos
- K
- Matriz adyacencia
- Borrar paneles
- Ayuda

Paneles

En el menú principal de la aplicación deberán aparecer dos paneles principales. El primero de ellos, que aparecerá a la derecha de la interfaz de usuario, será el panel de dibujo, donde se dibujarán los grafos. En el segundo panel, que deberá aparecer a la izquierda, será donde se visualicen los resultados de la ejecución de los distintos algoritmos que ofrece el software.

Barra de estado

Corresponderá a la barra inferior de la interfaz donde se deberá mostrar el estado actual de la aplicación, última acción ejecutada, fecha, hora, etc.

Otros requisitos no funcionales

La aplicación se podrá instalar sobre la infraestructura existente en las aulas de prácticas de la E.T.S. Ingeniería Informática.

La aplicación debe ser multiplataforma.

Deberá estar desarrollada en el lenguaje de programación JAVA.

El coste computacional de los algoritmos será siempre el menor posible.

La interfaz gráfica debe ser amigable y adaptada al tipo de usuario final.

Se mostrará en la aplicación la información general de la aplicación SWGraphs (versión, autor, ...).

2.2. Conceptos generales previos

A continuación se detalla todel contenido teórico necesario para poder trabajar con la aplicación desarrollada SWGraphs 2.0.

Un **grafo** G es una pareja de conjuntos G = (V, E) donde:

V ≠ ∅ es el llamado conjunto de vértices o nodos

E es un conjunto de pares de elementos de V

Grafo no dirigido (GND):

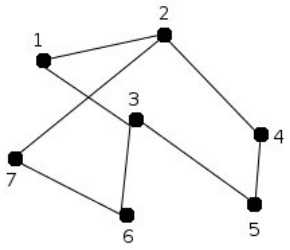
E = conjunto de pares no ordenados de elementos de E (conjunto de **aristas**).

Si (u, v) es una arista se dice que:

u y v son vértices **adyacentes,**

(u, v) es **incidente** en u y v,

si u=v se dice que (u, v) es un **bucle.**



Grafo dirigido (GD):

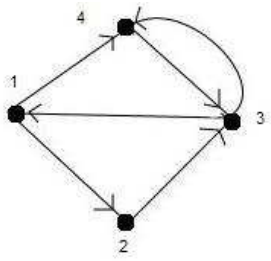
E = conjunto de pares ordenados de elementos de E (conjunto de **arcos**).

Si (u, v) es un arco se dice que:

u es **extremo inicial** de (u, v),

v es **extremo final** de (u, v),

si u=v se dice que (u, v) es un **bucle.**



Si un grafo, dirigido o no dirigido, no tiene bucles se llama **simple**.

Grafo ponderado

Sea G = (V, E), se dice que G es un grafo ponderado si se asigna un valor (peso) a cada una de sus aristas y/o arcos.

Subgrafos

Un grafo H = (V(H), E(H)) se dice que es **subgrafo** de G = (V(G), E(G)), si V(H) es subconjunto de V(G) y E(H) lo es de E(G).

Se dice que H es **subgrafo generador** de G si V(H) = V(G).

Sea G = (V, E), V' ⊂ V, E' ⊂ E

G[V'], grafo inducido por V'

Subgrafo de G cuyo conjunto de vértices es V' y que es maximal con respecto al conjunto de aristas.

G[E'], grafo inducido por E'

Subgrafo de G cuyo conjunto de aristas es E' y su conjunto de vértices está formado por los extremos de las aristas de E'

G-v

Subgrafo de G cuyo conjunto de vértices es el resultado de eliminar de G el vértice v y las aristas incidentes en él

G-e

Subgrafo de G cuyo conjunto de aristas es el resultado de eliminar la arista e del grafo G

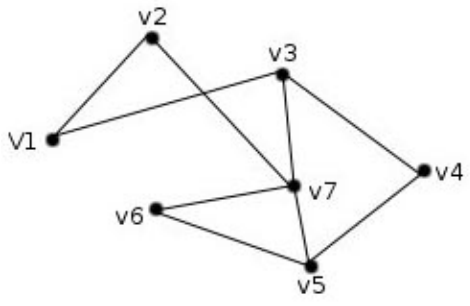
Matriz del grafo

1.- Matriz de adyacencia

Llamamos matriz de adyacencia de G = (V, E) a la matriz nxn, A = (a_{ij}), donde:

$$a_{ij} = \begin{cases} 1 \; si \; (v_i, v_j) \in E \\ 0 \; si \; (v_i, v_j) \notin E \end{cases}$$

Ejemplo:



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

2.- Matriz de pesos

Llamamos matriz de pesos de G a la matriz nxn, A = (a_{ij}), donde cada valor indica el peso de la arista (arco) que van desde el nodo i al nodo j.

3.- Matriz de multiplicidades (Hierholzer)

Llamamos matriz de multiplicidades de G a la matriz nxn, A = (a_{ij}), donde cada valor indica el número de aristas (arcos) que van desde el nodo i al nodo j.

Esta matriz se utiliza en el algoritmo de Hierholzer para especificar los s-grafos.

4.- Matriz de capacidades (Flujo máximo)

Llamamos matriz de capacidades de G a la matriz nxn, A = (a_{ij}), donde cada valor indica la capacidad de la arista (arco) que va desde el nodo i al nodo j.

Esta matriz se utiliza en el algoritmo del flujo máximo para especificar la función capacidad asociada al grafo dibujado.

Alcanzabilidad

Sea G = (V, E) grafo dirigido o grafo no dirigido:

Se dice que el vértice u alcanza al vértice v en G si u = v o existe una cadena de u a v.

Se entiende por tanto que cada vértice se alcanza a sí mismo.

Matriz de Acceso

La matriz de acceso de G es la matriz nxn A = a_{ij}, donde:

$$a_{ij} = \begin{cases} 1, & \text{si el vértice } v_i \text{ alcanza al vértice } v_j \\ 0, & \text{si el vértice } v_i \text{ no alcanza al vértice } v_j \end{cases}$$

Todos los elementos de la diagonal principal de la matriz valen 1 (cada vértice se alcanza a sí mismo).

¿Cómo se obtiene la matriz de acceso?

Sea G=(V,E) grafo, la matriz de acceso de G se puede obtener a partir de la aplicación reiterada de los conocidos métodos:

- Búsqueda en anchura (BFS)
- Búsqueda en profundidad (DFS)

Concretamente, cada una de las filas, i, i = 1, 2, ... , n, de la matriz de acceso se obtiene por aplicación de uno de los dos métodos mencionados al vértice v_i. En el caso de grafos no dirigidos, dado que si u alcanza a v entonces v alcanza a u, bastará con: aplicar el algoritmo a un vértice cualquiera y después ir aplicando reiteradamente el algoritmo a vértices que no hayan aparecido como alcanzados en iteraciones anteriores.

Los algoritmos sirven tanto para grafos dirigidos como grafos no dirigidos**.**

Conectividad

Grafos conexos

Un grafo no dirigido G es conexo si todos los vértices se alcanzan mutuamente.

Caracterización: G es conexo si y sólo si su matriz de acceso está formada sólo por 1's.

Se llama componente conexa de G a todo subgrafo conexo tal que no exista otro subgrafo conexo que lo contenga estrictamente.

Grafos fuertemente conexos

Un grafo dirigido G es fuertemente conexo si todos los vértices se alcanzan mutuamente.

Caracterización: G es fuertemente conexo si y sólo si su matriz de acceso está formada sólo por 1's.

Se llama componente fuertemente conexa de G a todo subgrafo fuertemente conexo tal que no exista otro subgrafo fuertemente conexo que lo contenga estrictamente.

Grafo subyacente de un grafo dirigido

El grafo subyacente del grafo dirigido G, es el grafo no dirigido obtenido al sustituir cada arco (u, v) por una arista (u, v).

Grafo débilmente conexo

Sea G = (V, E) grafo dirigido.

Se dice que un grafo es débilmente conexo si su grafo subyacente G' es conexo.

Se llama componente débilmente conexa de G a todo subgrafo tal que su subyacente sea una componente conexa en G.

ALGORITMOS IMPLEMENTADOS

Algoritmo BFS (Breadth First Search, Búsqueda en Anchura)

El algoritmo BFS se utiliza, entre otras cosas, para obtener la matriz de acceso de un grafo. Este algoritmo es válido tanto para grafos dirigidos como para grafos no dirigidos.

Algoritmo:

Sea G=(V,E) grafo. Considera un vértice de G, v.

1. Crea dos listas vacías L y A, y una cola vacía Q.

2. Elige un vértice v (en SWGraphs lo escoge el usuario), añádelo a L y añade a la cola Q todos los v' tal que (v, v') existe en E

3. Elimina de Q su primer elemento w, añádelo a L (al final) y añade a Q todos los vértices w' tales que (w, w') existe en E y que no estén en L ni en Q.

4. Añade a A una arista (w_o ,w) donde wo es el primer vértice de L tal que (w_o ,w) existe en E

5. Repite los pasos 3 y 4 hasta que Q sea vacía.

Algoritmo DFS (Depth First Search, Búsqueda en Profundidad)

El algoritmo DFS se utiliza, entre otras cosas, para obtener la matriz de acceso de un grafo. Este algoritmo es válido tanto para grafos dirigidos como para grafos no dirigidos.

Algoritmo:

Sea G = (V, E) grafo. Considera un vértice de G, v.

1. Crea dos listas vacías L y A, y una pila vacía P.

2. Elige un vértice v y añádelo a L y a P *.

3. Sea w el elemento más alto de la pila P.

Si existe un vértice w' que no esté en L y (w, w') existe en E , añádelo a P y a L, y añade a A la arista/arco (w, w')

En caso contrario elimina w de P.

4. Repite el paso 3 hasta que la pila P esté vacía.

* Los elementos se ponen en la parte alta de la pila

Algoritmo de DIJKSTRA (1959)

Si G = (V, E) es un grafo ponderado positivo, el algoritmo de Dijkstra proporciona el camino más corto desde cualquier vértice v_0 de V a todos los demás o a uno en concreto. Al seleccionar algoritmo de Dijkstra y pulsar ejecutar, dos menús emergentes nos pedirán el nodo inicial y nodo final.

Restricciones:

Este algoritmo sólo se puede aplicar a grafos con pesos positivos.

Algoritmo:

1.- Se parte de un vértice inicial y se calcula cuál es el más próximo.

2.- Se fija el vértice no fijado aún que está a una menor distancia del inicial.

3.- Se compruebe si utilizando como punto intermedio el último vértice fijado se llega a nuevos vértices o se encuentra un camino más corto a los que ya hemos accedido.

4.- Con la información anterior, se actualizan, si procede, las distancias mínimas desde el inicial a cada vértice y cuál es el vértice anterior en el camino más corto desde el inicial a cada uno de ellos.

5.- Volvemos al paso 2 hasta que todos los vértices estén fijados.

Algoritmo KRUSKAL

Aplicando el algoritmo de Kruskal a un grafo conexo, se obtiene el árbol de expansión mínima, también llamado árbol generador de mínimo coste. En caso de aplicar el algoritmo a un grafo no conexo, se obtendrá un bosque de expansión mínima o bosque generador de mínimo coste.

Si los pesos son distintos dos a dos el árbol/bosque generador de mínimo coste es único.

Restricciones:

Este algoritmo sólo se puede aplicar a grafos no dirigidos.

Conceptos previos:

Se llama **árbol** a todo grafo no dirigido conexo acíclico.

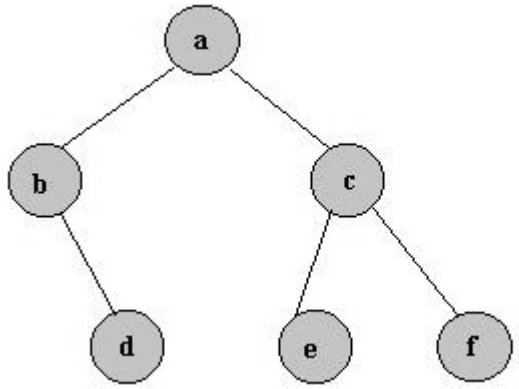
Nota: Recordemos que un bucle es una arista en la que los dos extremos coinciden y que un grafo se dice simple sino posee bucles.

En caso de haber más de un vértice la definición anterior es equivalente a cada uno de los siguientes grupos de condiciones.

Caracterización

Si G es un árbol con más de un vértice los siguientes grupos de condiciones son equivalentes:

- G es árbol
- G es simple y existe un único camino entre cada dos vértices distintos
- G es conexo y |E| = |V| -1
- G es acíclico y |E| = |V| -1



La longitud o coste del árbol es la suma de los pesos de las aristas del árbol.

Árbol generador de mínimo coste

Sea G un grafo no dirigido conexo ponderado G = (V, E).

Se llama árbol generador de G a todo subgrafo generador que sea conexo y acíclico.

Nota: Recordemos que un subgrafo G' de G es subgrafo generador de G si los conjuntos de vértices de G y G' coinciden.

Se llama árbol generador de mínimo coste de G (o de expansión mínima) a todo árbol generador de G cuyo coste sea menor o igual que el de cualquier otro árbol generador de G.

El árbol generador de mínimo coste de un grafo dado G no tiene porqué ser único.

Para determinar el árbol generador de mínimo coste se utiliza el algoritmo de Kruskal.

Algoritmo:

Se trata de un algoritmo sencillo que se aplica según los siguientes pasos:

Paso 1: Se selecciona, de manera arbitraria, cualquier nodo y se conecta (se selecciona la arista) al nodo más cercano (menor coste) distinto de éste.

Paso 2: Se identifica el nodo no conectado más cercano a un nodo conectado, y se conectan estos dos nodos.

Si hay nodos sin conectar, ir al Paso 2.

en otro caso, FIN

Los empates para el nodo más cercano distinto (paso 1) o para el nodo no conectado más cercano (paso2), se pueden resolver de forma arbitraria y el algoritmo conduce a una solución óptima. No obstante, estos empates son indicativos de que pueden existir (aunque no necesariamente) soluciones óptimas alternativas.

Todas esas soluciones se pueden identificar si se analizan las distintas formas de resolver los empates hasta el final.

Algoritmo EDMONDS (2)

Restricciones:

Este algoritmo sólo se puede aplicar a grafos no dirigidos y sin bucles.

Nota: Recordemos que un bucle es una arista en la que los dos extremos coinciden.

Conceptos previos

Sea G = (V, E) grafo no dirigido llamamos:

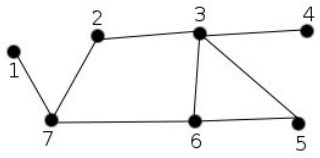
Emparejamiento: subconjunto de E, M, en el que no hay dos aristas que sean adyacentes y ninguna es un bucle.

Vértice M-saturado: vértice que posee una arista de M incidente en él. En caso contrario se dice que el vértice es M-insaturado.

Emparejamiento máximo: emparejamiento tal que no existe ningún otro emparejamiento con mayor número de aristas.

Emparejamiento perfecto: emparejamiento en el que todos los vértices son M-saturados.

