Escuela Politécnica Superior
Ingeniería Técnica en Informática de Gestión
Departamento de Ciencia, Ingeniería de Materiales e Ingeniería Química

PROYECTO FIN DE CARRERA

GrafoMin

Una herramienta para el aprendizaje y la construcción del camino más corto entre dos nodos de un grafo

Autor: Gustavo Adolfo Iglesias Bello
Tutor: Eduardo Jesús Sánchez Villaseñor
Codirector: Jesús Salas Martínez

Leganés, Julio de 2012

Título: GrafoMin: una herramienta para el aprendizaje y la construcción del camino más corto entre dos nodos de un grafo.
Autor: Gustavo Adolfo Iglesias Bello.
Tutor: Eduardo Jesús Sánchez Villaseñor.
Codirector: Jesús Salas Martínez.

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Mi reconocimiento a doña Elena Castro Galán y a don Manuel Velasco de Diego del departamento de Informática por su apoyo. Sin su inestimable ayuda emocional, en momentos críticos, y docente, no podría haber teminado mis estudios.

Mi gratitud a don Eduardo Jesús Sánchez Villaseñor, mi paciente tutor: diligente con cualquier aclaración solicitada, bien sea de su asignatura, bien de cualquier otra índole.

Mis complacencias a don Gregorio Peces-Barba Martínez, quien colocó la primera del edificio que profesionalmente soy.

A mi padre, Manuel Eduardo Iglesias Martínez, siempre presente, y a mi madre Marina Bello Osorio.

A mi hermana María de los Ángeles, a mi cuñado José Miguel, a mis sobrinos Iago y Cristina Sara, por acogerme en su casa los primeros años de carrera, y apoyarme en los momentos más difíciles.

Y, por supesto, a María Beatriz Ruano Pérez quien con su empeño me impulsó durante el presente proyecto.

Resumen

GrafoMin es el nombre de una herramienta gráfica para el aprendizaje y la construcción del camino más corto entre dos nodos (o vértices) de un grafo. Esta herramienta puede ser utilizada tanto por los estudiantes, en el proceso de aprendizaje usando el algoritmo de Dijsktra y como prueba de auto-evaluación, así como por los profesores que deseen impartir dicha materia.

El usuario puede crear el grafo a su gusto: añadiendo tantos vértices y aristas como desee, indicando los vértices inicial y final y dándole valores arbitrarios a los pesos de las aristas, o bien puede tratar de resolver el problema, en cuyo caso su solución será evaluada; o bien simplemente, puede solicitar a GrafoMin la resolución del mismo.

La herramienta es independiente de la plataforma, puesto que se accede a ella por medio de un navegador, pudiendo ser Mozilla Firefox o Google Chrome en cualquiera de sus versiones.

La herramienta puede estar alojada en un servidor sin requerimientos dinámicos o puede ser residente en el propio equipo, ya que todos los procesos son ejecutados en la computadora cliente.

Para el desarrollo de GrafoMin ha sido necesario utilizar la versión 5 de HTML ya que ésta es la única versión que dispone de la etiqueta <CANVAS>, necesaria para dibujar las aristas. Esta versión de HTML se encuentra actualmente en fase de experimentación por parte del consorcio W3C y por ello que ha sido necesaria una labor de investigación en este nuevo lenguaje.

GrafoMin es innovadora en tanto en cuanto es la primera aplicación existente programada en código HTML, CSS y JavaScript. Hasta el momento las únicas herramientas estaban basadas en "*applets*" de Java

Se han observado los estándares de "*usabilidad*usabilidadusabilidad" para un rápido aprendizaje, orientándose en todo momento al usuario a un manejo gráfico e intuitivo por medio del ratón y sin tener que hacer uso del teclado, excepto para la introducción de los valores de los pesos de las aristas. También se dispone de vídeo tutoriales *ad hoc* en línea.

La herramienta permite la consulta de documentación en la misma ventana, a través de un marco con acceso a otras direcciones de Internet para complementar la formación.

Palabras clave: GrafoMin, Dijkstra, Mozilla, Chrome, CANVAS, camino, camino más corto, grafo, vértice, nodo, arista, peso, usabilidad, grafo no dirigido, grafo dirigido, grafo ponderado, JavaScript, HTML5.

Abstract

GrafoMin is a graphic tool that computes, by using Dijsktra algorithm, the shortest path between two vertices of a given graph. This tool can be used by teachers and also by students, for both learning and self-evaluation proceses.

The user can create the graph as he wishes: adding vertices, edges, giving arbitrary values to the weights associated to the edges, and pointing out the initial and final vertices. The user may try to solve the problem by him(her)self and check the answer or may ask GrafoMin to solve the problem.

The tool is independent from the platform, because you can access to it throught the browser, beeing this one Mozilla Firefox or Google Chrome in any of its versions.

The tool can be hosted in a server without any dinamic requirements or it can be resident of the computer itself, for the reason that all the processes are implemented in the customers computer.

The tool can be hosted in a server or in any computer as all the processes are implemented on the client computer. The development of GrafoMin has required the use of HTML5 because this version supports the <CANVAS> tag, which is necessary to draw the edges. The consortium W3C consideres this version of HTML to be in beta state. This is why we have to learn this new language. The tool is platform-independent: you can access the application via a web browser like Mozilla Firefox or Google Chrome in any of their versions.

GrafoMin is the first existing application programmed in HTML, CSS, and JavaScript. As far as we know, all the previous tools were based on Java "applets". We have followed the usability standards for a quick learning, focusing in every moment towards the use the mouse (the keyboard is only needed to give the values of the edge weights). Additional documentation is also available on-line, including tutorial videos.

Keywords: GrafoMin, Dijkstra, Mozilla, Chrome, CANVAS, path, shortest path, vertex, vertices, node, edge, weight, usability, undirected graph, directed graph, weighted graph, JavaScript, HTML5.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

A lo largo de esta memoria se pueden descubrir todos los detalles y las bases teóricas de la implementación del Proyecto Fin de Carrera *GrafoMin: una herramienta para el aprendizaje y la construcción del camino más corto entre dos nodos de un grafo*.

Uno de los problemas clásicos de la Teoría de Grafos es la determinación del camino más corto entre dos de sus vértices. Dicho problema tiene aplicación directa en multitud de situaciones cotidianas, como encontrar el camino más corto entre dos ciudades, o entre dos direcciones de una ciudad.

En 1959 Edsger Wybe Dijkstra (Rotterdam 1930 – Nuenen 2002) describió el algoritmo para la determinación del camino más corto entre un vértice dado y el resto de vértices de un grafo ponderado de pesos positivos. Utilizó los principios de la búsqueda en anchura sobre un grafo para construir un árbol, cuyo nodo raíz es el nodo inicial (también llamado nodo origen) y cuyas aristas definen el camino más corto, si existe, entre dicho nodo raíz, y el resto de los nodos del grafo. El órden de complejidad temporal obtenido para la resolución del problema es de $O(n^2)$, como se puede verificar en el siguiente pseudo código[(1)]:

```
función Dijkstra (Grafo G, nodo_salida s)
  //Usaremos un vector para guardar las distancias del nodo salida al resto
```

[(1)] La complejidad temporal programática de un problema es el número de pasos que toma resolver una instancia de un algoritmo, a partir del tamaño de la entrada. Cada una de las entradas de dicha instancia se nota como "n".

```
entero distancia[n] //Inicializamos el vector con distancias iniciales
booleano visto[n] //vector de boleanos para controlar los vertices de los que ya tenemos la distancia mínima
para cada w ∈ V[G] hacer
  Si (no existe arista entre s y w) entonces
    distancia[w] = Infinito //puedes marcar la casilla con un -1 por ejemplo
  Si_no
    distancia[w] = peso (s, w)
  fin si
fin para
distancia[s] = 0
visto[s] = cierto
//n es el número de vertices que tiene el Grafo
mientras que (no_esten_vistos_todos) hacer
  vertice = coger_el_minimo_del_vector distancia y que no este visto;
  visto[vertice] = cierto;
  para cada w ∈ sucesores (G, vertice) hacer
    si distancia[w]>distancia[vertice]+peso (vertice, w) entonces
      distancia[w] = distancia[vertice]+peso (vertice, w)
    fin si
  fin para
fin mientras
fin función
```

El algoritmo de Dijkstra se ha convertido en materia fundamental de estudio en la gran mayoría de las ramas de la Ingeniería. El presente proyecto toma como base teórica dicho algoritmo, implementado en una herramienta de software orientada a la formación y aprendizaje de cómo resolver el problema del *camino más corto entre dos nodos de un grafo*. A continuación detallamos el esquema de la presente memoria:

- **Principales objetivos** que se persiguen con el proyecto: qué se quiere conseguir con el proyecto (típicamente un objetivo importante y varios subobjetivos, etc.).
- **Fases de desarrollo**: que ha habido que hacer para llevar a cabo todo el proyecto.
- **Medios con los que se ha contado para hacer el proyecto**: hardware, software, laboratorios, etc.
- **Esquema de la memoria**: breve resumen con un párrafo de cada capítulo para facilitar la tarea al lector.

1.2 Objetivos

El objetivo fundamental de este proyecto es la implementación, con fines didácticos, del algoritmo de Dijkstra. Para ello se ha enfocado el problema desde dos vertientes:

- **Aprendizaje por parte del alumno**. El alumno desea conocer y contrastar sus conocimientos sobre el algoritmo, por tanto la herramienta le ofrece:
  - El planteamiento del problema a su gusto. Podrá crear un grafo a su voluntad, planteando cualquier supuesto de ésta índole.
  - Contrastará sus resultados, tanto analíticos como de resolución.
  - Repetición de ejercicios tantas veces como el alumno crea necesario.

- **Enseñanza por parte del formador**. El profesor cuenta con la misma herramienta que el alumno. El profesor dispondrá de una herramienta adaptada a los métodos de enseñanza actuales, valiéndose de las últimas tecnologías existentes.
  - Creación de cualquier supuesto válido.
  - Resolución gráfica para un planteamiento a un grupo de alumnos.
  - Una herramienta docente adaptada a los actuales elementos de formación con los que cuentan las aulas de la Universidad Carlos III: computadora y proyector.

El enfoque adoptado nos ha mostrado los objetivos anteriores, los primeros y principales, y como consecuencia natural, obtenemos necesariamente los siguientes:

- Elaboración de la herramienta en un entorno multiplataforma que permita el acceso tanto al alumno, como al profesor.
- El formador debe tener disponible la herramienta en diferentes aulas.
- El alumno también debe poder disponer de la herramienta desde su lugar habitual de estudio: su domicilio, la biblioteca u otros lugares de estudio.
- La herramienta debe ser de fácil aprendizaje y uso, para animar a los estudiantes a su utilización.

1.3 Terminología específica

Es preciso hacer una breve descirpción de los términos y conceptos matemáticos utilizados en la resolución del problema.

DEFINICIONES

- Vértice: (también llamdo nodo) un vértice es simplemente un punto de un cierto espacio. Un vértice puede ser o no incidente con una arista.
- Arista: (también llamado arco) cada una de las uniones o enlaces entre vértices.
- Grado d(v) de un vértice *v*: el número de aristas que se inician o terminan en él, y se denota por d(*v*).
- Par ordenado: una pareja de elementos denotada por (*a , b)* donde, *a* es el primer elemento y *b* es el segundo.

- Par no ordenado: conjunto de la forma *{a , b}* de manera que *(a , b) identifica (b , a)*.
- Relación binaria *R* entre elementos de dos conjuntos *A* y *B*:

$$R = \{(a, b): a \in A \ \wedge b \in B \ \wedge R(a, b) = cierto\}$$

- Grafo: informalmente es conjunto de vértices unidos por aristas, que permiten representar relaciones binarias entre elementos de un conjuto. Grafo se llama al par ordenado *G = (V , A)*, donde *V* es un conjunto de vértices y *A* el conjunto de aristas que relacionan dichos vértices.
- Conjunto potencia de *V* o conjunto de partes de *V*: el conjunto formado por todos los posibles subconjuntos de *V*. Se denota mediante *P(V)*.
- Cardinal: número de elementos de un conjunto. Si el conjunto es llamado *X* se denota por $|X|$.
- Grafo no dirigido: *G = (V , A)* donde,
  - $V \neq \emptyset$
  - $A \subseteq \{X \in P(V): |X| = 2\}$ es un conjunto de pares no ordenados de elementos de *V*.
  - Por definición los grafos no dirigidos no contienen bucles -aristas del tipo *(a,a)*-.
- Grafo dirigido: *G = (V , A)* donde,
  - $V \neq \emptyset$
- $A \subseteq \{(a, b) \in \ V \times V: a \neq b\}$ es el conjunto de pares ordenados de elementos de *V*. Dada la arista *(a,b): a* es el vértice inicial y *b* el vértice final.
- Grafo simple: grafos en los que solo puede haber una arista entre dos vértices. Trivialmente no posee bucles.
- Grafo ponderado: es aquel que se le ha asignado una por definición función de ponderación. (ver propiedad de ponderación).
- Grafo etiquetado: se dice del grafo cuyos vértices y aristas se identifican unívocamente mediante un identificador.
- Camino mínimo entre dos vértices de un grafo ponderado (ver propiedades): camino entre dos vértices (o nodos) de tal manera que la suma de los pesos de las aristas que lo constituyen es mínima.
- Caminos mínimos desde un vértice inicial V de un grafo ponderado: partiendo de la generalización anteriror, podemos restringir el problema a la búsqueda de todos los caminos más cortos partiendo de un mismo vértice origen, hasta todos los restantes vértices del grafo. El algoritmo de Dijkstra resuelve de dicho problema de manera óptima. El algoritmo no admite pesos negativos.

PROPIEDADES

- Incidencia: una arista es incidente a un vértice si ésta lo une a otro.
- Adyacencia:
  - dos aristas son adyacentes, si tienen un vértice común;
  - dos vértices son adyacentes si tiene una arista común.
- Ponderación: corresponde a una función que a cada arista le asocia un valor (costo, peso, longitud, etc.). Se utiliza para aumentar la expresividad del modelo.

Capítulo 2

Estado del arte

2.1 Introducción

Antes de proceder al análisis de nuestra herramienta, es preciso hacer un estudio del diferente software existente. Cabe notar que no hay un campo abundante de programas formativos sobre el algoritmo de Dijkstra, sino más bien un grupo marginal, heterogéneo en lo que se refiere a los diferentes tipos de leguajes en que han sido implementados. En algunos casos nos encontraremos con aplicaciones orientadas a la formación, al mundo empresarial, a la comunidad de programadores, etc.

Todos los programas analizados están disponibles en castellano: al estar orientado a la formación de los alumnos de primeros cursos de Grado, hemos decidido descartar las herramientas que no estuvieran disponibles en castellano.

2.2 PathFinder – eMath Teacher

Esta aplicación (residente y programada Java) ha sido el Proyecto Fin de Carrera en el departamento de Matemática Aplicada de la Facultad de Informática de la Universidad Politécnica de Madrid de Miguel A. López Martínez y dirigida por la profesora M. Gloria Sánchez Torrubia.

El objetivo de la aplicación es, mediante un diseño sencillo, dar a conocer el algoritmo de Dijkstra a sus usuarios. Dispone para ello de dos tipos de resolución de pantalla (alta y baja) y está disponible tanto en español como en inglés. También ofrece una guía de referencia rápida en línea, si está disponible la conexión a Internet.

Es fácil crear el grafo no dirigido (no admite dirigidos): con simples pulsaciones de ratón en el modo gráfico se crean los vértices (toma por defecto el primer nodo como el vértice inicial), se crean las aristas (y valorando cada una de ellas con un peso). Durante el proceso de creación, llamado por la aplicación "modo dibujo", la herramienta va mostrando mensajes para guiar al usuario en el planteamiento del problema. Una vez "fijado el grafo" (así es como llama al modo de resolución del problema), ofrece tres opciones de ejecución.

I. Corrección paso a paso: el usuario introduce los valores correctos en las celdas de datos de la tabla. La herramienta permite la modificación de una celda secuencialmente al desarrollo del algoritmo de Dijkstra y evalúa en cada secuencia de introducción de los datos los valores insertados, como correctos o incorrectos. El proceso se repite hasta fijar (alcanzar) el nodo final.
II. Corrección por cada iteración: similar a la opción anterior pero introduciendo los datos en la tabla no uno a uno, sino los de cada iteración, verificando si la iteración ha sido completada correctamente.
III. Ejecución directa: la herramienta ejecuta el proceso en su totalidad, mostrando el resultado final en forma de tabla de una única fila y, coloreando el camino en el grafo.

Permite guardar el problema o bien cargar uno guardado anteriormente.

Está disponible para entornos Microsoft Windows que tengan instalado Java Web Start Laucher y Java Runtime Environment (JRE) 1.5 ambos de Oracle.