Ejemplo:



Sea G el grafo de la figura:

M1 = {(2, 3), (6, 7)} es un emparejamiento.

Los vértices 2, 3, 6, 7 son M1-saturados, los restantes son M1-insaturados.

M1 no es máximo ya que M2 = {(1, 7), (5, 6), (2, 3)} es también emparejamiento y |M1| < |M2|.

M2 es máximo puesto que |V| = 7 y |M2| = 3 pero no es perfecto, porque el número de vértices es impar y por lo tanto siempre habrá un vértice M-insaturado.

En muchas ocasiones, dado un grafo no dirigido ponderado G = (V, E), es interesante conocer no sólo un emparejamiento máximo o perfecto si existe, sino de estos el que más pese, entendiendo por peso del emparejamiento la suma de los pesos de las aristas que lo forman. Por ello introducimos los siguientes conceptos.

Sea G = (V, E) grafo no dirigido y M emparejamiento en G. Llamamos:

Emparejamiento máximo de máximo peso: todo emparejamiento máximo de G tal que el peso de cualquier otro emparejamiento M1 sea menor o igual que el de M. En caso de que el emparejamiento máximo de máximo peso sea además perfecto, se llama emparejamiento óptimo.

Camino M-alternado: camino en G cuyas aristas pertenecen alternativamente a M y E-M.

Camino M-incrementable: camino M-alternado en G cuyos extremos son M-insaturados.

Ejemplo: En el grafo anterior

Es M1-alternado. No es M1-incrementable.

Algoritmo:

El algoritmo de Edmonds (2) obtiene un acoplamiento (emparejamiento) de máxima cardinalidad y peso máximo o mínimo en el grafo dado.

Durante su ejecución se emplean distintas estructuras de la Teoría de Grafos, como son el subgrafo igualdad y el árbol alternado.

En esta aplicación los arboles alternados emplean la siguiente representación:

- Nodos E = Contorno rojo (relleno de rojo si es el nodo E activo en ese paso, en rojo más oscuro se representa el nodo raíz)
- Nodos I = Contorno azul

Opciones:

Maximizar - Minimizar: Permite elegir que emparejamiento (acoplamiento) se buscará, el de máximo peso o el de mínimo peso; esta opción se aplica al iniciar el algoritmo, por lo que cualquier modificación durante el trascurso del mismo no tendrá ningún valor.

Algoritmo de HIERHOLZER

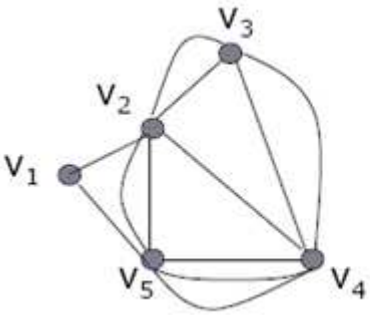
Matriz del grafo

La matriz del grafo, cuando ejecutamos el algoritmo de Hierholzer, es una matriz de multiplicidades, para más información consulta en la ayuda el apartado de Matriz del grafo.

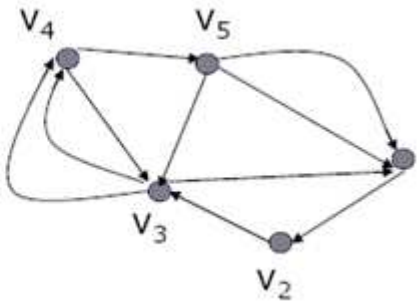
G = (V, E) es un **s-grafo** si es un multigrafo donde el mayor número de aristas (respectivamente arcos) entre dos vértices es s.

Nota: Un multigrafo es un grafo en el que se admiten aristas/arcos múltiples entre dos vértices dados.

Ejemplos:



3-grafo no dirigido



2-grafo dirigido

Cadenas y ciclos eulerianos

Sea G = (V, E) un s-grafo:

Se llama cadena euleriana en G a toda cadena en G que pase por todas y cada una de sus aristas (respectivamente arcos) una sola vez.

Se llama ciclo euleriano en G a toda cadena euleriana cerrada.

Un grafo G se dice que es euleriano si posee un ciclo euleriano.

Grafos no dirigidos

Sea G un s-grafo no dirigido, conexo:

G es euleriano si y sólo si todos los vértices tienen grado par.

G tiene cadena euleriana no cerrada si y sólo si G tiene exactamente dos vértices de grado impar (que serán los extremos de la cadena).

Teorema:

Sea G un s-grafo no dirigido, conexo:

G es euleriano si y sólo si se puede expresar como unión de ciclos aristodisjuntos.

Grafos dirigidos

Sea G un s-grafo dirigido, débil conexo:

G es euleriano si y sólo si todos los vértices tienen el mismo grado de entrada que de salida.

G tiene cadena euleriana no cerrada si y sólo si existen dos vértices u y v tal que

a) $d_e(u) = d_s(u) + 1$ y $d_s(v) = d_e(v) + 1$ y

b) todos los vértices distintos de u y de v tienen el mismo grado de entrada y salida.

Hierholzer:

Sea G un s-grafo dirigido, débil conexo:

G es euleriano si y sólo si se puede expresar como unión de ciclos aristodisjuntos.

Para obtener un ciclo euleriano tanto en grafos no dirigidos como en grafos dirigidos, podemos utilizar diferentes algoritmos. En esta aplicación, se ha implementado el algoritmo de Hierholzer.

Algoritmo de Hierholzer (1873)

Válido tanto para grafos dirigidos como no dirigidos.

1.- Se fija un vértice v y se busca un ciclo en el que esté contenido. Para ello se construye una cadena a partir de v añadiendo aristas/arcos hasta que volvamos a v otra vez. (Ninguna arista/arco se puede añadir más de una vez)

Si no quedan más aristas/arcos hemos terminado.

2.- Si quedan más, tomamos un vértice del ciclo existente que sea adyacente a aristas/arcos no considerados, por ejemplo w, y seguimos añadiendo aristas/arcos como antes hasta que lleguemos nuevamente a w.

3.- Las aristas/arcos del ciclo hallado a partir de w se intercalan en el ciclo anterior en la posición que ocupaba w.

Si no quedan más aristas/arcos por añadir hemos terminado; si no es así, entonces volver al paso 2.

Algoritmo del CARTERO CHINO

Restricciones:

Este algoritmo sólo se puede aplicar a grafos no dirigidos, sin bucles y con pesos positivos.

Nota: Recordemos que un bucle es una arista en la que los dos extremos coinciden.

Algoritmo:

El algoritmo Cartero Chino consiste en buscar un recorrido de mínimo peso que recorra todas las aristas del grafo, comenzando y terminando en el mismo vértice.

En general, este algoritmo hace uso de tres algoritmos: Edmonds (2), Fleury (o Hierholzer) y Dijkstra aunque en algunos casos no será necesario emplear Edmonds (2) o Dijkstra.

Para más detalle ejecute el algoritmo por pasos.

Panel camino del Cartero Chino: Muestra gráficamente y de forma lineal el resultado del algoritmo.

Algoritmo de FORD-FULKERSON

Restricciones:

Este algoritmo sólo se puede aplicar a grafos dirigidos débil conexos.

Conceptos básicos:

Una red N(G, s, t, c) es una cuaterna definida de la siguiente forma:

Red N(G, s, t, c)

donde:

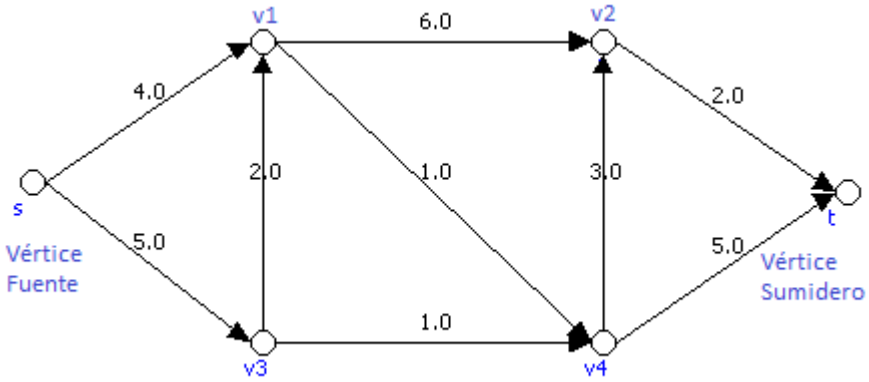
G -> Grafo dirigido débil conexo

s -> vértice de G tal que su grado de salida sea mayor que 0 (vértice fuente)

t -> vértice de G tal que su grado de entrada sea mayor que 0 (vértice sumidero)

c -> función c: E(G) -> N U {0} (función capacidad)

Ejemplo:



Flujo f en N(G, s, t, c)

Se llama **flujo** f **en** N a una función f: E(G) -> N U {o} que verifica las siguientes condiciones:

Se llama **flujo** f **en** N a una función f: E(G) -> N U {o} que verifica las siguientes condiciones:

a) $\forall u \varepsilon E(G)$ o β f(e) β c(e) (limitación por capacidad)

b) $\forall u \varepsilon V-\{s, t\}$ $\Sigma_{v\varepsilon\Gamma(u)} f(u, v) = \Sigma_{v\varepsilon\Gamma^{-1}(u)} f(v, u)$ (ecuación de conservación)

Propiedad:

Así como la capacidad es una característica de cada red, el flujo no lo es, y un flujo dado puede ser modificado. El flujo más sencillo es f(e) = o para todo e que pertenece a E(G).

Flujo "de" la red

Dada la red N(G, s, t, c) se llama **flujo de la red** al valor f(N) definido como

$f(N) = \Sigma_{v\varepsilon\Gamma(s)} f(s, v) - \Sigma_{v\varepsilon\Gamma^{-1}(s)} f(v, s)$

Propiedad:

Se comprueba que

$f(N) = \Sigma_{v\varepsilon\Gamma^{-1}(t)} f(v, t) - \Sigma_{v\varepsilon\Gamma(t)} f(t, v)$

Nota: El flujo, f, representa lo que se transporta en la red.

Grafo residuo de G

Grafo Gr = (Vr, Er) con Vr) V, Er = E U {(u, v) E} y función asociada r: Er --> N U {0} donde:

$$r(e) = \begin{cases} c(e) - f(e) \; si \; e \in E \\ f(e) \; si \; e \notin E \end{cases}$$

Camino f-incrementable en Gr

Se llama **camino f-incrementable** de s a t, a todo camino P de s a t en el grafo residuo Gr que verifique r(e) > 0.

Se denota Δ = min { r(e) donde e ϵ E(P) }

Nota: Observa que los caminos que hemos utilizado para localizar en Gr el flujo máximo se llama f-incrementable.

Teorema:

Sea f flujo en la red N(G, s, t, c)

f es flujo máximo en N <--> No existe camino f-incrementable

Algoritmo de Ford-Fulkerson:

inicializar flujo f=0;

construir grafo residual G_f asociado a f = 0;

maxflow = 0;

while (existe camino de aumento de s a t en G_f) {

  delta = menor capacidad del camino de aumento;

  maxflow += delta;

  para cada arista (u, v) del camino de aumento hacer {

    c_f(u, v) -= delta;

    c_f(v, u) += delta;

  }

}

return maxflow;

Algoritmo CICLOS HAMILTONIANOS

Conceptos previos

Sea G = (V, E) un grafo no dirigido, |V| = n

Camino hamiltoniano: todo camino P en G que contenga a todos los vértices del grafo.

Ciclo hamiltoniano: todo ciclo C de G que contenga todos los vértices de G.

Grafo hamiltoniano: grafo que contiene un ciclo hamiltoniano.

1.- Ciclo hamiltoniano de 'bajo' peso

Sea G=(V,E) grafo no dirigido.

Se dice que CHB es un ciclo hamiltoniano de **bajo peso** en G si es un ciclo hamiltoniano tal que

p(CHB) ≤ 2 p(CHM) donde CHM es un ciclo hamiltoniano de mínimo peso.

Tanto el algoritmo basado en el árbol generador de mínimo coste, como el algoritmo voraz, permiten obtener un ciclo con estas características. Pero hay que tener en cuenta que para que sea posible su aplicación el grafo debe cumplir las siguientes condiciones:

- G = (V,E) completo, |V| >= 3, ponderado positivo
- G verifica la desigualdad triangular, es decir, $p(v_i, v_j) \leq p(v_i, v_k) + p(v_k, v_j)$

Restricciones:

Estos algoritmos sólo se pueden aplicar a grafos no dirigidos, simples, con pesos positivos y con al menos 3 nodos para que pueda existir un ciclo hamiltoniano.

a) Algoritmo basado en el árbol generador

1.- Obtener el árbol generador de mínimo coste (T)

Aplicar Kruskal a G para obtener T

2.- Encontrar ciclo euleriano dirigido CE

2.1.- Considerar el árbol como un grafo dirigido (por cada arista (x, y) con peso p(x, y) de G, se consideran dos arcos (x, y) e (y, x) ponderados, ambos, con p(x, y)

2.2.- Encontrar un ciclo euleriano para el grafo dirigido

Aplicar algoritmo de Hierholzer

3.- Obtener ciclo hamiltoniano a partir de CE, eliminando los nodos repetidos del ciclo euleriano.

Se obtiene un ciclo hamiltoniano de bajo peso

b) Algoritmo voraz

(en grafos.m, paquete para Mathematica 4.0: ViajanteBajo[G])

1.- Seleccionar cualquier vértice de G para formar el ciclo hamiltoniano.

2.- Buscar de entre todas las aristas incidentes en el vértice seleccionado, la de menor peso.

3.- Añadir el nuevo vértice al ciclo hamiltoniano.

3.1.- Si el número de vértices del ciclo coincide con el número de vértices del grafo, Fin.

3.2.- Si el número de vértices del ciclo es menor al número de vértices del grafo, volver a 2.

2.- Ciclo hamiltoniano de 'mínimo' peso

Algoritmo por fuerza bruta

1.- Buscar todos los ciclos hamiltonianos posibles en el grafo.

2.- Obtener el peso de cada uno de los ciclos.

3.- Devolver el ciclo hamiltoniano de mínimo peso.

3. Diseño

3.1. Introducción

Según Pressman, el diseño del software es en realidad un proceso de muchos pasos que se clasifican dentro de uno mismo. En general, la actividad del diseño se refiere al establecimiento de estructuras de datos, la arquitectura general del software, representaciones de interfaz y algoritmos. El proceso de diseño traduce requisitos en una representación de software.

El diseño de la aplicación se va a realizar a través del estudio de diagramas de clases ya que permite capturar adecuadamente los requerimientos descritos en la fase de análisis del proyecto.

Un diagrama de Clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Nos sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de convencimiento. Un diagrama de clases está compuesto por los siguientes elementos: Clase: atributos, métodos y visibilidad; Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

Se ha decidido utilizar este tipo de diagrama porque muestra una visión orientada a objetos del problema planteado y nos sirve de ayuda en la futura fase de implementación, en la cual, si recordamos, utilizaremos un lenguaje de desarrollo orientado a objetos como es JAVA.

Para la representación del diseño de la aplicación se han utilizado tres diagramas de clases.

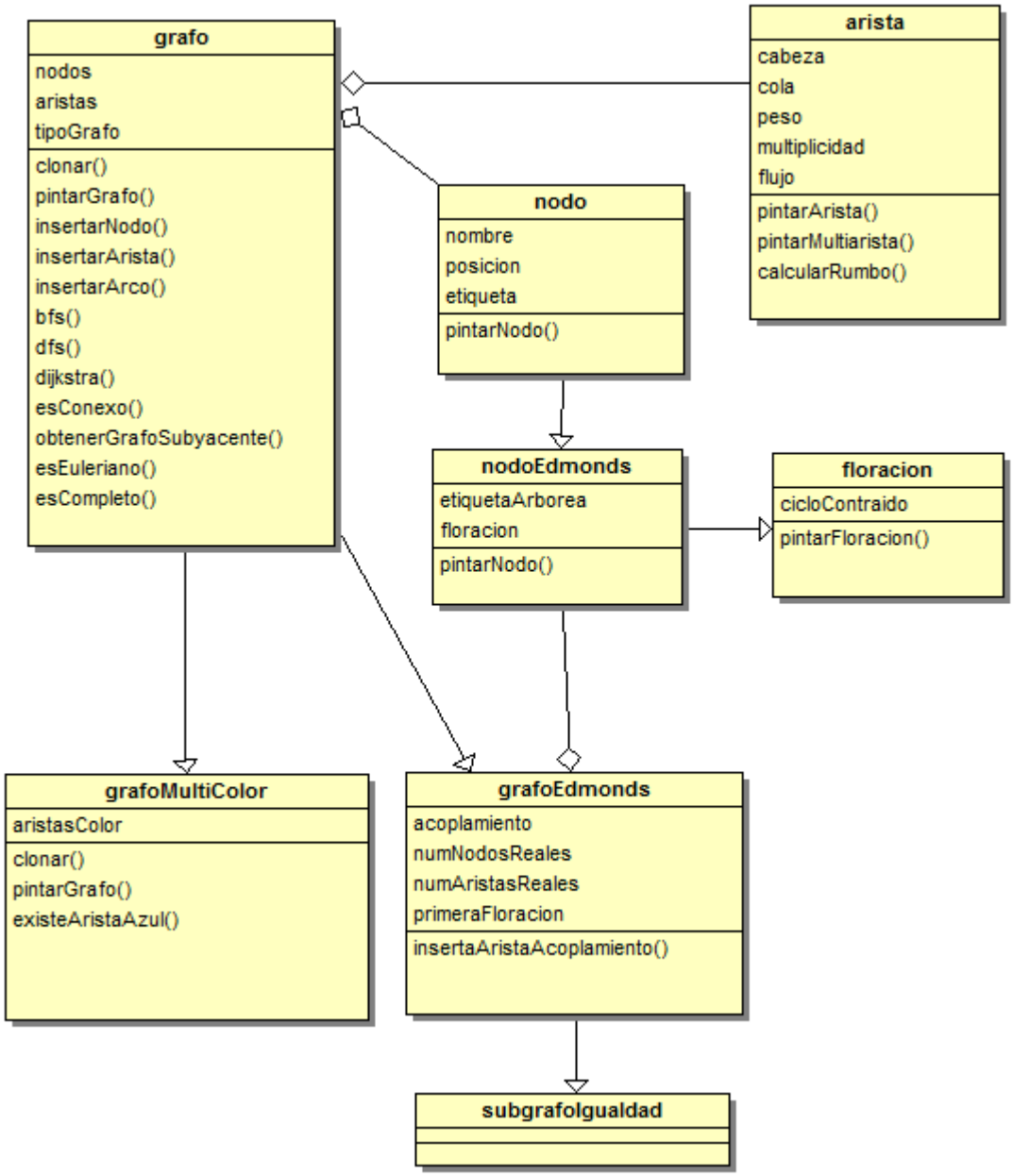
El primer diagrama de clases representa la relación entre las distintas estructuras utilizadas para representar los objetos que utiliza la teoría de grafos para la resolución de problemas (grafos, nodos, aristas,...).

En el segundo diagrama se representan las clases que se utilizarán para resolver los algoritmos que se deben poder ejecutar en la aplicación. En él se puede observar la clase Algoritmo de la cual heredan los otros algoritmos específicos.

En tercer y último lugar se especifican las clases correspondientes al diseño de la interfaz gráfica de usuario, en ella se puede observar la ventana principal, sus componentes y los demás cuadros de diálogo utilizados en la aplicación.

A continuación se muestran y se detallan los tres diagramas de clases mencionados.

3.2. Clases estructurales



Grafo

La clase grafo, será una de las principales clases de la aplicación debido a la importancia del objeto que se representa y por los métodos que se deben implementar en esta clase.

Un grafo es una colección de nodos unidos mediante una colección de aristas, por ello la clase grafo tiene un atributo nodos y un atributo aristas. El atributo tipoGrafo se utilizará para especificar si el grafo es dirigido o no dirigido.

Se puede observar en el diagrama que en la clase grafo se han implementado los algoritmos BFS, DFS y Dijkstra. Se ha preferido implementar los algoritmos en la clase grafo porque son reutilizados en otros algoritmos más complejos.

Nodo

Las características principales de la representación de un nodo para la aplicación se pueden observar en el diagrama de clases.

Debe admitir un nombre que lo identifique dentro del grafo. Por defecto la aplicación los nombrará como v_1, v_2,... aunque ese nombre podrá ser cambiado por el usuario cuando lo desee.

De soportar una etiqueta numérica para los algoritmos de Edmonds (2) y Cartero Chino.

Deberá tener un atributo donde se guarde la posición que ocupa dentro del panel de grafos para que al abrir un grafo, previamente guardado, se dibuje tal y como espera el usuario.

Se puede observar que la clase nodo tiene una relación de agregación con la clase grafo. Esta relación se emplea cuando se desea modelar una relación “Parte de”.

“Un grafo está formado por nodos”

Arista

Una arista representa la unión entre dos nodos del grafo. Denominamos a estos nodos como cabeza y cola de la arista. Se exige que todas las aristas del grafo tengan asignado un peso definido. También se podrá especificar la multiplicidad de la arista para ciertos algoritmos que trabajan con grafos Eulerianos. Además, se podrá indicar el flujo de la arista para el algoritmo de Ford-Fulkerson.

Se puede observar que la clase arista tiene una relación de agregación con la clase grafo. Esta relación se emplea cuando se desea modelar una relación “Parte de”.

“Un grafo está formado por aristas”

Grafo multicolor

La clase grafo multicolor es una especialización de la clase grafo. En esta clase se añade un atributo que representa las aristas del grafo que forman parte de la solución del algoritmo ejecutado y que deberán pintarse de un color diferente para que el usuario vea el resultado por pantalla.

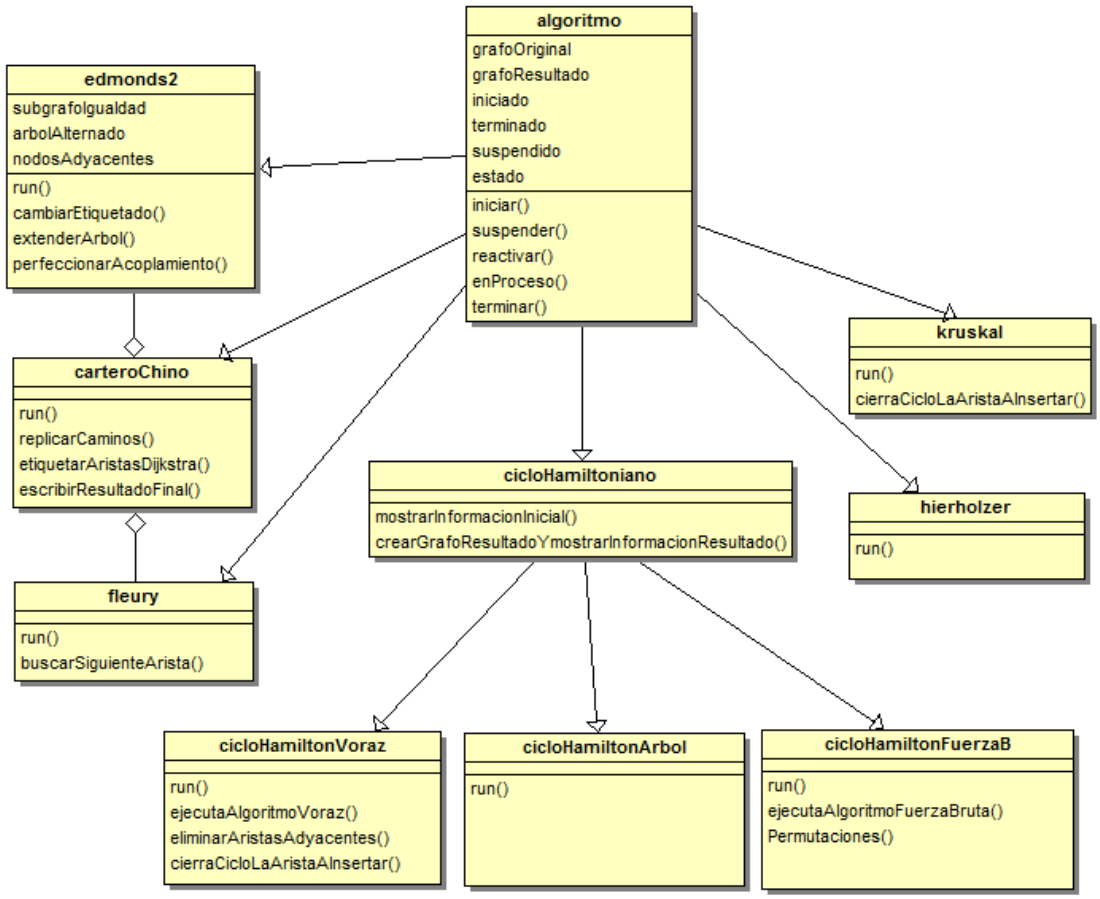
Grafo edmonds

El grafo edmonds es una especialización de la clase grafo que se utiliza para implementar el algoritmo de Edmonds (2). Se debe crear una clase nueva porque tiene atributos que para el resto de grafos utilizados no tendrían sentido.

Subgrafo igualdad

Un subgrafo no necesitaría una nueva clase ya que sus características estructurales son idénticas a la de un grafo. Pero el subgrafo igualdad tiene dos grupos de aristas relevantes, por un lado las aristas que forman el subgrafo por razones obvias, y por otro el subconjunto C de aristas que no están contenidas en el subgrafo, pero que son necesarias para el cálculo en una situación de cambio de etiquetado.

3.3. Clases algoritmicas



Algoritmo

La clase Algoritmo es una clase abstracta que contiene atributos y métodos genéricos que se implementarán en las clases que heredan de ella. Es una clase que no posee instancias, pero sus clases descendientes sí.

Una clase abstracta es una clase que tiene al menos una operación sin código. Estas clases no pueden instanciarse ya que poseen operaciones sin definir.

Las clases abstractas se utilizan para definir operaciones que serán heredadas por sus subclases. Proporcionan el protocolo de la operación (interface) sin proporcionar el método correspondiente.

Cabe destacar los atributos grafoOriginal y grafoResultado. El primero de ellos es de tipo Grafo y es el grafo dibujado por el usuario que será objeto de estudio y al que se le aplicará el algoritmo seleccionado. El segundo atributo es de tipo GrafoMultiColor y es el que mostrará el resultado de aplicar un algoritmo determinado con la solución en color azul.

Ciclo hamiltoniano

A partir de un conjunto de clases, si estas tienen en común una serie de atributos y operaciones, por generalización se puede crear una versión más general (superclase) de las clases iniciales (subclases).

Los atributos y operaciones comunes a estas se sitúan en la superclase y son compartidos por todas las subclases.

Esta es la técnica utilizada para los algoritmos que resuelven problemas de ciclos hamiltonianos. Se ha creado la superclase cicloHamiltoniano, con las características comunes a las clases cicloHamiltonVoraz, cicloHamiltonArbol y cicloHamiltonFuerzaB.

Cartero Chino

Destacamos la clase Cartero Chino, ya que se puede observar en el diagrama de clases que tiene dos relaciones de agregación con las clases fleury y edmonds2. Estas relaciones significan que para la implementación del algoritmo del Cartero Chino se han utilizado los algoritmos Fleury y Edmonds (2).

Otras clases algorítmicas

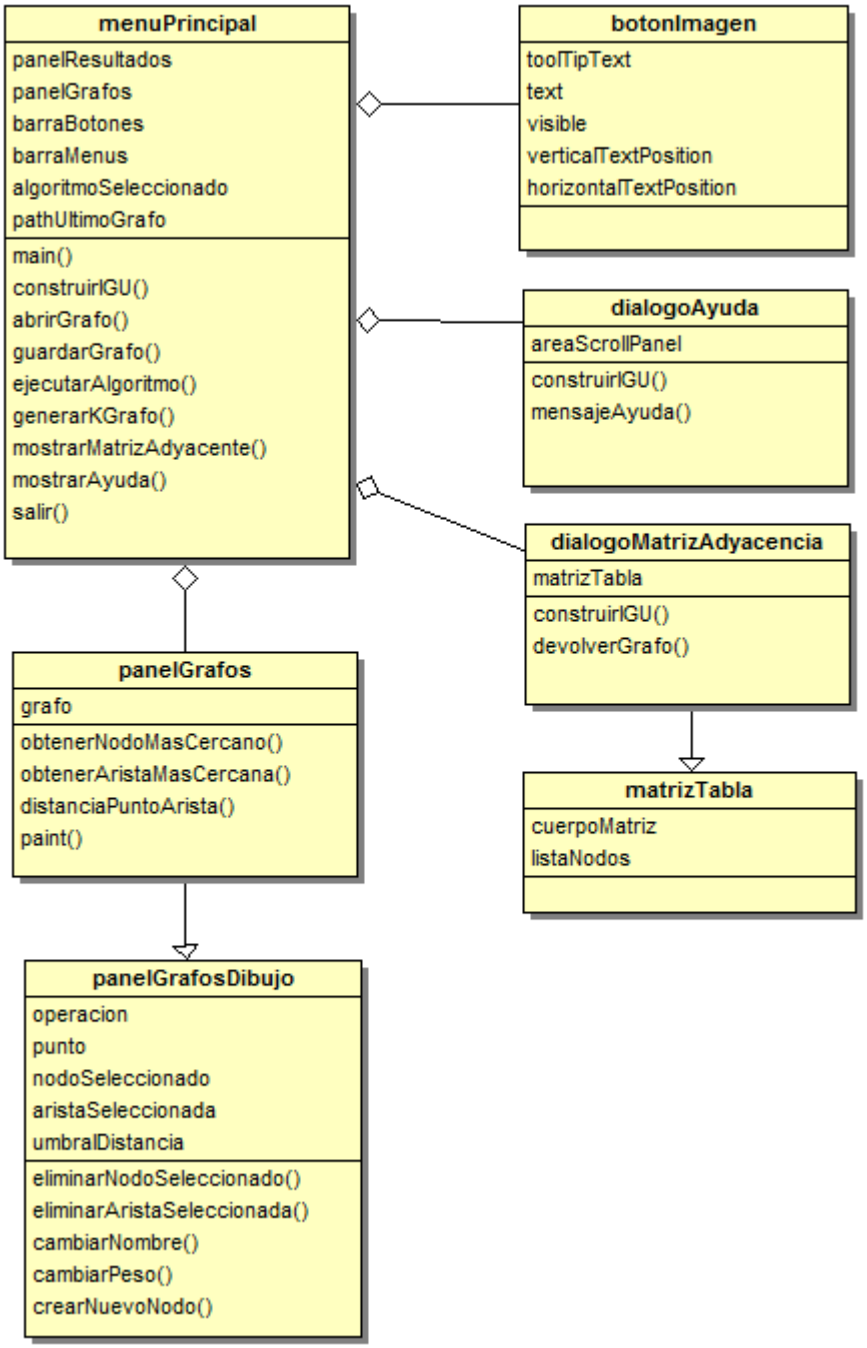
La herencia es un mecanismo de reutilización de código que permite a los programadores crear nuevas clases a partir de clases existentes

Es una relación transitiva entre clases que permite a una nueva clase utilizar los métodos y atributos definidos en otra clase como si fuesen propios. Las subclases heredan los atributos y las operaciones definidas en la superclase, pudiendo añadir atributos y operaciones propios.