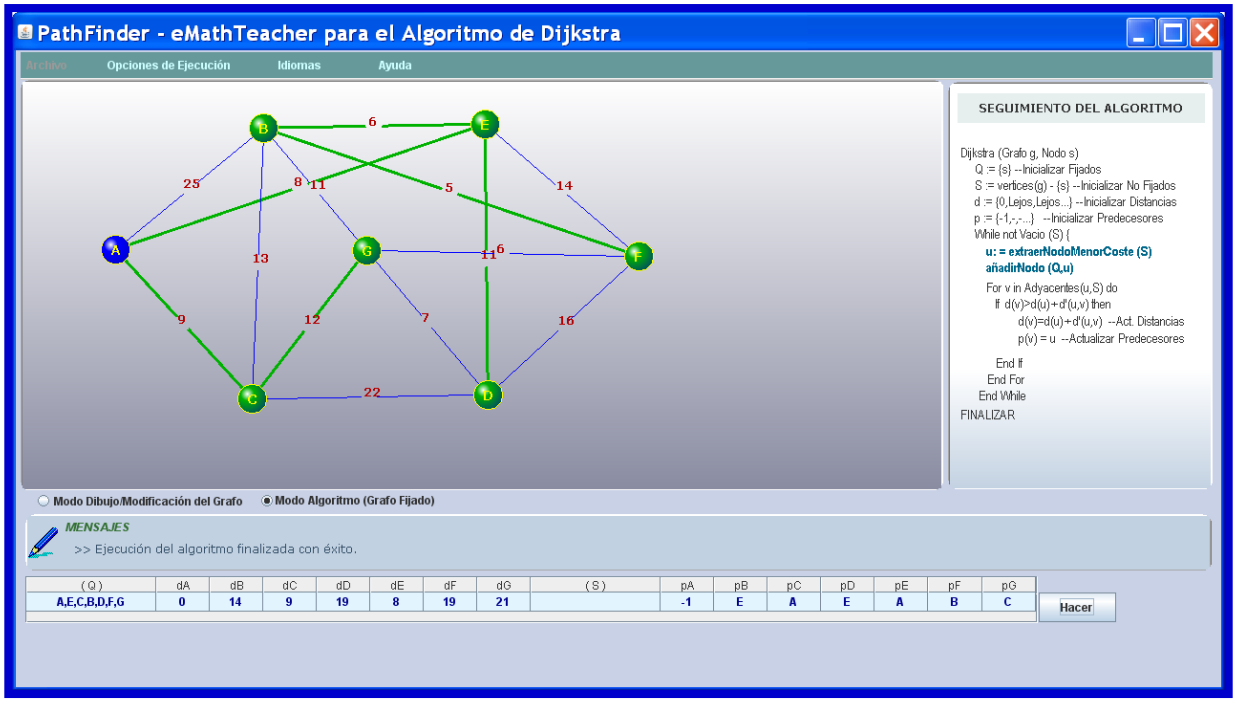


Figura 1 Pantalla típica de PathFinder

2.3 Minimum path

Es un sencillo “applet” de Java (ejecutable en un navegador) que utiliza el algoritmo de Dijkstra para encontrar el camino más corto entre un vértice y los restantes nodos de un grafo dirigido.

Ha sido creado por Roberto Dircio Palacios Macedo, ingeniero de sistemas computacionales por la Universidad de las Américas, Puebla, México.

Con un interfaz sumamente sencillo, este applet no permite el redimensionamiento: el tamaño del área de trabajo es un tanto pequeño y es el mismo independientemente de las dimensiones de la ventana del navegador. Algunos textos sobrepasan el área de maquetación y su diseño gráfico es sencillo. Por otro lado, el modo paso a paso sí muestra claramente cada iteración del bucle del algoritmo, coloreando cada vértice con distinto color según sea el vértice inicial, definitivo o en proceso.

La funcionalidad de la herramienta relativa a la creación del grafo es sencilla: añadir o borrar un vértice (siempre toma como inicial el primer nodo introducido), añadir o borrar una arista y modificar la valoración de un peso dentro del intervalo [0,100]. La herramienta indica la posibilidad del borrado de nodos y de aristas, pero esta función no ha ejcutado correctamente en el análisis efectuado con Google Chrome versión 18.0.

Permite en grafos dirigidos en los que un par de vértices sean incidentes a dos aristas, cada una con un sentido distinto, y muestra un grafo ejemplo como único método de ayuda.

La ejecución del algoritmo de Dijkstra se muestra coloreando y valorando cada uno de los vértices, de manera secuencial, bien de modo manual, bien de modo automático, hasta la consecución final del algoritmo.

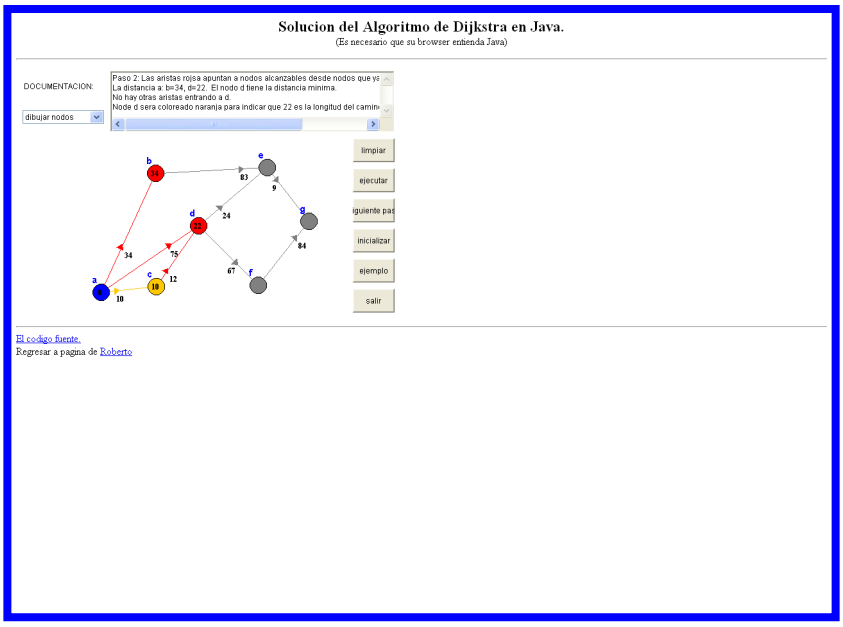


Figura 2 Paso 2º del algoritmo de Dijkstra en Minimum Path

2.4 Algraf Project

El Profesor D. Alberto Márquez Pérez, del departamento de Matemática Aplicada I de la Universidad de Informática de Sevilla, dirige el proyecto "Algraph Project: una herramienta para ayuda al desarrollo de las prácticas de las asignaturas dedicadas al estudio de los grafos y la investigación". Algraph Project ha dado lugar a los Proyectos Fin de Carrera de Rafael Borrego Ropero y Daniel Recio Domínguez.

Su nivel de "usabilidad" es bueno gracias a la barra de herramientas que cuenta con el conjunto suficiente de iconos, gráficamente claros, para la ejecución de los distintos problemas planteables. El grupo de iconos se amplía en la medida que el problema acepta diferentes opciones.

El usuario (que no será profano en la temática a tratar) aprende fácilmente el manejo de la herramienta pues es muy intuitiva. Permite el centrado del grafo en el área de trabajo, girar, acercar y alejar. Dispone de manuales de usuario claros -accesibles por medio de conexión a internet- en formato PDF, manuales de algoritmia, un grupo de preguntas frecuentes en Red Iris y un archivo con ejemplos y gráficos. El diseño gráfico es sencillo, cuestión común en este tipo de software orientado a personas del ámbito de la docencia.

La funcionalidad está orientada a diferentes tipos de grafos. El usuario puede incluir el número de vértices que desee, sus aristas y el valor del peso de cada arista. La herramienta también genera aleatoriamente el grupo de etiquetas, si es etiquetado, o el valor de los pesos. Permite guardar, abrir guardado, exportar como PDF, exportar como imagen. Tiene las opciones necesarias de creación, modificación y borrado de elementos. Presenta diferentes análisis de grafo: estadísticas, matriz de adyacencia y estudio de diferentes conectividades. Maneja distancias, árboles y coloraciones. Analiza si el grafo es euleriano y/o hamiltoniano con diferentes opciones de análisis. Para la valoración del camino más corto utiliza los algoritmos de Dijkstra y de Floyd. También estudia emparejamientos.

Es una aplicación residente, desarrollada en C# y ejecutable en entornos Microsoft Windows (*Win32*). Es un requerimiento el tener instalado el paquete Microsoft .NET Framework Versión 1.1 redistributable. La versión actual disponible del programa es la 1.1 de noviembre de 2006.