La relación de especialización se emplea en la fase de modelado de un sistema, mientras que la relación de herencia se ve como un mecanismo de reutilización de código en la fase de implementación o diseño.

Esta es la técnica utilizada para las clases del resto de algoritmos: Kruskal, Hierholzer, Edmonds (2),..

3.4. Clases interfaz



Menú principal

La clase menuPrincipal es la que contiene la función main del programa, es decir, es la primera que se ejecuta al iniciar la aplicación. En dicha clase, se especifica la interfaz gráfica de usuario.

Como se puede observar en el diagrama de clases, la clase menuPrincipal contiene todos los métodos relacionados con las funcionalidades que ofrece la aplicación: abrir y guardar grafos, ejecutar algoritmos, mostrar matriz de adyacencia, etc.

Se puede observar que las clases panelGrafos, botonImagen, dialogoAyuda y dialogoMatrizAdyacencia tienen una relación de agregación con la clase menuPrincipal. Esta relación se emplea cuando se desea modelar una relación "Parte de": El menuPrincipal está formado por panelGrafos, botonImagen, dialogoAyuda y dialogoMatrizAdyacencia.

Panel grafos

La clase panelGrafos se utiliza para representar los caminos de los grafos resultados de aplicar los algoritmos del Cartero Chino y de Ford-Fulkerson. La característica principal de este panel es el usuario no puede hacer ninguna modificación sobre los grafos representados en él. Estos grafos sólo pueden ser modificados por la aplicación tras ejecutar los algoritmos mencionados.

Panel grafos dibujo

SWGraphs dispone de un panel donde el usuario podrá dibujar sus propios grafos para la resolución de problemas de teoría de grafos. Dicho panel se implementará utilizando la clase panelGrafosDibujo.

Botón imagen

La clase botonImagen se utiliza para añadir los botones de la interfaz gráfica de usuario. Se ha decidido crear una clase específica para que todos los botones tengan las mismas características y apariencia en la aplicación.

Diálogo ayuda

La clase dialogoAyuda se ha creado para mostrar tanto el manual de usuario como toda la ayuda disponible sobre teoría de grafos, que está disponible a través del menú Ayuda de la aplicación.

Diálogo matriz de adyacencia

La clase dialgoMatrizAdyacencia se utiliza para mostrar las distintas representaciones de las matrices de los grafos representados en el panel de grafos. En este cuadro de diálogo se representarán las matrices de pesos, de multiplicidades, etc.

Matriz tabla

La clase matrizTabla se utiliza en el cuadro de diálogo de matriz de adyacencia para mostrar la tabla que representan dichas matrices. En ella se puede modificar los pesos de las aristas de los grafos, las multiplicidades de las aristas, etc.

4. Implementación

4.1. Introducción

La implementación es la parte del proceso en el que los ingenieros de software programan el código para el proyecto tomando como punto de partida el modelo del diseño especificado de la fase anterior.

La herramienta de desarrollo que se ha escogido es Eclipse Ganymede versión 3.4.2 y el lenguaje de programación JAVA.

Se ha diseñado una interfaz básica sobre la que probar los algoritmos, comenzando por los pasos básicos en el tratamiento de grafos (definición del grafo mediante dibujo de vértices y aristas, etiquetado de aristas, etiquetado de vértices…) y continuando por resolver, previa modelización, los distintos problemas de teoría de grafos propuestos en las prácticas de las asignaturas.

Tras comprobar el correcto funcionamiento de todos los algoritmos implementados se incluyen funcionalidades adicionales, como la ejecución por pasos en dos de ellos, vistas de la construcción de los grafos en los resultados, etiquetado mediante matriz de adyacencia, guardar y abrir los grafos en formato XML, guardar el grafo resultado, definir una imagen de fondo en el panel de dibujo de grafos, etc.

Durante la implementación se han ido realizando una serie de prototipos que la directora del proyecto ha ido revisando y validando.

Se ha decidido, por razones de tamaño y cantidad de líneas de código, no añadir por escrito en la presente memoria la codificación en JAVA del proyecto. En cualquier caso, a pesar de no incluir el código sí que hay varios aspectos a destacar de la fase de implementación.

En relación a las dificultades técnicas que se ha encontrado a la hora de desarrollar el proyecto, cabe destacar el interés por bajar el coste computacional de los algoritmos implementados. Aunque los algoritmos ya están definidos, a la hora de implementarlos en un lenguaje de programación, se ha de tener en cuenta el uso óptimo de las estructuras de datos, las instrucciones de repetición, los bucles, etc, para que el tiempo de ejecución sea el mínimo posible.

En cuanto a la codificación de los algoritmos, se han seguido una serie de buenas prácticas a la hora de programar a fin de que el código fuente sea fácilmente ampliable y entendible por cualquier programador. Entre ellas cabe destacar:

- La planificación y organización previa.
- La limpieza del código presentado como se puede observar en el código perfectamente tabulado, los saltos de línea y espacios en blanco que se han dejado.

- Algo que hace muy entendible el código fuente, son los comentarios añadidos en todas las funciones que permiten saber que está pasando en la aplicación en cada momento.

- La utilización de versiones, ya que es una buena técnica para realizar copias de seguridad y permiten volver a una versión anterior en la que todo funciona correctamente en el caso de realizar un cambio peligroso.

Otro detalle técnico a tener en cuenta es que se han externalizado todas las cadenas de texto que aparecen en la aplicación, para que en un futuro, sea fácilmente traducido a cualquier idioma.

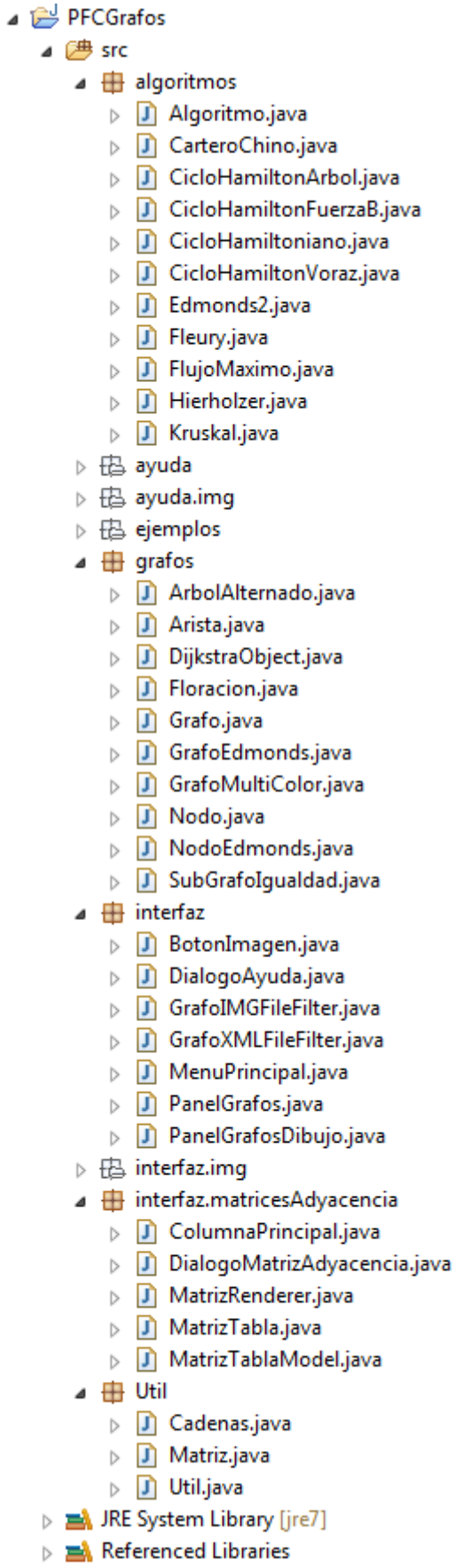
4.2. Organización de las clases

La organización de las clases utilizadas en la codificación del proyecto se ha realizado utilizando los paquetes que el entorno de trabajo Eclipse facilita.

Un paquete de clases es un grupo de clases que agrupamos porque consideramos que están relacionadas entre sí, o bien por su finalidad, por ámbito, por herencia o porque tratan de un tema común.

Además, los paquetes resuelven el problema del conflicto entre los nombres de las clases. Al crecer el número de clases crece la probabilidad de designar con el mismo nombre a dos clases diferentes. Añadir todas las clases en un mismo directorio no suele ser lo más elegante. Es mejor hacer grupos de clases de forma que todas las clases que traten de un determinado tema o estén relacionadas entre sí vayan juntas.

En SWGraphs se han construido seis paquetes.



1.- El paquete **algoritmos** engloba la clase Algoritmo y todas las clases que heredan de ella, en estas clases se implementan la mayoría de algoritmos implementados en la aplicación.

2.- El paquete **ayuda** contiene los archivos HTML que contienen el manual de usuario de la aplicación así como el contenido teórico que ayuda al usuario a comprender como funcionan los algoritmos. Dentro del paquete ayuda se ha añadido otro paquete con las imágenes que se utilizan en todas los ficheros .html de la ayuda y manual de usuario.

3.- El paquete **ejemplos** contiene los archivos XML que contienen los grafos de ejemplo que se utilizaran para ver y entender el funcionamiento de los algoritmos.

4.- El paquete **grafos** contiene las clases de tipo estructural y una clase utilizada para el mantenimiento de los resultados de la aplicación de Dijkstra.

5.- El paquete **interfaz** abarca todas las clases relacionadas con la interfaz gráfica de usuario. Además dentro de este paquete se han creado otros dos que contienen por un lado las clases relacionadas con el dialogo de matriz de adyacencia y, por otro, todas las imágenes usadas para los botones de la aplicación.

6.- El paquete **Útil** contiene la clase cadenas que a su vez contiene todas las palabras que aparecen en la aplicación, la clase Matriz que permite realizar diferentes operaciones de matrices y una clase denominada Util. En la clase Util se implementan los métodos que son usados por más de una clase. Es el caso de los métodos de lectura de archivos con grafos y también de la escritura del código XML del grafo en un archivo.

4.3. La clase Main

Toda aplicación informática consta de una serie de instrucciones que se van ejecutando, desde el principio hasta el final. Según el lenguaje de programación que estemos utilizando, la forma en la que dicho programa empieza, acaba o se va ejecutando puede variar mucho, esto depende en gran medida del paradigma que utilice dicho lenguaje.

JAVA es un lenguaje orientado a objetos y como tal hace uso de unas estructuras llamadas clases, que no es más que un almacén de datos y procedimientos que dicha clase es capaz de hacer. Pero, aunque en JAVA todo el código vaya en varias clases, exceptuando algunas sentencias como los includes o los packages, existe un método especial llamado Main que será el punto de partida de nuestro programa; dentro de este método ya se podrán hacer las llamadas a todas las rutinas que compondrán nuestra aplicación software.

Al usuario de este proyecto se le entrega un único archivo .jar, con punto de entrada en la clase Main.

Las características de este método son:

- Debe ser ***public:*** Es decir, se le puede llamar desde cualquier parte de la aplicación.
- Debe ser ***static***: Se le puede llamar sin necesidad de instanciar la clase (hay que ponerlo para que el código nos quede más simple y porque el Main obliga a ponerlo).
- Es ***void***: Este método no devuelve ningún valor como resultado.
- El parámetro del método, `String[] args`, que va entre paréntesis detrás de él, es un array de String (cadenas de texto). Como no se va a usar, no se detalla exáctamente el significado de esto.

El método de nuestro proyecto quedará de la siguiente manera:

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

	SwingUtilities.invokeLater(new Runnable() {

		public void run() {
			MenuPrincipal inst = new MenuPrincipal();
			inst.setLocationRelativeTo(null);
			inst.setVisible(true);
		}
	});
}
```

donde:

```
MenuPrincipal inst = new MenuPrincipal();
```

Es la instrucción donde se crea la instancia de la clase MenuPrincipal. En el constructor de esta clase se llama al método construirIGU donde se crean todos los componentes que forman la interfaz gráfica de usuario para su posterior visualización por pantalla.

```
SwingUtilities.invokeLater
```

Este método debe ser utilizado cuando un subproceso de aplicación necesita actualizar la IGU (interfaz gráfica de usuario).

```
inst.setLocationRelativeTo(null);
```

Establece la posición de la ventana relativa a un componente pasado como parámetro. Si se le pasa null como parámetro la ventana se posiciona en el centro de la pantalla.