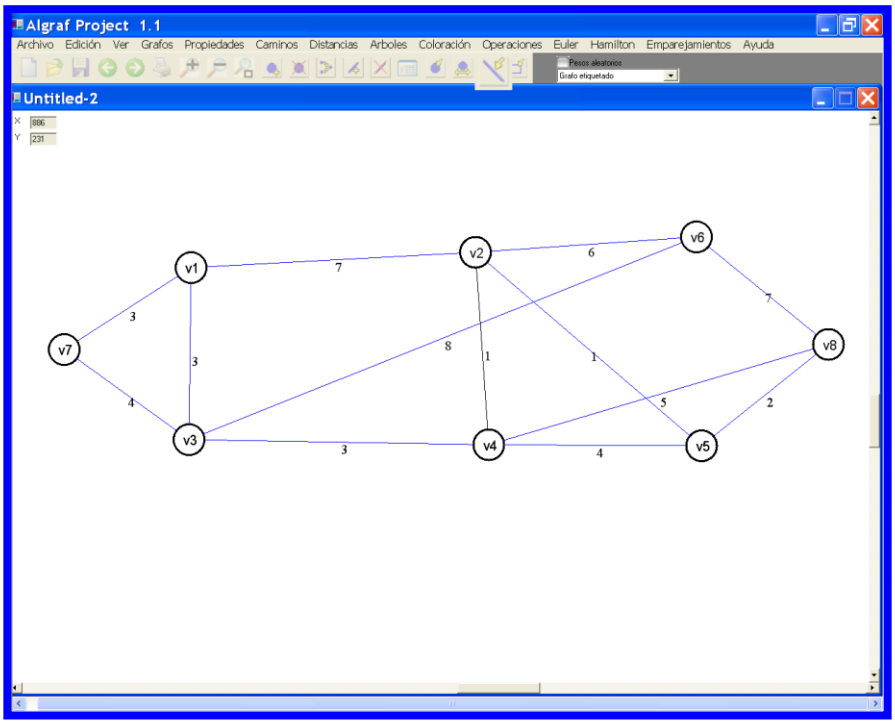


Figura 3 Algraf Project para un grafo no dirigido

2.5 Algraf: algoritmos sobre grafos

Muy similar a la aplicación anterior (posiblemente una evolución), pero elaborada por Fátima Rico (2001), José Luis Santisteban (2001), Abraham Fernández (2001) y Julián Ramírez (2002), bajo la dirección del profesor del departamento de Matemática Aplicada de la Facultad de Informática de la Universidad Politécnica de Madrid, D. Gregorio Hernández Peñalver. Dichos autores han elaborado una herramienta diseñada en Visual Basic para el estudio de los algoritmos sobre grafos. El objetivo de la aplicación es fundamentalmente didáctico: ayudar a la mejor comprensión de conceptos y para visualizar de forma animada algoritmos sobre grafos.

El programa responde a numerosas cuestiones sobre un grafo dirigido o no dirigido y/o ponderado o no ponderado (con o sin pesos en las aristas): sucesión de grados, matriz de adyacencia, conectividad, vértices corte, aristas puente, componentes conexas y bloques, grafo complementario y grafo de aristas (“*line graph*”), árboles (Prim, Kruskal y Boruvka), código de Prüfer de un árbol etiquetado, caminos en un grafo dirigido o no dirigido y/o ponderado y no ponderado, recorridos eulerianos (Hierholzer, Fleury y Tucker), problema del cartero, algoritmos secuenciales, variantes y algoritmo de Brelaz.

Los requisitos y entorno de trabajo de la aplicación son los mismos que los de la aplicación Algraf Project pero en su versión 1.0, si bien ALgoritmos sobreGRAFOS ha sido desarrollada en Visual Basic y por ello necesita de varios archivos controladores que son facilitados por el departamento, en la misma página en la que se puede descargar la última versión de Mayo de 2002 y sus archivos de ejemplo.

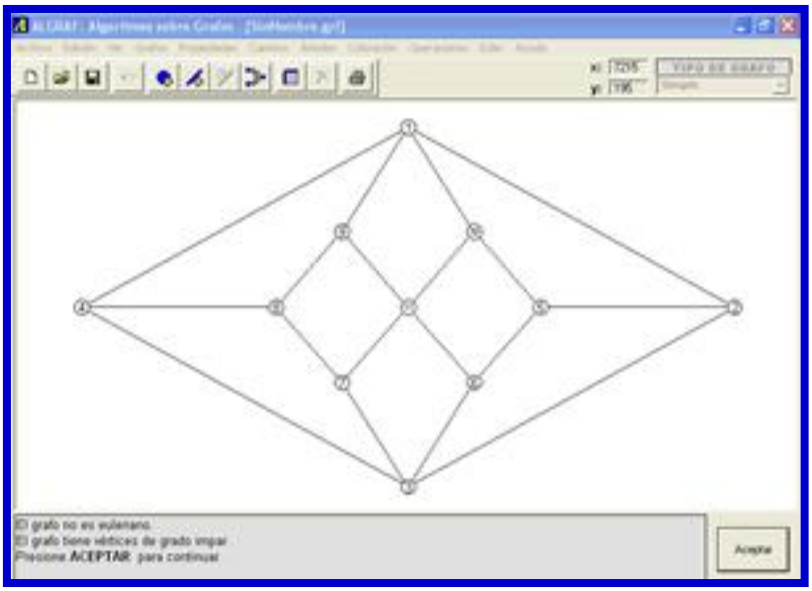


Figura 4 Algraf en un grafo de 11 vértices

2.6 Dijkstra Algorithm

Es una pequeña aplicación realizada por el fotógrafo y programador Chris Wenk que utiliza del algoritmo de Dikjstra para la búsqueda del camino más corto entre dos vértices y que consta de un único ejecutable (por tanto no necesita instalación) para sistemas operativos Microsoft Windows 98|2000|7|XP|Vista. La última versión en el momento de creación de este proyecto es la 1.9.

Este es un software libre que se distribuye sin coste alguno y por tiempo ilimitado en el que el autor declina cualquier tipo de mantenimiento y/o garantías de uso, así que, en algunos casos, se puede obtener en diferentes de páginas web poco fiables, en las que le han añadido programas de tipo "*malware*".

Su interfaz es pobre: consta de una ventana de 320 x 420 píxeles, reduciendo el área de trabajo a 140 x 200 píxeles.

Su nivel de funcionalidad está restringido a la creación de grafos no dirigidos, definibles por el usuario en cuanto a sus vértices y aristas, pero no permite la modificación de los pesos de dichas aristas; los peso son asignados aleatoriamente.

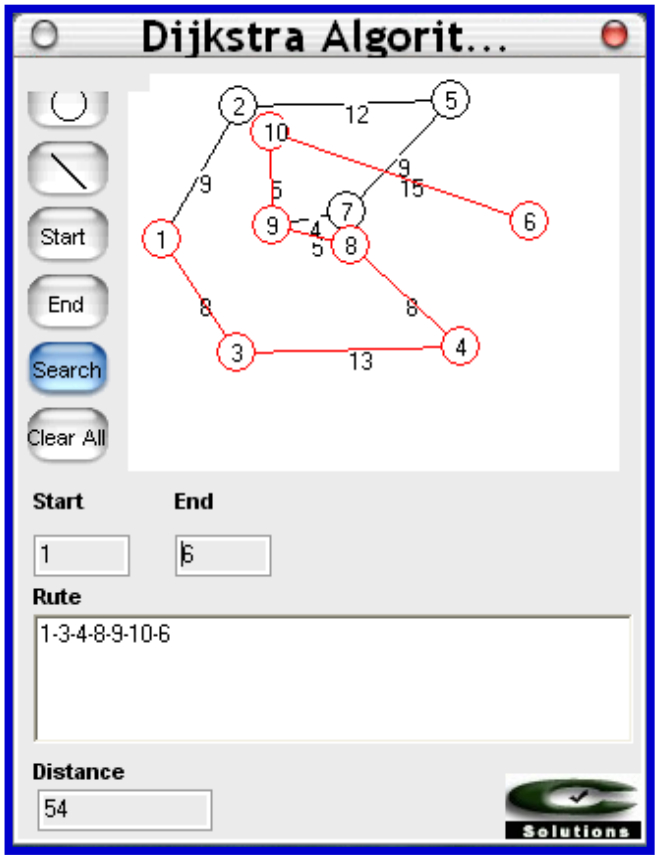


Figura 5 Dijkstra Algorithm, a tamaño real

2.7 Grafos

Grafos, es un programa completamente gratuito y no tiene ninguna limitación funcional ni de uso en el tiempo. Se distribuye sin garantías, bajo las condiciones de la licencia Creative Commons Licenses (by-nc-sa), y está permitida la distribución gratuita en repositorios o foros de instituciones educativas o Universidades. Su creador es el Prof. Dr. Alejandro Rodríguez Villalobos del departamento de Organización de Empresas de la Escola Politècnica Superior d'Alcoi, en la Universitat Politècnica de València. Este software está pensado para el aprendizaje de la teoría de grafos y la docencia de otras disciplinas relacionadas.