```
inst.setVisible(true);
```

Una vez hemos instanciado la clase deberemos de hacer visible la ventana. Para lo cual invocaremos el método setVisible pasando como parámetro true.

4.4. La interfaz gráfica de usuario

La intención del diseño visual no es que una aplicación sea lo más "bonita" posible sino que esté centrada en la comunicación y la usabilidad para el usuario final.

Para el diseño de la Interfaz Gráfica de Usuario (IGU) se ha seguido una serie de pautas de diseño y unos criterios de usabilidad para conseguir un diseño orientado al usuario que facilite su uso y comprensión de una manera rápida.

La estructura de la organización de los elementos en todas las ventanas de una aplicación debe ser constante, por lo que se ha estandarizado todos los elementos como presentación de menús, botones de comandos, etiquetas, etc.

Se han utilizado un conjunto limitado de colores; colores apagados, sutiles y complementarios que suelen ser los más apropiados en el diseño de interfaces para aplicaciones de tipo empresarial y académicas.

Teniendo todo lo anterior en cuenta hemos dividido la interfaz en tres partes:

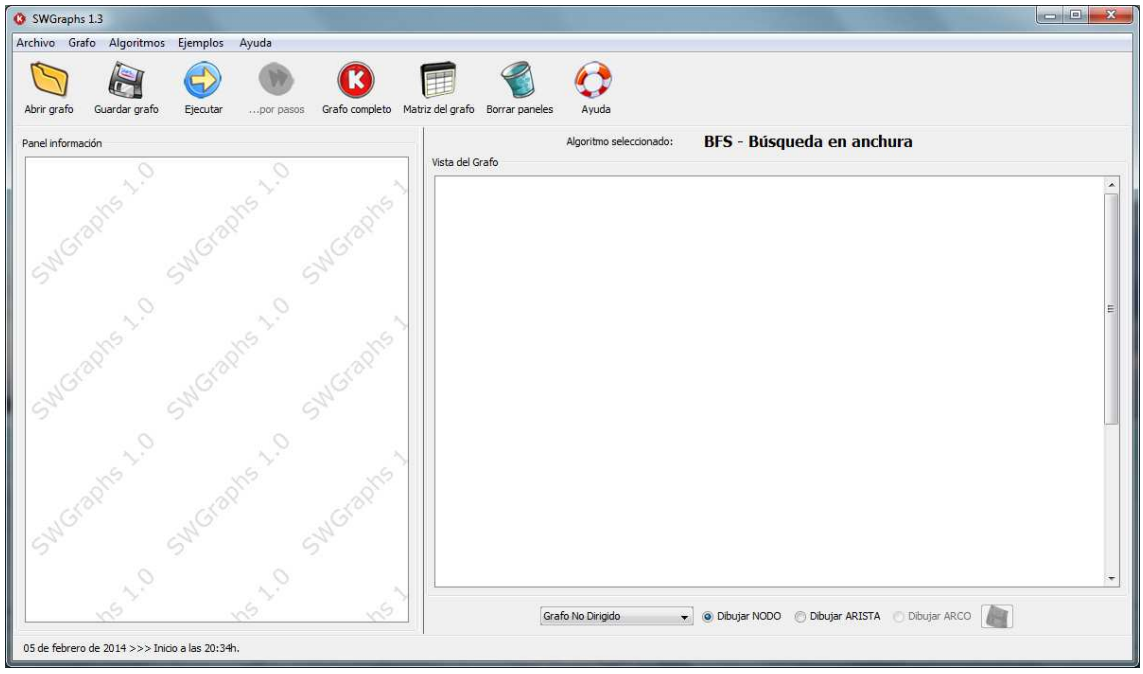
1.- En la parte superior aparecen los menús y los botones para interactuar con el usuario de una forma clara y ordenada.

2.- A la izquierda aparece el panel de resultados donde el usuario podrá ver de forma precisa la información que le ofrece la aplicación en cuanto a resultados se refiere.

3.- Y, por último, a la derecha de la interfaz hemos dejado el panel para dibujar los grafos que serán objeto de estudio.

Además, la barra inferior muestra información sobre el estado actual de la aplicación.

El resultado, es la interfaz clara y sencilla que se muestra a continuación:



5. Pruebas

Antes de entregar el proyecto, se han realizado una serie de pruebas con el objetivo de valorar, por una lado la funcionalidad de la aplicación y por otro la seguridad y estabilidad del sistema. Las pruebas que hemos realizado se basan en la introducción de datos en la aplicación una vez finalizada. Comprobaremos que los algoritmos y la funcionalidad de la interfaz funcionan correctamente.

Hemos dividido las pruebas en dos partes:

- Pruebas positivas: basadas en la introducción de datos correctos en el sistema con el objetivo de comprobar los resultados esperados, que todo funcione correctamente.

- Pruebas negativas: introduciendo datos incorrectos (valores no permitidos en los pesos de los grafos, grafos que no cumplen las condiciones para ejecutar ciertos algoritmos, etc.). El sistema ofrece un mecanismo de protección que realiza una serie de comprobaciones y responde con una pantalla de aviso notificando al usuario que los datos introducidos no son correctos.

Además, durante el primer cuatrimestre del curso 2013-2014 la aplicación se ha instalado en los laboratorios del departamento de Matemática Aplicada de la UPV dónde se ha utilizado en la realización de las prácticas de Matemática Discreta, siendo puesta a prueba por más de 300 alumnos y 8 profesores.

Cabe destacar que sólo han surgido un par de detalles en el funcionamiento que se han solucionado y se han añadido a la última versión de la aplicación entregada. Además se han ido añadiendo utilidades a propuesta de los profesores y alumnos que han utilizado el programa para terminar ofreciendo un producto de calidad que se adapta perfectamente a las necesidades de la asignatura.

6. Instalación

La instalación de la aplicación en un ordenador no requiere de ninguna característica hardware mínima para ser instalada.

El único requisito que debe cumplir cualquier ordenador en el que se instale la aplicación es que debe tener instalado la Máquina Virtual Java. A continuación se deja el enlace para la descarga gratuita:

https://www.java.com/es/download/

JAVA es un lenguaje multiplataforma lo que significa que el mismo código compilado puede ser ejecutado en distintas plataformas (Windows, Linux, Mac y todas aquellas plataformas para las que exista una Máquina Virtual Java) sin necesidad de volver a escribirlo ni compilarlo.

El funcionamiento del programa JAVA es el mismo en todas las plataformas y sólo cambia la apariencia que se adapta a la del sistema operativo que lo ejecuta (Windows, Linux, Mac, etc.)

SWGraphs no requiere de ninguna instalación típica en el sistema operativo. Se presenta como un archivo .JAR (por sus siglas en inglés, Java ARchive) el cual es un tipo de archivo que permite ejecutar la aplicación desarrollada en el lenguaje JAVA de forma directa, es decir, es un archivo ejecutable, sólo hay que copiar el archivo.

SWGraphs se ha instalado en los laboratorios de la Escuela Técnica Superior de Informática de la UPV así como en todos los ordenadores accesibles a los alumnos de la ETSINF en las distintas aulas de acceso libre.

Además, la aplicación se ha dejado a disposición de los alumnos que cursen la asignatura Matemática Discreta, en la carpeta Recursos del poliformaT de la UPV, para su libre descarga.

7. Manual de usuario

La aplicación Grafos UPV está dividida en las siguientes secciones que posteriormente se detallan.

1. Barra de botones
2. Barra de menús
3. Panel de información
4. Panel de grafos
5. Barra de estado

1.- Barra de botones

Botones para manejo de información:

 ABRIR**:** Abre uno de los grafos guardados en XML.

 GUARDAR**:** Guarda el grafo actual en un archivo XML, lo que permitirá acceder al grafo en otro momento.

Botones para ejecutar los algoritmos:

 EJECUTAR**:** Ejecuta el algoritmo seleccionado.

 POR PASOS**:** Permite ejecutar el algoritmo paso a paso (sólo disponible para los algoritmos Edmonds (2) y Cartero Chino).

Botones para introducción de datos:

 K**:** Dibuja un grafo completo con el número de nodos que se indique, elimina cualquier grafo que hubiera anteriormente en el panel.

 MATRIZ ADYACENCIA**:** Abre un diálogo en el que se muestra la matriz de pesos del grafo. Si no hay ningún grafo dibujado, se pregunta el número de nodos que tendrá el grafo a dibujar y permite crear el grafo a partir de los pesos que introduzca el usuario en la matriz de pesos. Las modificaciones en los pesos de la matriz, serán visibles en el grafo dibujado cuando se vuelva a la pantalla inicial.

 BORRAR PANELES**:** Limpia por completo el panel de grafos, eliminando todos los nodos y aristas, así como, el panel de información.

 AYUDA**:** Abre en una ventana nueva el manual de usuario de la aplicación.

2.- Barra de menús

Menú Archivo:

ABRIR**:** (ver botón Abrir)

GUARDAR**:** (ver botón Guardar)

DEFINIR FONDO**:** Esta opción permite al usuario escoger una imagen de fondo para el panel de grafos. Esta utilidad permite dibujar el grafo sobre un fondo que represente el problema real a analizar mediante la teoría de grafos.

BORRAR FONDO**:** Elimina la imagen de fondo que se haya establecido con anterioridad.

SALIR**:** Permite salir de la aplicación, previa confirmación.

Menú Grafo:

GRADO DE LOS VÉRTICES**:** Para los grafos no dirigidos se muestra el grado de cada uno de los vértices del grafo. Para los grafos dirigidos, se muestra el grado de entrada y el grado de salida de cada uno de los vértices del grafo.

Nota: En el caso de tener seleccionado el algoritmo de Hierholzer, el grado de los vértices del s-grafo se calcula a partir de la matriz de multiplicidades.

GRAFO SUBYACENTE**:** Permite obtener el grafo subyacente del grafo dibujado en el panel de grafos.

¿ES CONEXO?**:** (disponible para grafos no dirigidos) Indica si un grafo es conexo o no lo es. (ver Ayuda - Grafos - Conectividad de grafos)

¿ES FUERTEMENTE CONEXO?**:** (disponible para grafos dirigidos) Indica si un grafo es fuertemente conexo o no lo es. (ver Ayuda - Grafos - Conectividad de grafos)

¿ES DÉBILMENTE CONEXO?**:** (disponible para grafos dirigidos) Indica si un grafo es débilmente conexo o no lo es. (ver Ayuda - Grafos - Conectividad de grafos)

GENERAR GRAFO COMPLETO**:** (ver botón K)

OBTENER MATRIZ DE ADYACENCIA**:** (ver botón Matriz Adyacencia)

 MATHEMATICA 4.0**:** Genera el código del grafo en Mathematica 4.0 (con el módulo grafos.m) de forma que un grafo dibujado en esta aplicación puede portarse a Mathematica simplemente copiando y pegando.

Menú Algoritmos:

EJECTUAR**:** (ver botón Ejecutar)

POR PASOS**:** (ver botón Por pasos...)

BFS, DFS, DIJKSTRA, ...**:** Selecciona el algoritmo que se desea ejecutar sobre el grafo introducido.

Menú Ejemplos:

BFS, DFS, DIJKSTRA, ...**:** Selecciona el algoritmo del que se desea obtener un grafo de ejemplo y escoger el deseado de los grafos disponibles.

Menú Ayuda:

MANUAL DE USUARIO**:** (ver botón Ayuda)

GRAFOS - Matriz del grafo**:** Muestra la información acerca de las diferentes matrices del grafo (matriz de adyacencia, de pesos, de multiplicidades, ...)

GRAFOS - Conectividad de grafos**:** Muestra la información relativa a la conectividad de grafos (conexo, no conexo, fuertemente conexo, ...)

ALGORITMOS**:** Se detalla la teoría de grafos necesaria para entender que hace cada uno de los algoritmos programados en este programa.

ACERCA DE**:** Información general de la aplicación SWGraphs.