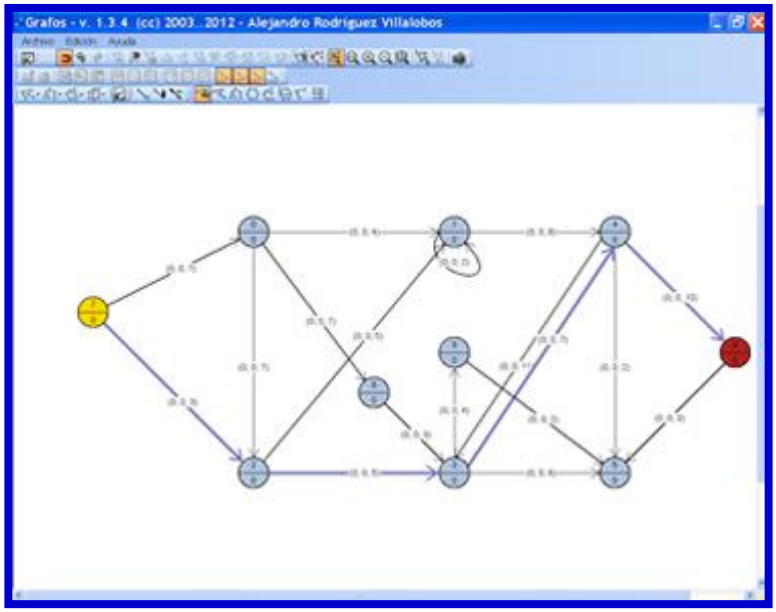


Figura 6 Grafos con un grafo dirigido y con bucles

Dispone de breves manuales en línea para su instalación y puesta en marcha. También cuenta con manuales ampliados en formato de libro electrónico y en papel, pero ambos son de pago. Su diseño es cuidado y su "usabilidad" es compleja, dado el número de funciones disponibles: análisis de flujos (Ford-Fulkerson, MILP), de árboles generadores de peso mínimo (Kuskal, Prim), de caminos (Dijkstra, Bellman-Ford, Floyd-Warshal), y de rutas (MILP, CVRP). Permite dibujar el grafo sobre una imagen prediseñada (mapas, etc.), importar y exportar los datos (podemos guardar el problema en el formato estándar *GraphML* y en formato CSV –importación en modo tabular-), exportar el grafo como imagen, cuenta con opción de impresión, redistribución del grafo (las aristas se reconfiguran). Dispone de dos tipos de vista: gráfica y en modo tabla.

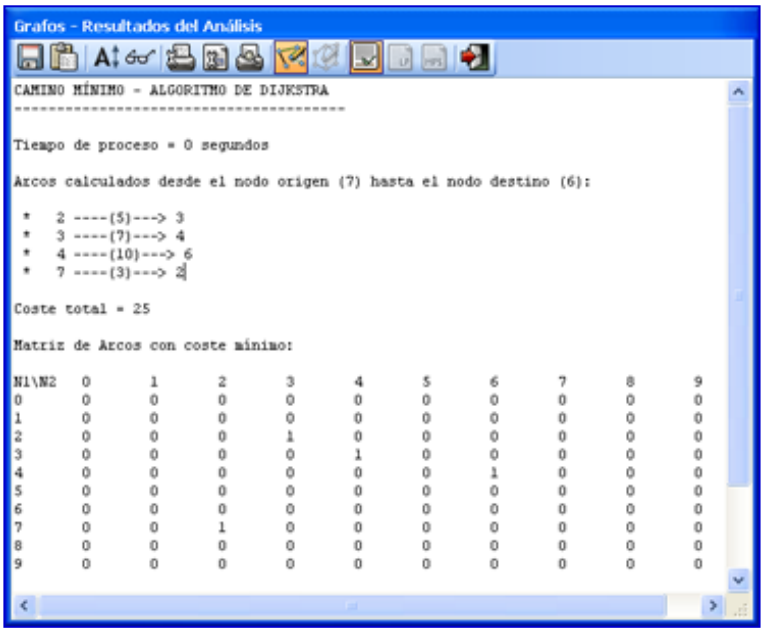


Figura 7 Grafos: resultado analítico del algoritmo de Dijkstra

Ésta es una aplicación residente, desarrollada en Microsoft Visual Studio 2010 .NET, disponible en diferentes versiones para sistemas operativos Microsoft Windows XP|Vista|7|2000. Requiere tener preinstalado *.NET Framework 3.5 redistributable* para poder funcionar en su última versión 1.3.0.

Capítulo 3

Análisis, diseño e implementación

3.1 Introducción

En este capítulo del proyecto se estudia el ciclo de vida del software. Se ha elegido el modelo en cascada por ser el paradigma adecuado para este tipo de situaciones en las que se deben superar las pruebas para la adaptación al nuevo estándar HTML5, obligando a un rediseño si fuere necesario y una nueva codificación.

3.2 Análisis

3.2.1 Introducción

Esta sección describe el análisis de software para la posterior implementación del código. Para ello se han obtenido los requisitos y requerimientos relativos al programa que se desea elaborar. Se recogen los requisitos del usuario y los requisitos de funcionalidad.

3.2.2 Requisitos de usuario

Desde la adjudicación del proyecto se han recogido las conversaciones mantenidas con los profesores Eduardo Jesús Sánchez Villaseñor y Jesús Sallas Martínez, con el fin de definir las necesidades de la herramienta de software a fabricar. Requisitos que a continuación se enumeran, han sido recogidos en lenguaje natural. Hemos separado el tipo de requisitos en dos grandes grupos: funcionales y no funcionales.

3.2.2.1 Requisitos no funcionales

- La arquitectura será cliente-servidor.
- La aplicación podrá ser ofrecida por las computadoras residentes en el departamento con procesadores de tipo x86 ó por cualquier otra máquina que adquiera el departamento, siempre y cuando su procesador sea superior a en capacidad de proceso al de los modelos Intel Pentium o AMD K5 y conste de servidor web con protocolo HTTP.
- La herramienta será ejecutable para computadoras cliente con cualquier sistema operativo: Microsoft Windows, Linux, Macintosh, etc.
- No será necesaria la instalación de ningún archivo complementario a la apropia aplicación (controladores, entornos de desarollo, etc.) que no sea un navegador para la ejecución de la misma.
- La herramienta será totalmente autónoma e independiente de cualquier otro sistema de información.
- No se guardará ningún dato de los usuarios, en referencia a la Ley de Protección de datos, por tanto no son necesarios requerimientos de seguridad por parte de la herramienta, exceptuando efectuar una copia de seguridad de la propia herramienta.
- El número de usuarios que soliciten la herramienta al servidor estará limitado por las capacidades de dicho servicio, esto es, el ancho de banda, la memoria, capacidad de proceso y capacidad de gestión de usuarios por parte de la máquina servidora.
- La herramienta será funcional para las últimas versiones de navegadores de código abierto:
  - Mozilla Firefox,
  - Google Chrome.
- Si el usuario ha guardado la página web de la herramienta en su propia máquina, la podrá utilizar mediante el navegador sin tener que volver a conectar con el servidor. Por tanto, no se ejecutará código alguno en el servidor, excepto las propias peticiones HTTP solicitantes para que el cliente acceda a la herramienta bien para su ejecución directa, bien para guardarla.
- La herramienta será eminentemente gráfica, donde el usuario pueda dibujar el grafo e introducir fácilmente los pesos, vértices y aristas.
- El usuario podrá crear un grafo a su gusto, esto es: con el número de nodos que desee y el número de aristas que desee.

3.2.2.2 Requisitos funcionales

- Las aristas serán valoradas por pesos, los cuales serán indicados por el usuario.
- El grafo podrá ser dirigido o no dirigido.
- Los grafos no dirigidos serán simples, ya que las multiaristas y bucles se eliminan de manera trivial.
- Todas las aristas estarán valoradas con pesos positivos.
- El usuario indicará el vértice origen y el vértice destino a fin de resolver el problema del camino más corto entre ambos vértices, sin construir el árbol completo.
- La herramienta deberá resolver el problema con le algoritmo de Dijkstra si así se lo solicita el usuario
- La herramienta permitirá al usuario, dentro de lo posible, rectificar una acción ejecutada, como por ejemplo:
  - borrar un vértice (excepto si está unido por alguna arista),
  - modificar el valor del peso de una arista,
  - cambiar de grafo dirigido a no dirigido,
  - en la construcción del camino más corto:
    - borrar una arista seleccionada,
    - reordenar las aristas seleccionadas.
- La herramienta mostrará, cuando sea solicitado por el usuario y antes de la resolución, indicativos analíticos del problema:
  - matriz de adyacencia,
  - matriz analítica.
- Se dispondrá de una ayuda en línea para el aprendizaje de la herramienta. Para ello es requisito necesario el que el navegador contenga el complemento de Adobe Flash Player, disponible gratuitamente en la dirección http://get.adobe.com/es/flashplayer/. También podrá acceder a Internet, si dispone de conexión el usuario, para obtener cualquier otra ayuda complementaria que pueda ser útil para el usuario en el estudio y|o resolución del problema:
  - apuntes,
  - documentación en general,
  - vídeos, etc.

A continuación se especificarán los requisitos de usuario dividiéndolos en dos categorías, requisitos de usuario funcionales o de capacidad y requisitos de usuario restrictivos. Estos requisitos se especifican a continuación siguiendo un formato tabular en el cual se incluye el siguiente contenido para cada uno de ellos:

- ID: indica de manera unívoca a un requisito. Nomenclatura: RU_CAP_XX para los requisitos capacitivos y RU_RES_XX para los restrictivos; donde XX corresponde al número de requisito.
- DESCRIPCIÓN: especificación del requisito.
- NECESIDAD: relevancia del requisito para el proyecto. Nomenclatura: valores entre 1 y 5 siendo 5 la necesidad más alta y 1 la más baja.
- PRIORIDAD: establece la prioridad del requisito dentro del proyecto. Nomenclatura: valores entre 1 y 5 siendo 5 la prioridad más alta y 1 la más baja.