3.- Panel de información

El panel de información es el que se encuentra a la izquierda de la interfaz gráfica de usuario. En él se mostrará el resultado tras ejecutar cualquiera de los algoritmos disponibles.

4.- Panel de grafos

PANEL DE DIBUJO**:** Es el panel en el cual se dibuja el grafo. Si desea dibujar un grafo no dirigido, sólo puede dibujar aristas, no arcos. Si desea dibujar un grafo dirigido, sólo puede dibujar arcos (aparece una flecha en lugar de una línea) y no podrá dibujar aristas. Es decir, en un grafo, o sólo hay aristas (líneas entre nodos) o sólo hay arcos (flechas entre nodos).

SELECCIÓN DEL TIPO DE GRAFO**:** Es el panel situado bajo el panel de dibujo, donde se especifica si se desea dibujar un grafo dirigido o no dirigido, así como si se desea dibujar un nodo o un arco/arista.

Nota: Si se desea cambiar de tipo de grafo, si se modifica esta opción, el programa le advertirá de que opciones dispone para realizar el cambio de tipo.

 Guardar Grafo Resultado**:** Permite guardar el grafo resultado de la ejecución de algunos algoritmos en un archivo XML, lo que permitirá acceder al grafo en otro momento.

Acciones del panel de dibujo de grafos

¿Cómo dibujo un nodo? Verifique que en la botonera inferior esta seleccionada la opción "dibujar NODO", una vez hecho esto para dibujar un nodo simplemente haga clic con el botón izquierdo en el panel de dibujo.

¿Cómo dibujo una arista o un arco? En este caso la función elegida en la botonera debe ser "dibujar ARISTA" o "dibujar ARCO". Para dibujar la arista/arco pulse el botón izquierdo del ratón sobre un nodo, mantenga el botón pulsado y arrastre el ratón hacia el otro nodo, suelte el ratón sobre este nodo e inmediatamente aparece una arista/arco uniendo ambos nodos.

¿Cómo borro un nodo? Elija la opción de nodos en la botonera, pulse con el botón derecho sobre el nodo a eliminar y en el menú emergente seleccione la opción "Borrar el nodo".

¿Cómo puedo renombrar un nodo? El proceso es similar al de borrado de un nodo, pero en este caso seleccione la opción "Cambiar el nombre" en el menú emergente.

¿Cómo muevo un nodo? Con la opción de nodos seleccionada haga clic con el botón izquierdo sobre un nodo y arrástrelo al lugar donde desee situarlo (existe un margen mínimo de distancia entre nodos). Las aristas/arcos incidentes en el nodo se desplazarán adecuadamente.

¿Cómo borro una arista o un arco? Con la opción aristas/arcos seleccionada pulse con el botón derecho sobre la arista/arco a eliminar y en el menú emergente seleccione la opción "Borrar arista". Nota: al intentar borrar un bucle, es posible que la aplicación borre una arista/arco no deseada, se recomienda borrar los bucles desde la matriz de adyacencia.

¿Cómo doy peso a una arista? Habrá observado que al dibujar una arista por defecto tiene peso 1, el proceso de cambio de peso es similar al de borrado de aristas, pero en este caso seleccione la opción "Cambiar peso de la arista" en el menú emergente. También se pueden cambiar los pesos en la matriz de adyacencia (botón de la barra superior). Mismo proceso para el cambio de peso de un arco. Para más información vaya al menú Ayuda, opción Grafos - Matriz del grafo.

¿Y si quiero un arista con peso cero? Habrá observado que al cambiar el peso de una arista a peso 0, si entra a ver la matriz del grafo, al salir y volver a dibujar el grafo la arista desaparece. Esto se debe a que el programa entiende que una arista con peso 0 es que no existe la arista.

IMPORTANTE: Si se desea trabajar con aristas con peso 0, hay que asignarle peso 0.1 ó 0,01 ó... inferior, es decir, un valor muy pequeño comparado con el rango de valores con el que se esté trabajando.

¿Por qué no aparece el nodo que quiero dibujar? Es posible que no aparezca el nodo si lo está dibujando muy próximo a uno ya existente, o que tenga seleccionada la opción "dibujar ARISTA" o "dibujar ARCO".

5.- Barra de estado

Corresponde a la barra inferior de la interfaz donde se irá mostrando el estado actual de la aplicación, última acción ejecutada, fecha, hora, etc.

8.Ejemplos de utilización

Como ya se ha dicho con anterioridad, la aplicación SWGraphs, ha sido desarrollada para la resolución de problemas de la asignatura Matemática Discreta de primer curso de Ingeniería Informática. Para la resolución de dichos problemas, lo más importante es su modelización antes de ejecutar el algoritmo que lo resuelva.

Describimos a continuación el significado concreto de la modelización en el ámbito de teoría de grafos. Para ello se va a proporcionar el esquema a seguir para transformar un problema enunciado en contexto real en un problema resoluble utilizando técnicas de la teoría de grafos, es decir, lo que llamamos modelización.

Los pasos a seguir son los siguientes:

1.- Definir un grafo G=(V,E) que represente la situación dada, es decir, definir los conjuntos V y E que se ajusten al contexto planteado teniendo presente qué es lo que se pretende conseguir.

Dado que pueden existir varias alternativas razonables para los citados conjuntos, habrá que seleccionar la que parezca más adecuada para lograr el objetivo.

2.- Convertir todos los datos del problema en conceptos o datos útiles desde el punto de vista de la teoría de grafos.

3.- Mientras se transforma el problema real en un problema de la teoría de grafos observar si la modelización sigue siendo adecuada o, a la vista de los datos, debería ser cambiada y reformular el problema.

4.- Una vez transformado el problema original en un problema perteneciente a la teoría de grafos, estudiar cuál es el método idóneo para su resolución. En ocasiones habrá que hacer cambios en el grafo considerado, es decir, definir un nuevo grafo G', para que se ajuste a las condiciones teóricas estudiadas.

5.- Aplicar el algoritmo que resuelve el problema, o bien a mano, o bien utilizando un software de resolución de algoritmos como SWGraphs.

6.- En caso de haber hecho modificaciones para poder aplicar un método concreto, hay que analizar en primer lugar que respuesta proporciona en el grafo originalmente definido G la solución obtenida en G'.

7.- A continuación debe darse una interpretación de la solución obtenida en el contexto original, esto es, la solución en términos de grafos debe convertirse en una solución al problema real planteado, haciendo el cambio inverso al definido anteriormente.

A continuación se detallan dos problemas planteados en la asignatura Matemática Discreta y resueltos, previa modelización, utilizando el software SWGraphs 2.0 desarrollado en el actual proyecto.

Ejemplo 1: PortAventura

Problema: Las familias Martínez, Pérez, Fernández y Jiménez van a ir a Port Aventura. Disponen de 4 coches: el de los Martínez y el de los Jiménez tiene 5 plazas cada uno, el de los Fernández tiene 4, mientras que el de los Pérez tiene 6.

A última hora surge un problema y los Martínez no pueden ir, pero deciden que si hay sitio, vayan sus trillizos. Además sería conveniente, que cada uno fuera en un coche distinto, para que no alboroten demasiado.

Los Pérez son 5 de familia, los Fernández son 4 y los Jiménez son 3.

Por otra parte a fin de intercambiar opiniones sobre unas obras que se están realizando en la urbanización donde viven, deciden que no vayan más de dos personas de la misma familia en cada coche, pero como es natural cada uno conduce el suyo. ¿Es posible hacer una distribución que les permita ir a todos a Port Aventura y que además respete las condiciones que ellos mismos se han impuesto? En caso afirmativo, ¿cuál será dicha distribución?

Resolución del problema utilizando SWGraphs

Paso 1 - Modelización del problema:

Definición de la red N = (G, s, t, c) donde:

G: Grafo débil conexo.

s: nodo fuente cuyo grado de salida debe ser mayor a 0.

t: nodo sumidero cuyo grado de entrada debe ser mayor a 0.

c: función capacidad: E(G) --> $N \cup \{0\}$

1.- Capacidad de las aristas que van desde s a los nodos de las familias indican el número de familiares que pertenecen a cada familia.

2.- Capacidad de las aristas que van desde los nodos de las familias a los nodos de los coches indican en número de familiares que pueden subir en cada uno de los coches.

3.- Capacidad de las aristas que van desde los nodos de los coches al nodo sumidero indican el número de plazas de cada uno de los coches.

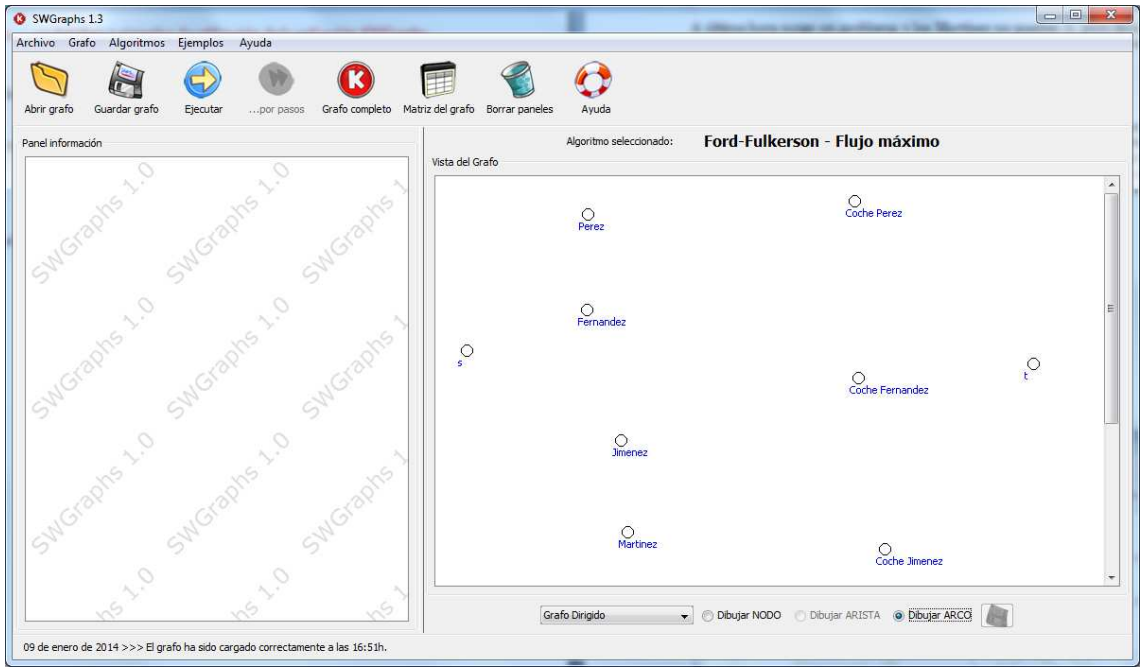
Flujo de la Red: f(N)

Para resolver el problema debemos obtener el máximo flujo de la red.

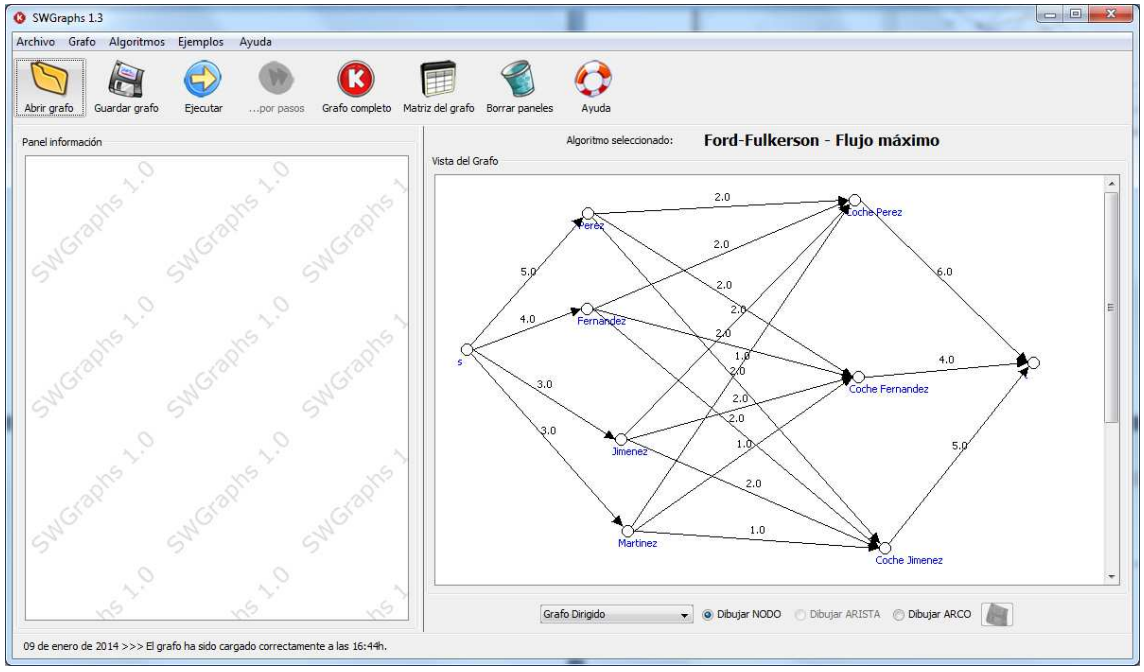
Para las condiciones de conductor y las recomendaciones de separar a los niños en diferentes coches, se modifican las capacidades de los arcos, restando una unidad por cada uno que suponemos subido al coche.

Paso 2 - Insertar el grafo en el panel de dibujo

2.1.- Insertar nodos

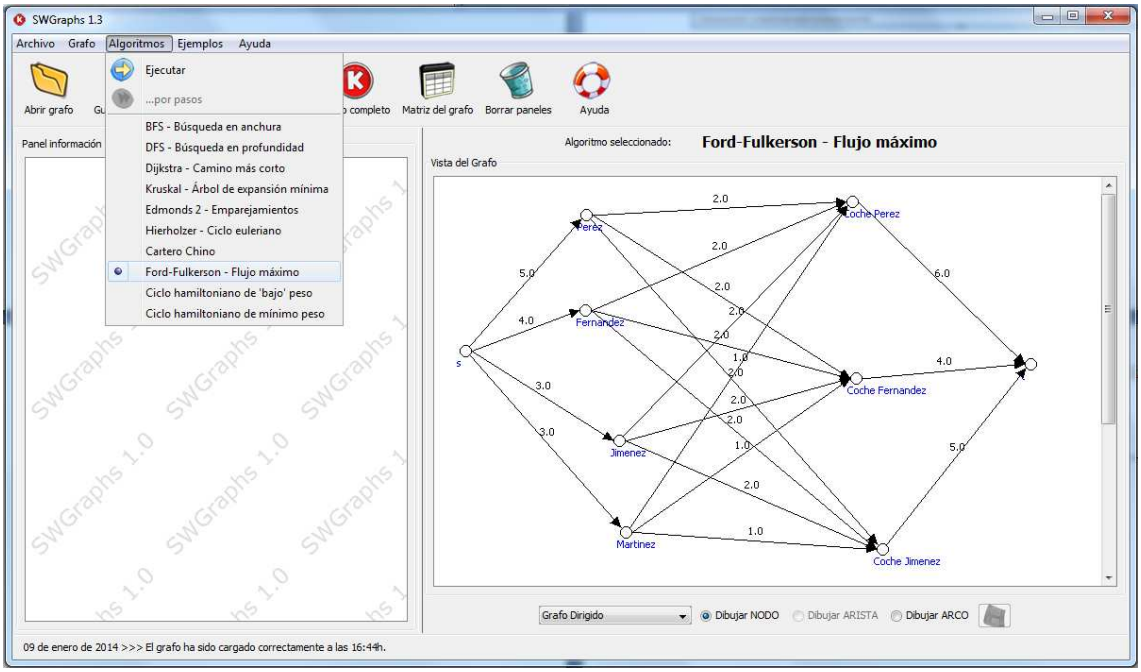


2.2.- Insertar aristas



Paso 3 - Escoger el algoritmo

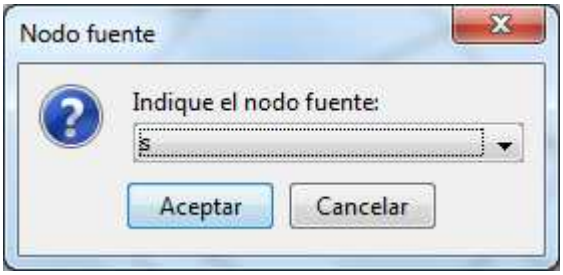
En este caso escogeremos el algoritmo de Ford-Fulkerson, que obtiene el flujo máximo de la red



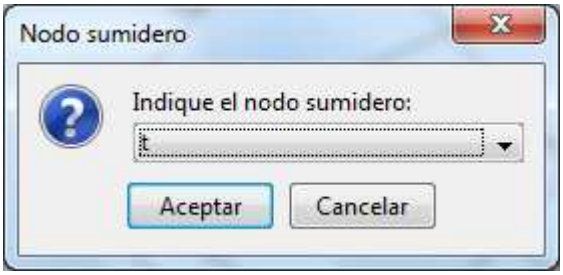
Paso 4 - Clic en el botón ejecutar

Al hacer clic en el botón de ejecutar, la aplicación nos preguntará cual es el nodo fuente y el nodo sumidero del grafo dibujado.

4.1.- Escoger cual es el nodo fuente.

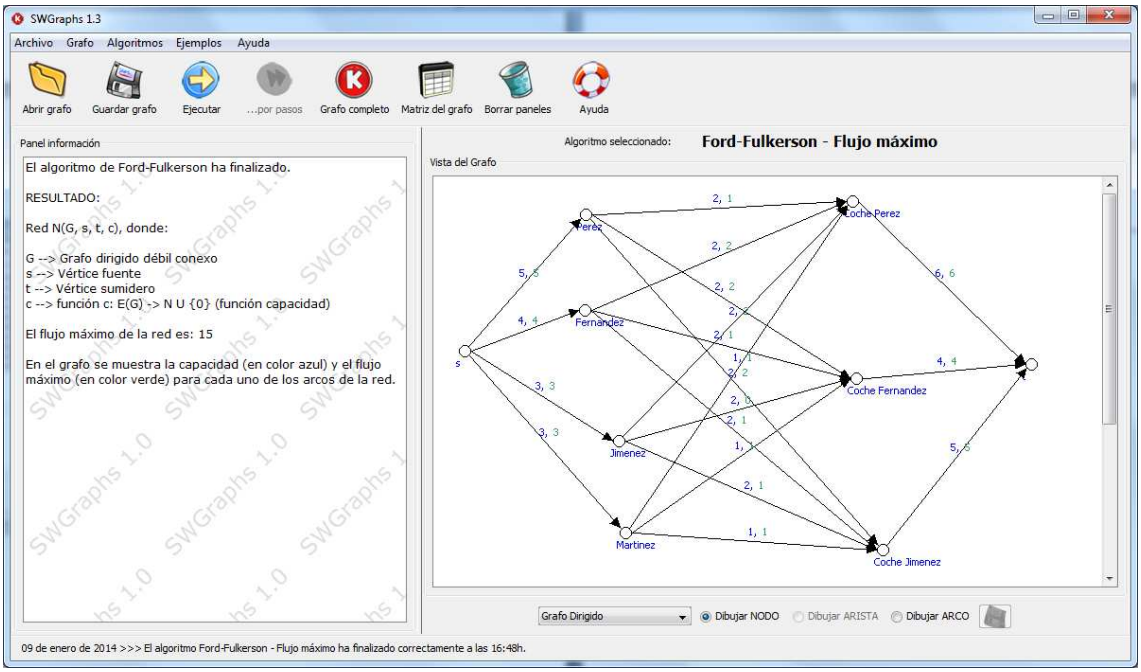


4.2.- Así como, escoger cual es el nodo sumidero.



Paso 5 - Interpretación de los resultados

Cuando le indiquemos a la aplicación cual es el nodo fuente y el nodo sumidero, el programa ejecutará el algoritmo de Ford-Fulkerson para obtener el flujo máximo de la red, así como el flujo de cada uno de los arcos de la red.



Una vez, SWGraphs ha ejecutado el algoritmo y ha mostrado los resultados por pantalla, se deben extraer las **conclusiones** para resolver el problema inicial.

Las conclusiones que se pueden extraer son:

Las familias Pérez, Fernández, Jiménez y Martínez sí pueden ir a Port Aventura y podrán hacerlo de la siguiente forma:

En el coche de los Pérez viajaran: El padre de la familia Pérez, 2 miembros de la familia Fernández, 2 de la familia Jiménez y 1 trillizo de la familia Martínez (total 6 personas).

En el coche de los Fernández viajaran: El padre de la familia Fernández y un miembro de cada una de las otras tres familias (total 4 personas).

En el coche de los Jiménez viajaran: El padre de la familia Jiménez, 2 miembros de la familia de los Pérez, 1 miembro de la familia Fernández y un trillizo de los Martínez (total 5 personas).

Ejemplo 2: Comerciante por Córdoba

Problema: Supongamos que un comerciante ha de visitar varias poblaciones conectadas entre sí. Hemos escogido 5 poblaciones de la provincia de Córdoba. El comerciante debe partir de Hinojosa del Duque, visitar las otras cuatro poblaciones y regresar a la población de origen. La pregunta es:

¿Cuál es el camino más corto para visitar todas las poblaciones y volver al lugar de origen?

Resolución del problema utilizando SWGraphs

Paso 1 - Modelización del problema:

Este problema es un problema clásico de teoría de grafos y se suele conocer como el problema del viajante de comercio (Traveling Salesman Problem).

El objetivo es que el viajante visite unas ciudades preestablecidas y vuelva al punto de partida recorriendo la mínima distancia.

Grafo G=(V, E), grafo no dirigido ponderado, donde:

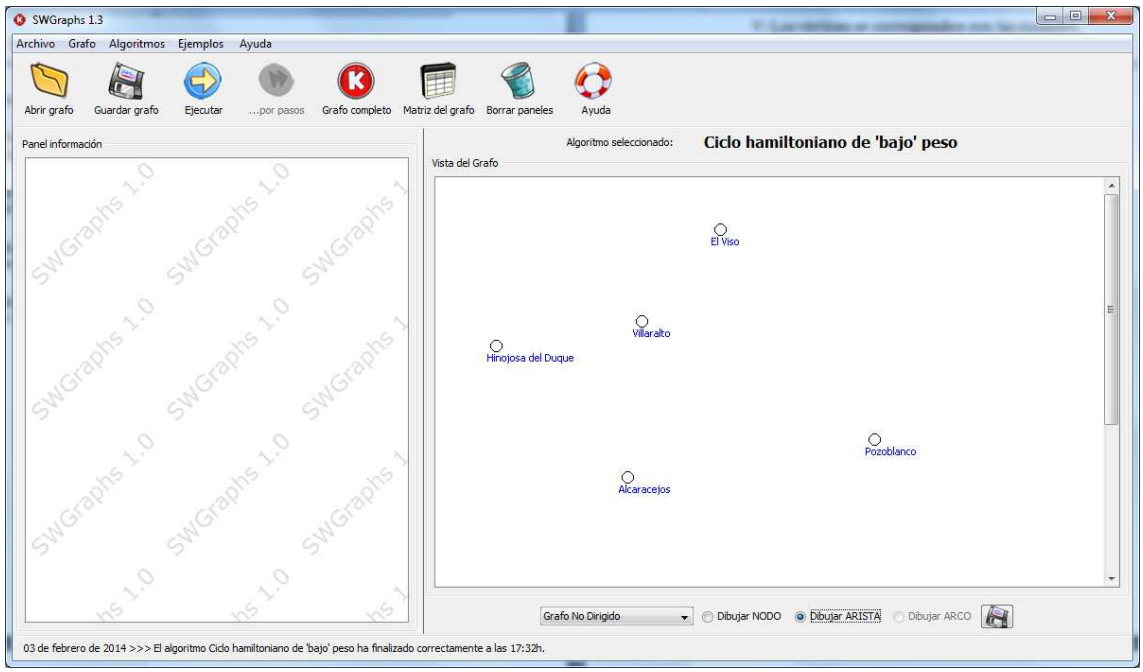
V: Los vértices se corresponden con las ciudades.

E: Las aristas con las carreteras que unen determinadas ciudades.

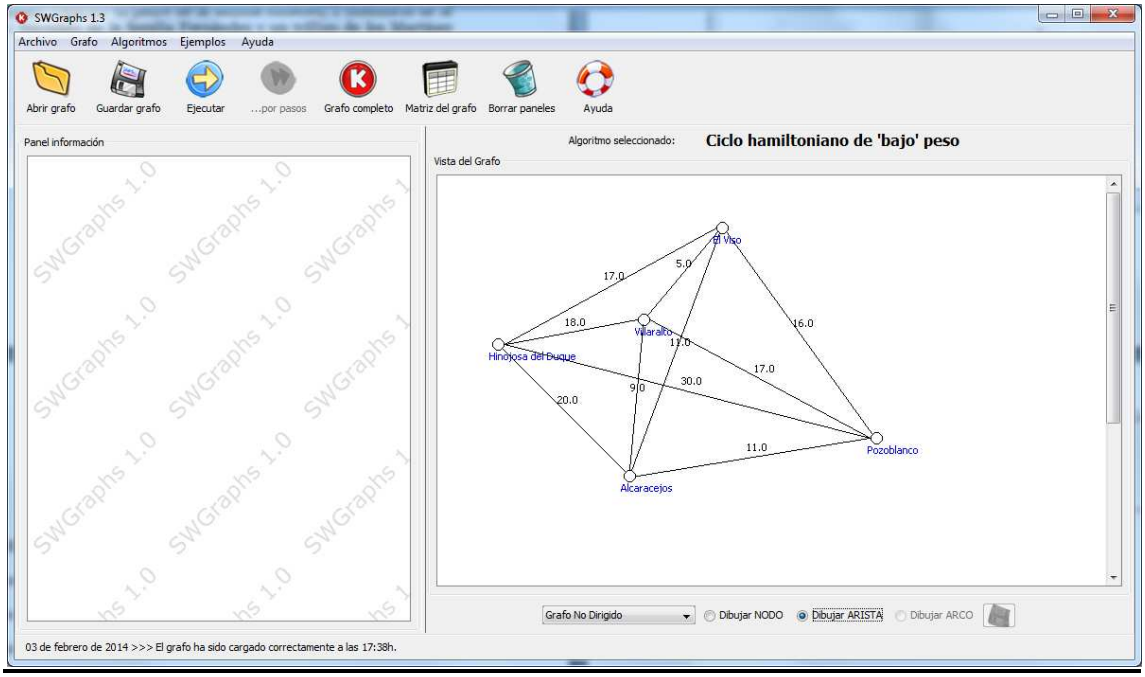
Resolveremos el problema con 3 algoritmos distintos, pero para ello primero insertaremos el grafo con el que vamos a trabajar.

Paso 2 - Insertar el grafo en el panel de dibujo

2.1.- Insertar nodos



2.2.- Insertar aristas

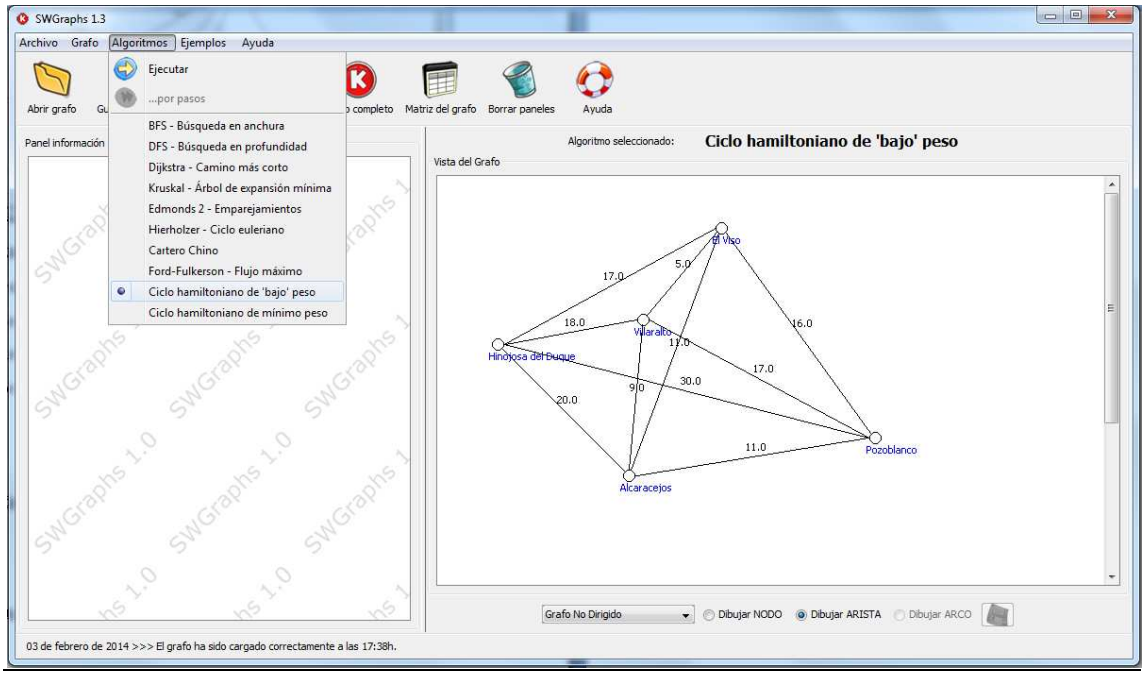


Paso 3 - Escoger el algoritmo

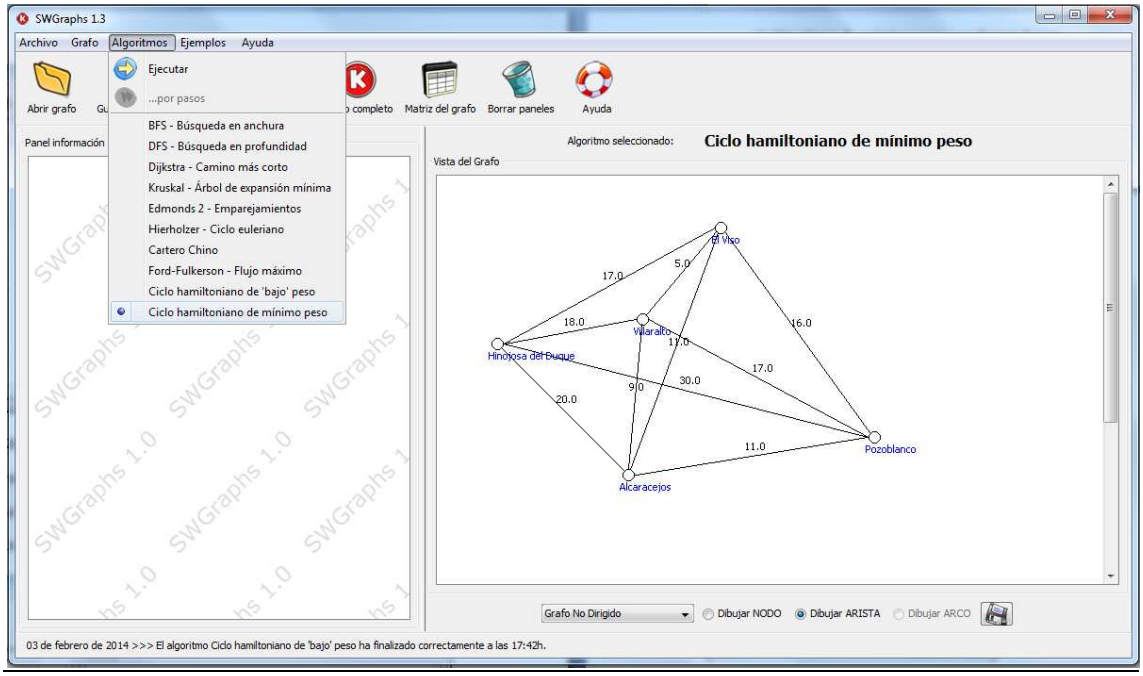
Vamos a resolver el problema con 3 algoritmos distintos. Para la ejecución de los primeros dos, se ha tenido que comprobar previamente que el grafo es completo y cumple la desigualdad triangular.

a) Algoritmo de bajo peso basado en árbol generador

b) Algoritmo de bajo peso voraz



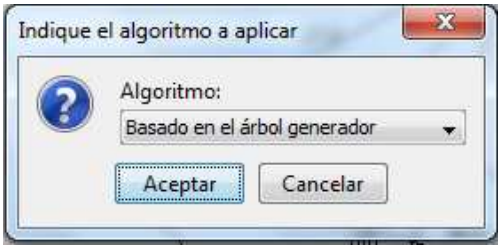
c) Algoritmo de mínimo peso por fuerza bruta

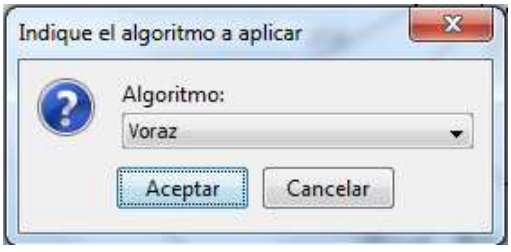


Paso 4 - Clic en el botón ejecutar

Al hacer clic en el botón de ejecutar, la aplicación nos preguntará cual es el algoritmo que deseamos utilizar para resolver el problema.

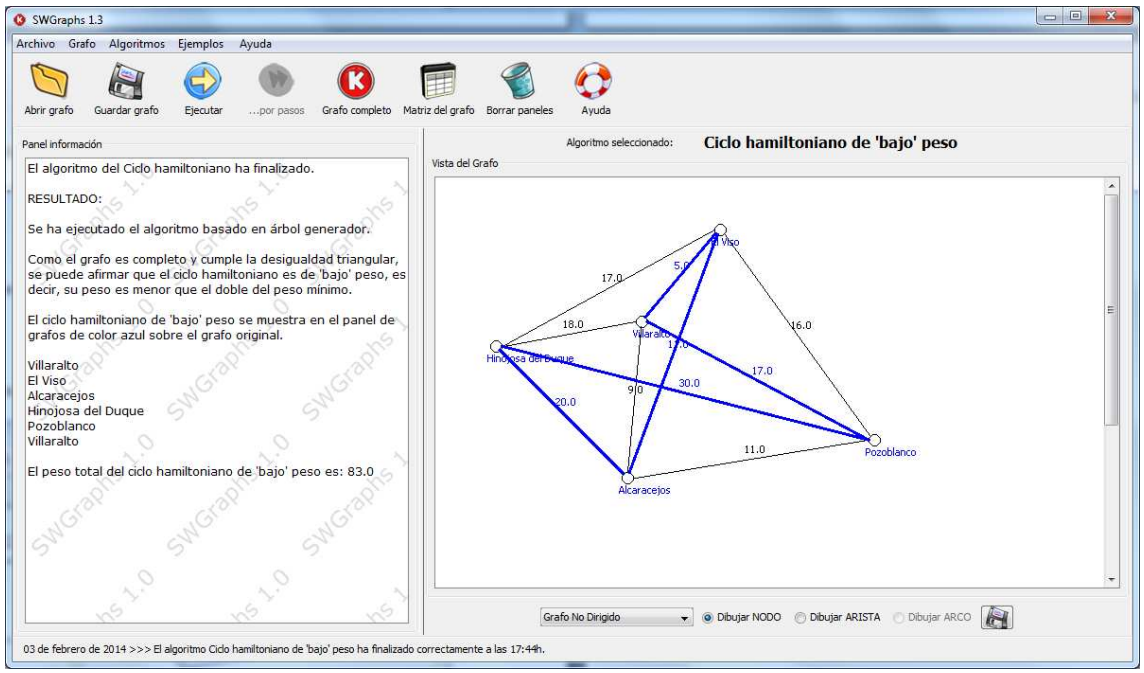
4.1.- Escoger que algoritmo deseamos ejecutar.





Paso 5 - Interpretación de los resultados

a) Algoritmo de bajo peso basado en árbol generador

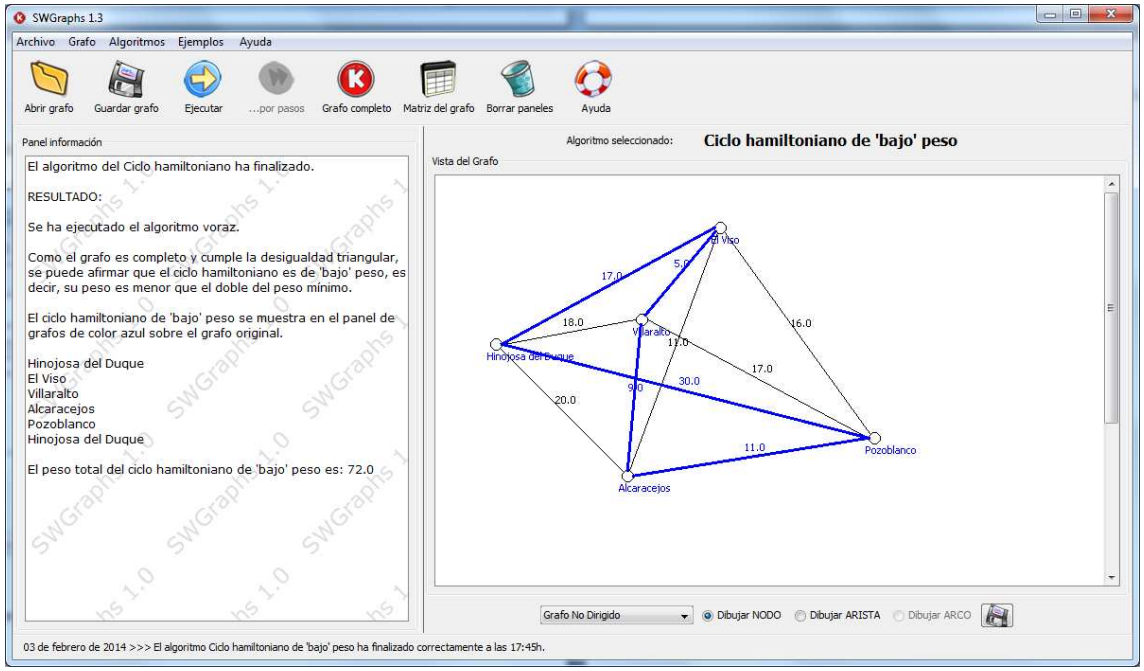


El ciclo hamiltoniano de 'bajo' peso se muestra en el panel de grafos de color azul sobre el grafo original.

Hinojosa del Duque -Pozoblanco -Villaralto -El Viso - Alcaracejos - Hinojosa del Duque

El peso total del ciclo hamiltoniano de 'bajo' peso es: 83.0

b) Algoritmo de bajo peso voraz

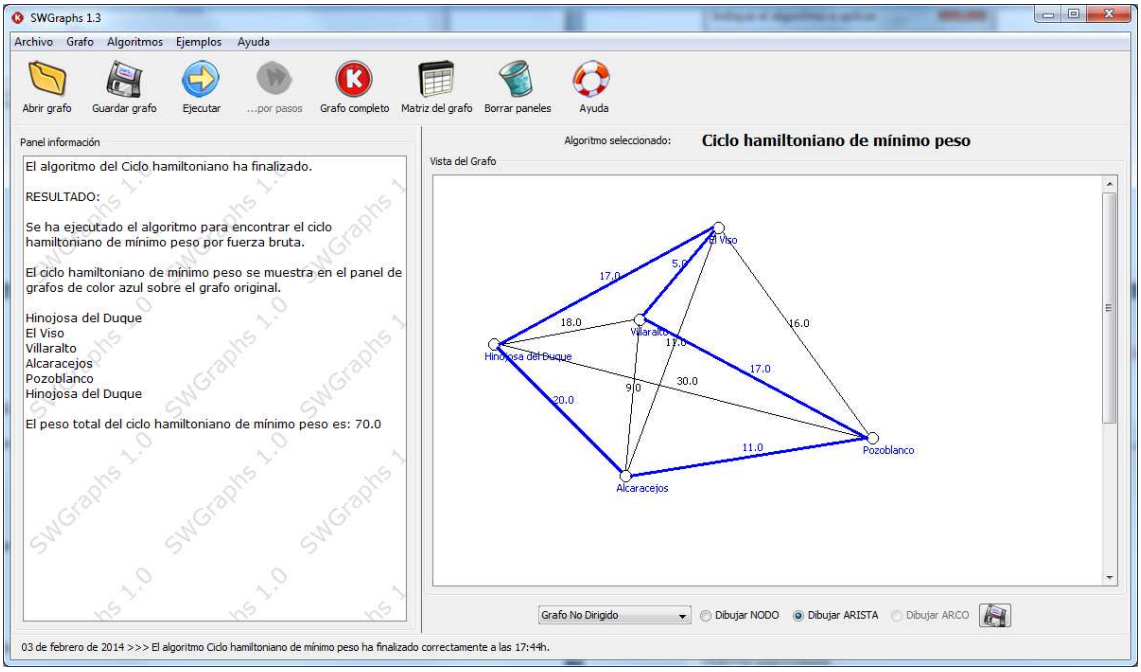


El ciclo hamiltoniano de 'bajo' peso se muestra en el panel de grafos de color azul sobre el grafo original.

Hinojosa del Duque - El Viso - Villaralto -Alcaracejos -Pozoblanco- Hinojosa del Duque

El peso total del ciclo hamiltoniano de 'bajo' peso es: 72.0

c) Algoritmo de mínimo peso por fuerza bruta



El ciclo hamiltoniano de mínimo peso se muestra en el panel de grafos de color azul sobre el grafo original.

Hinojosa del Duque -El Viso -Villaralto - Pozoblanco - Alcaracejos -Hinojosa del Duque

El peso total del ciclo hamiltoniano de mínimo peso es: 70.0

9.Conclusión

La Teoría de Grafos constituye un parte importante de la Matemática Discreta, asignatura de primero del grado de la ETSINF. Como es sabido se trata de una materia con gran aplicabilidad tanto a problemas de la vida real como pertenecientes a cualquier campo científico. Por ello, en las clases de laboratorio se presta especial atención a resolver problemas previa modelización.

Es necesario por tanto disponer de un software que contenga los algoritmos que se estudian durante el curso, de sencillo aprendizaje y manejo. Por otra parte, es también conveniente que tenga un interfaz agradable, se pueda hacer correr en cualquier ordenador y del que los alumnos puedan disponer libremente en casa. Necesitamos por tanto un programa a medida.

Un software personalizado o a la medida es muy superior a los genéricos porque los programas genéricos no necesariamente hacen frente a todos los aspectos que se quieren tratar en la asignatura.

Las ventajas y beneficios se pueden obtener con un software a medida son innumerables ya que es una solución personalizada que se adapta a las necesidades concretas de la asignatura, por poner un ejemplo se podrían mencionar los siguientes:

- Hacer que el trabajo para el que fue diseñado sea mucho más fácil de realizar que anteriormente con otros programas.

- Disminuir la cantidad de tiempo dedicado a la formación del programa a utilizar, ya que SWGraphs es muy sencillo de aprender a utilizar.

- Alumnos mucho más motivados y satisfechos ya que el programa es más sencillo de utilizar.

- Aumento de la cantidad de ejercicios trabajados en la asignatura.

Por todo ello se decidió crear un software a medida para las asignaturas cuyo contenido forma parte total o parcialmente de la teoría de grafos impartidas por la unidad docente de la ETSInf del departamento de Matemática Aplicada de la UPV.

El programa ha cubierto con creces las expectativas, ya que, con un agradable interfaz, nos permite centrarnos en lo que es objeto de estudio, la Teoría de Grafos y su aplicación a la resolución de problemas reales.

Todos los objetivos propuestos al principio del proyecto se han cumplido y así lo demuestra la buena acogida del programa SWGraphs por parte tanto de profesores de la asignatura como de alumnos.

Por tanto, podemos considerar que la aplicación ha sido un éxito como demuestra el uso del programa en 16 de los 20 grupos de prácticas que tiene la asignatura. Además, es palpable la confianza que los profesores que han depositado en SWGraphs, ya que han realizado los exámenes de prácticas con el software desarrollado en este proyecto.

La realización de este proyecto ha supuesto el desafío de analizar, sintetizar y resolver problemas desarrollando técnicas de investigación como observación, planteamiento del proceso, emisión de hipótesis, búsqueda de soluciones, elaboración de experimentos discusión de resultados, etc.

Ya se está hablando de ampliar en un futuro las funcionalidades de la aplicación para tener una mejora continua del software. Entre las posibles ampliaciones, se está pensando en la implementación de nuevos algoritmos que puedan estudiarse en las diferentes asignaturas donde se utiliza SWGraphs, añadir nuevos grafos de ejemplo, o un generador de grafos tipo, que el usuario podría escoger para ver el funcionamiento de determinados algoritmos.

Otra posible ampliación del programa será traducirlo a cualquier idioma, tarea que no presentaría demasiadas dificultades ya que se han externalizado en un fichero todos los textos que aparecen en la aplicación.

Bibliografía

- JORDAN LLUCH, Cristina, TORREGROSA SANCHEZ, Juan R. *Introducción a la Teoría de Grafos y sus algoritmos*. Valencia: Servicio de publicaciones de la Universidad Politécnica de Valencia. 1996. ISBN: 84-7721-438-7.

- OCW (OpenCourseWare) Estructuras Matemáticas para la Informática 2 http://www.upv.es/pls/oalu/sic_asi.Sak_Recursos_ocw?P_OCW=E&P_ASI=6024&P_CACA=2010&P_IDIOMA=c&P_VISTA=MSE

- APLICACIONES DE LA TEORÍA DE GRAFOS A LA VIDA REAL. Cristina Jordán Lluch [enero de 2013] Disponible en web: https://polimedia.upv.es/catalogo/curso.asp?curso=9f11f633-55a9-ce4b-91b9-03b0e3850abb

- MATHLAND. *Konigsberg Bridges*. [en línea] [ref. marzo de 2005] Disponible en web: http://math.youngzones.org/Konigsberg.html

- INGENIERIA DEL SOFTWARE (6ª ED.) Roger S. Pressman, McGraw-Hill, 2005. ISBN 9789701054734

- SUN MICROSYSTEMS. *Java™ 2 Platform Standard Edition 5.0. API Specification* [en línea] [ref. septiembre 2004 a septiembre 2005] Disponible en web: http://java.sun.com/j2se/1.5.0/docs/api/

- SUN MICROSYSTEMS. *The Java Tutorial*. [descargable: http://java.sun.com/docs/index.html]

Anexo I: Código XML de un grafo