- ESTABILIDAD: indica la sensibilidad del requisito a ser modificado. Nomenclatura: "Estable" y "No Estable".

3.2.2.2.1 Requisitos de capacidad

ID	RU_CAP_01 – CreaNodo
DESCRIPCIÓN	El usuario podrá crear un vértice haciendo doble click
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_02 – IniciarArista
DESCRIPCIÓN	El usuario indicará con el ratón el vértice donde se inicia la arista. Si el grafo es dirigido, éste corresponde al vértice inicial de la arista.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_03 – TerminarArista
DESCRIPCIÓN	El usuario indicará con el ratón, el vértice (final si el grafo es dirigido) donde termina la arista.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_04 – ModificaPesoArista
DESCRIPCIÓN	El usuario podrá cambiar el valor del peso de la arista en el valor indicado en ella
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_05 – AsignaDirigido
DESCRIPCIÓN	Indica si el grafo es dirigido o no dirigido. Podrá seleccionarse en cualquier momento antes de la resolución del problema.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_06 – AsignaNodoInicial
DESCRIPCIÓN	El usuario indicará el identificador del nodo inicial antes de la resolución del problema.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_07 – AsignaNodoFinal
DESCRIPCIÓN	El usuario podrá borrar con el ratón un vértice, siempre y cuando no sea incidente con ninguna arista.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_08 – BorraPunto
DESCRIPCIÓN	El usuario podrá borrar, con el ratón, un vértice siempre y cuando dicho vértice no contenga aristas.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_09 – empezar
DESCRIPCIÓN	El usuario seleccionará una arista, previamente no seleccionada ya, para incluirla en el camino más corto. Se utilizará el método “arrastrar y soltar”.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_10 – recolocar
DESCRIPCIÓN	El usuario podrá reordenar la posición de orden de una arista dentro de las elegidas para el camino más corto. “Arrastrar y soltar”.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_11 – borrar
DESCRIPCIÓN	El usuario podrá eliminar una arista de las elegidas para el camino mínimo de dicho conjunto. “Arrastrar y soltar”.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_12 – MuestraMatrizAdyacencia
DESCRIPCIÓN	La herramienta generará y mostrará la matriz de adyacencia valorada con pesos.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_13 – ConstruyeCaminoMasCorto
DESCRIPCIÓN	La herramienta generará la matriz analítica ejecutando el algoritmo de Dijkstra.
NECESIDAD	5
PRIORIDAD	5

ESTABILIDAD	Estable

ID	RU_CAP_14 – ConstruyeCaminoAristas
DESCRIPCIÓN	La herramienta ejecutará el algoritmo de Dijkstra y dibujará sobre el propio grafo la solución. También mostrará de forma ordenada el conjunto de aristas seleccionadas para dicho camino, así como las matrices analítica y de adyacencia.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_15 – CambiaTutorial
DESCRIPCIÓN	La herramienta mostrará ayuda en línea si hay conexión a Internet. Lo hará en un IFRAME, sin modificar el tamaño del grafo.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_CAP_16 – MuestraNavegación
DESCRIPCIÓN	La herramienta mostrará en un IFRAME el acceso a cualquier página de Internet, si hubiere conexión, sin modificar el tamaño del grafo.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

3.2.2.2.2 Requisitos de restricción

ID	RU_RES_01 – HTML5
DESCRIPCIÓN	La aplicación se codificará en HTML5 y JavaScript con CSS3 para el diseño
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_02 – Idioma de la interfaz
DESCRIPCIÓN	El español será el idioma por defecto
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_03- Navegadores de código abierto
DESCRIPCIÓN	La aplicación podrá ejecutarse sobre los navegadores Mozilla Firefox V 10 y Google Chrome V 18
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_04- JavaScript
DESCRIPCIÓN	El navegador deberá tener habilitada la ejecución de JavaScript
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_05 – Grafo sin lazos
DESCRIPCIÓN	No se permitirán lazos para ningún vértice
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_06 – Doble arista
DESCRIPCIÓN	No se permitirá al usuario crear dos aristas entre dos mismos vértices
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_07 – Borrado de Vértices con arista
DESCRIPCIÓN	No se permitirá borrar vértices que contengan alguna arista
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_08 – Resolución del camino más corto
DESCRIPCIÓN	No se calculará el camino más corto si en número de vértices o de aristas es menor que 2 o si no se han asignado los vértices inicial y final.
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_9 – Conexión a Internet
DESCRIPCIÓN	Si el usuario desea acceder a información contenida en Internet para complementar el aprendizaje y que sea mostrado el área correspondiente IFRAME
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_10 – Complemento Adobe Flash Player
DESCRIPCIÓN	Para poder ver los video tutoriales de aprendizaje de la herramienta, el navegador deberá tener instalado el correspondiente complemento. Se mostrará en el área IFRAME
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

ID	RU_RES_11 – Archivos de videotutorial

DESCRIPCIÓN	Si el usuario no dispone de conexión a Internet, deberá descargar la versión de la herramienta que contiene los archivos tipo "*flash*" de aprendizaje, para disponer de ellos en la carpeta "help".
NECESIDAD	5
PRIORIDAD	5
ESTABILIDAD	Estable

3.2.3 Requisitos Tecnológicos

La herramienta será ejecutable sobre cualquier computadora con 256 MB de RAM como mínimo, con una tarjeta gráfica con capacidad mínima de 800 x 600 píxeles de resolución, pero se recomiendan computadoras con 512 MB de RAM y capacidad para resoluciones gráficas de 1280 x 1024 píxeles.

El sistema operativo utilizado no está condicionado, mientras que los navegadores pueden ser Mozilla Firefox desde su versión 3.5 o Google Chrome desde versión 1.0 ya que estas versiones soportan CSS3, JavaScript y la etiqueta <CANVAS> de HTML5, un requisito éste último, totalmente necesario para la ejecución de GrafoMin.

3.2.4 Requisitos de Desarrollo

Para desarrollar el presente proyecto, además de los requisitos tecnológicos citados anteriormente, será necesario contar para el desarrollo de la aplicación, el complemento Firebug Lite 1.4 para Google Chrome o bien, Firebug 1.9.2 para el caso de Mozilla Firefox. Los citados complementos son los utilizados por las últimas versiones de los anteriores navegadores; para versiones anteriores consultar en la dirección del propietario del complemento Firebug.

Si el usuario desea obtener el paquete de archivos GrafoMin podrá copiarlo o ejecutarlo directamente, como desee, desde la copia residente en el departamento http://euler.uc3m.es/matematicadiscreta/, o bien accediendo a la dirección de internet www.yottabyte.es/discreta/. Para los accesos a Internet, deberá contar con conexión y tarjeta de red.

Para la elaboración de la documentación del presente proyecto se ha utilizado Microsoft Word 2007, Microsoft Project y Microsoft Microsoft Visio 2010, sobre un sistema operativo Microsoft Windows XP Services Pack 3.

3.2.5 Requisitos de la interfaz

A continuación de describen los requisitos aplicables a los intercambios de información entre GrafoMin y el usuario.

[RI 1] La herramienta podrá trabajar con resoluciones de 800x600 píxeles, pero se recomiendan 1280x1024 píxeles para mantener el diseño uniforme y, la capacidad de mostrar

en la misma pantalla la información adicional de ayuda IFRAME, junto con la funcionalidad de creación del grafo y su camino más corto.

[RI 2] Se observarán los criterios de usabilidad y sencillez, así como las funciones intuitivas de los modos ventana: "*click*" de ratón, arrastrar y soltar, icono del ratón según sus posibles funciones sobre cada elemento, etc.

[RI 3] Se ofrecerán tanto una ayuda continua utilizando cuadros de diálogo tipo *globo* para la creación del grafo, así como vídeos de aprendizaje para el manejo de GrafoMin.

[RI 4] El idioma designado para la herramienta será el castellano.

[RI 5] Los campos que sean de carácter obligatorio en su inserción de datos por parte del usuario, se indicarán por medio de cuadros de diálogo o bien por medio de imágenes descriptivas.

[RI 6] Se indicarán, claramente, las funciones no disponibles y los errores cometidos por el usuario por medio de textos en cuadros de diálogo.

[RI 7] Se respetarán los estándares de diseño Web en cuanto a maquetación, utilizando hojas de estilo en cascada.

[RI 8] Se indicarán en el pie de página de la herramienta los requisitos [RI 1] y [RI 7].

3.3 Diseño

3.3.1 Introducción

En este capítulo, dedicado al diseño del presente proyecto, definiremos cómo se cumplirán los requisitos citados en el capítulo anterior. Esto es, antes de proceder a la programación de la herramienta, deberemos conocer cuál será la estructura que deberá adoptar el sistema de software GrafoMin.

3.3.2 Diseño de la arquitectura

3.3.2.1 Casos de Uso

Los casos de uso nos identificarán el comportamiento de la herramienta ante el usuario. Para mostrar dicha interacción, utilizaremos gráficos UML generados por la herramienta Microsoft Visio 2010.

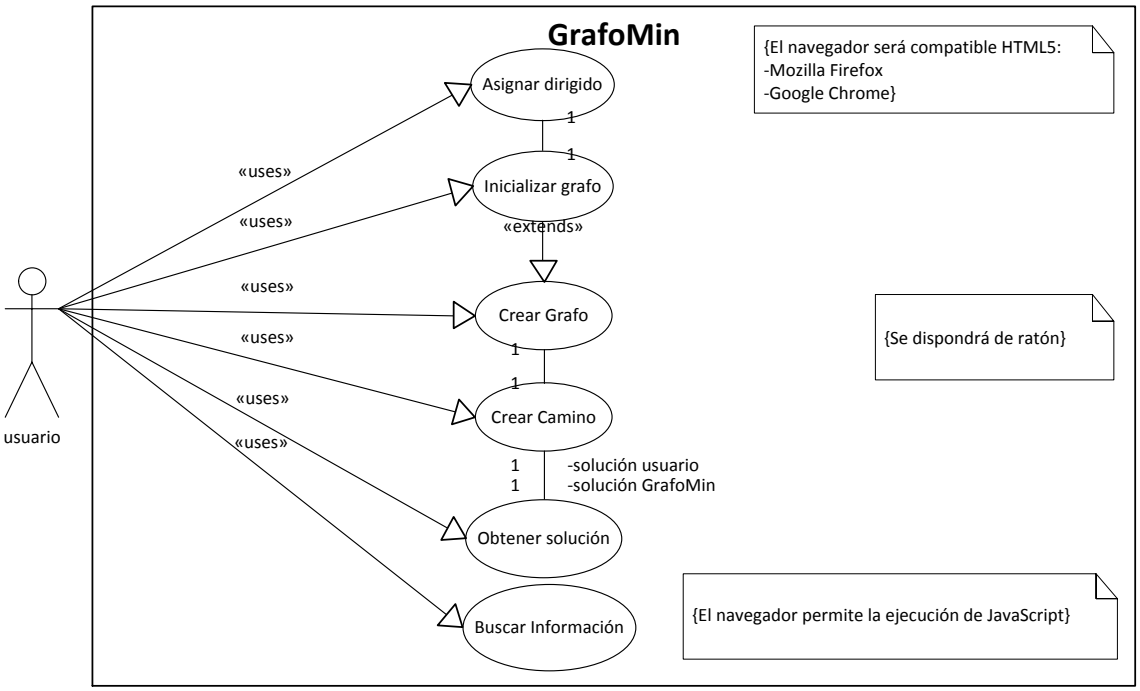


Figura 8 Caso de Uso 1