```
<?xml version="1.0" standalone="yes" ?>

<grafo>

<tipoGrafo> "0" </tipoGrafo>

<nodos>

<nodo>
        <nombre> "v0" </nombre>
        <posicion> (65,85) </posicion>
</nodo>
<nodo>
        <nombre> "v1" </nombre>
        <posicion> (178,175) </posicion>
</nodo>
<nodo>
        <nombre> "v2" </nombre>
        <posicion> (262,85) </posicion>
</nodo>
<nodo>
        <nombre> "v3" </nombre>
        <posicion> (342,175) </posicion>
</nodo>
<nodo>
        <nombre> "v4" </nombre>
        <posicion> (445,265) </posicion>
</nodo>
<nodo>
        <nombre> "v5" </nombre>
        <posicion> (445,85) </posicion>
</nodo>
<nodo>
        <nombre> "v6" </nombre>
        <posicion> (65,265) </posicion>
</nodo>
<nodo>
        <nombre> "v7" </nombre>
        <posicion> (522,175) </posicion>
</nodo>

</nodos>

<aristas>

<arista>
        <nodo1> "0" </nodo1>
        <nodo2> "1" </nodo2>
        <peso> "3.0" </peso>
        <tipo> "0" </tipo>
</arista>
```

```
<arista>
        <nodo1> "0" </nodo1>
        <nodo2> "2" </nodo2>
        <peso> "1.0" </peso>
        <tipo> "0" </tipo>
</arista>
<arista>
        <nodo1> "1" </nodo1>
        <nodo2> "3" </nodo2>
        <peso> "1.0" </peso>
        <tipo> "0" </tipo>
</arista>
<arista>
        <nodo1> "1" </nodo1>
        <nodo2> "6" </nodo2>
        <peso> "5.0" </peso>
        <tipo> "0" </tipo>
</arista>
<arista>
        <nodo1> "2" </nodo1>
        <nodo2> "5" </nodo2>
        <peso> "5.0" </peso>
        <tipo> "0" </tipo>
</arista>
<arista>
        <nodo1> "3" </nodo1>
        <nodo2> "4" </nodo2>
        <peso> "4.0" </peso>
        <tipo> "0" </tipo>
</arista>
<arista>
        <nodo1> "4" </nodo1>
        <nodo2> "6" </nodo2>
        <peso> "2.0" </peso>
        <tipo> "0" </tipo>
</arista>
<arista>
        <nodo1> "4" </nodo1>
        <nodo2> "7" </nodo2>
        <peso> "1.0" </peso>
        <tipo> "0" </tipo>
</arista>
<arista>
        <nodo1> "5" </nodo1>
        <nodo2> "7" </nodo2>
        <peso> "3.0" </peso>
        <tipo> "0" </tipo>
</arista>

</aristas>

 </grafo>
```

Anexo II. Contenido del entregable

Memoria

Código fuente

Ejecutable

Grafos de ejemplo