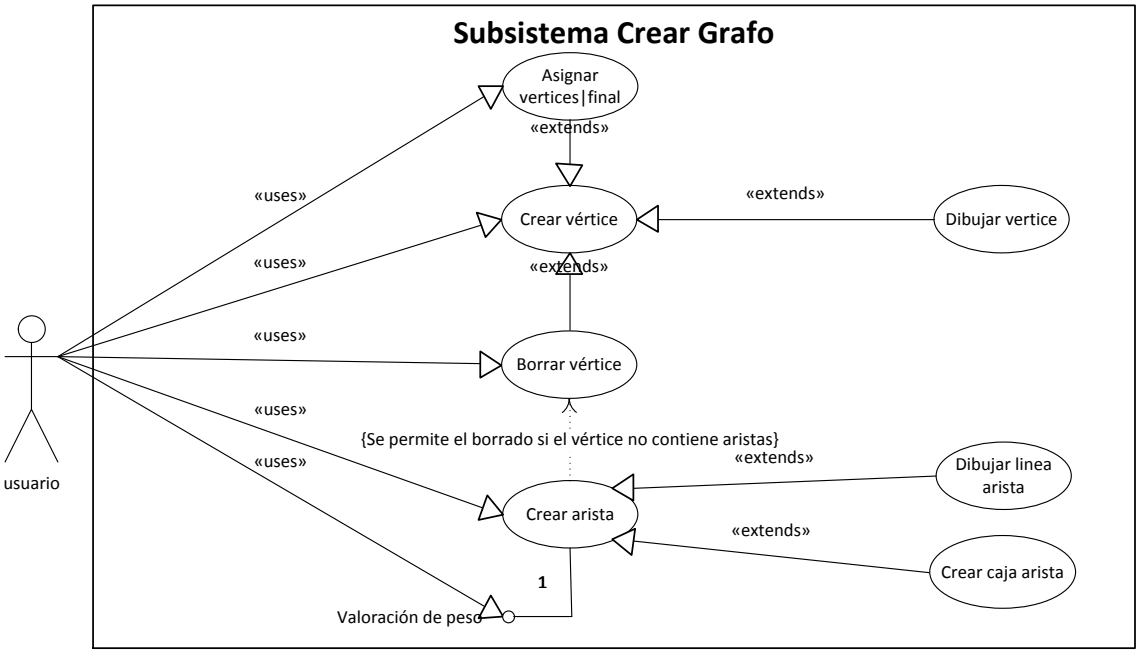


Figura 9 Caso de Uso 2

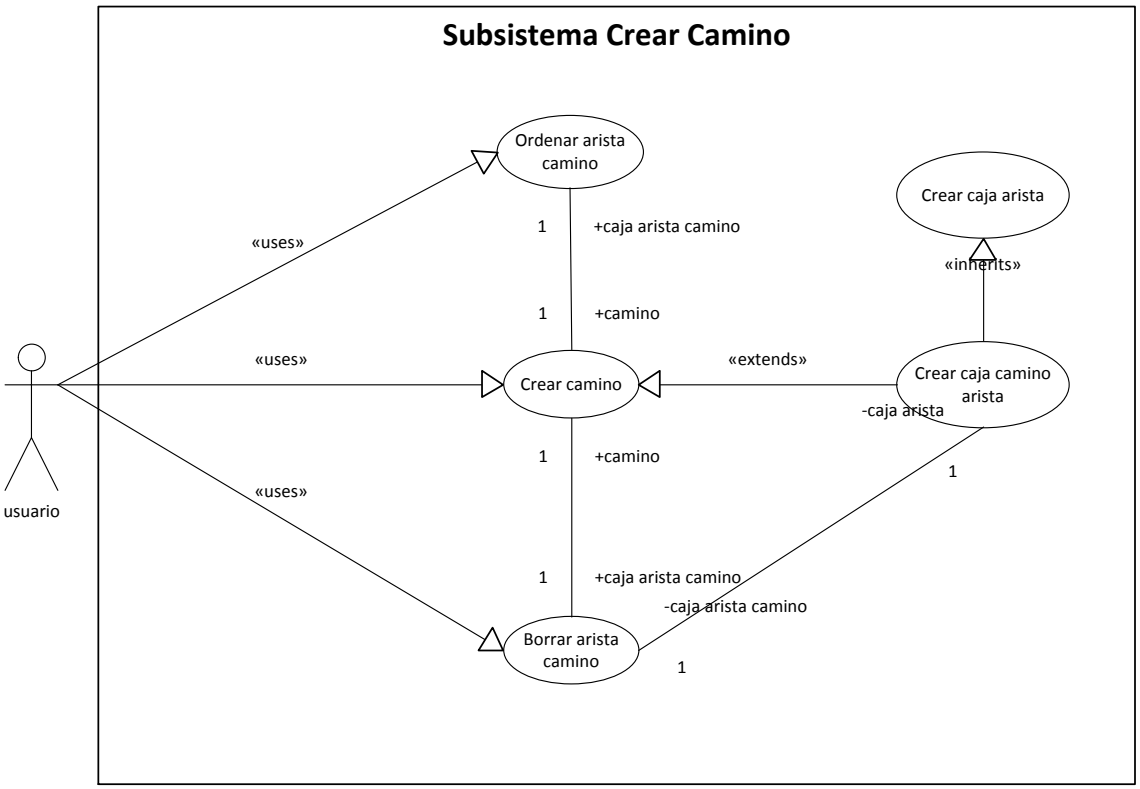


Figura 10 Caso de uso 3

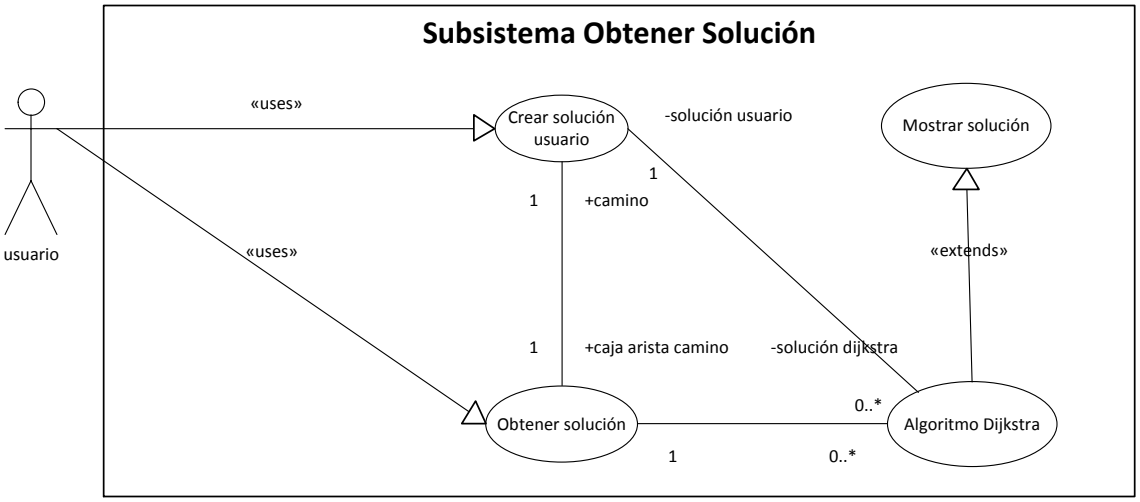


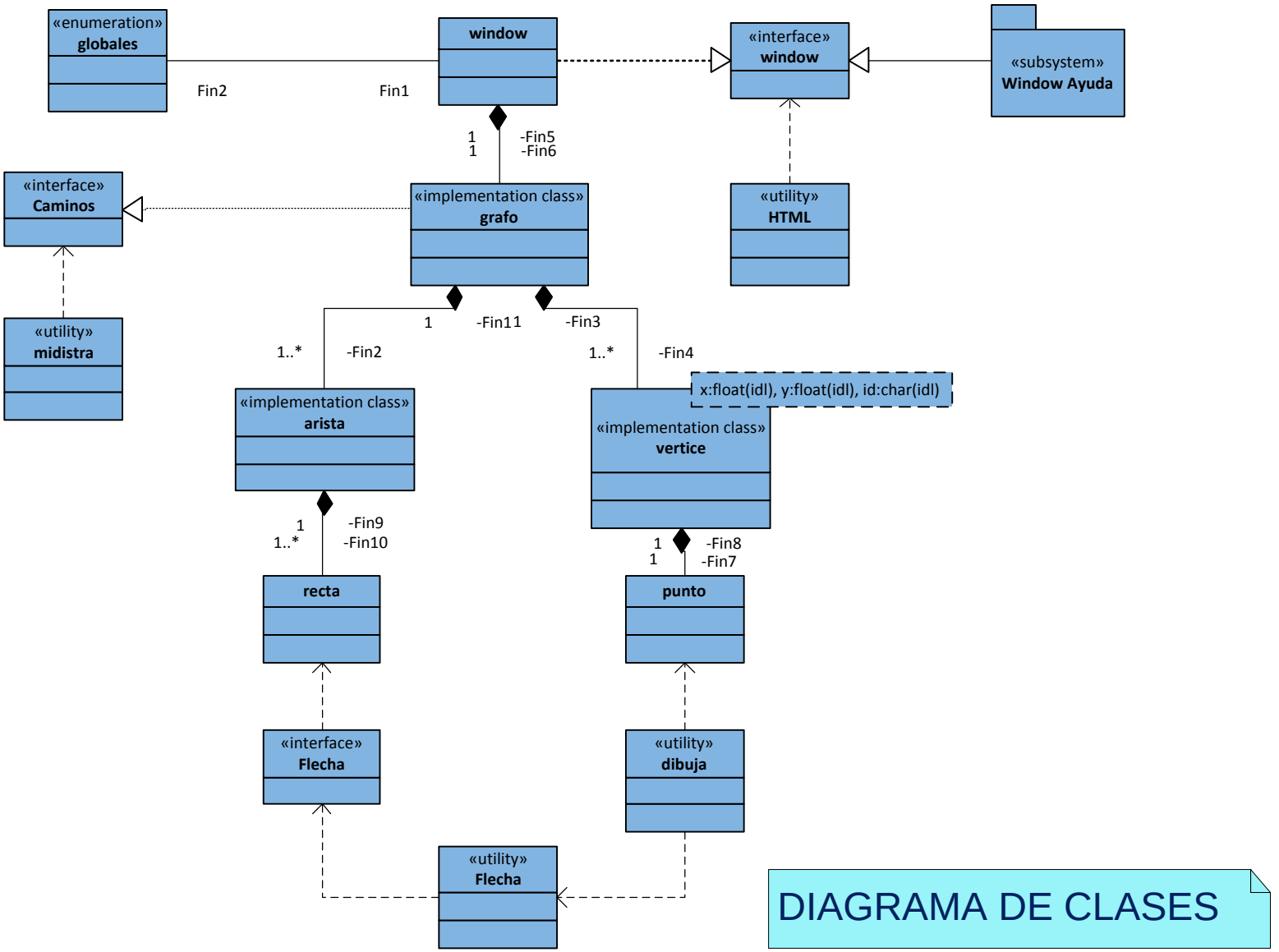
Figura 11 Caso de Uso 4

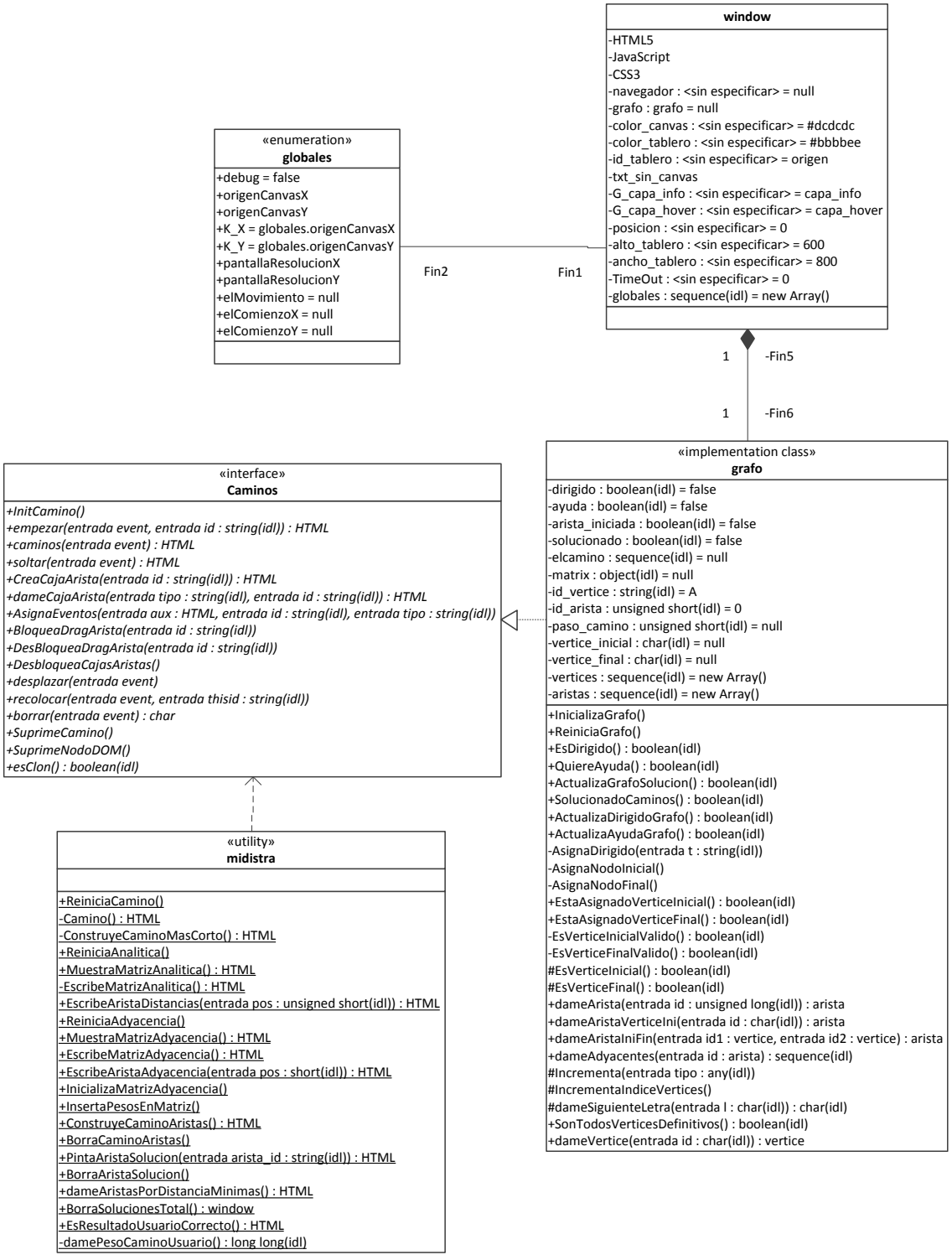
3.3.2.2 Diagrama de clases

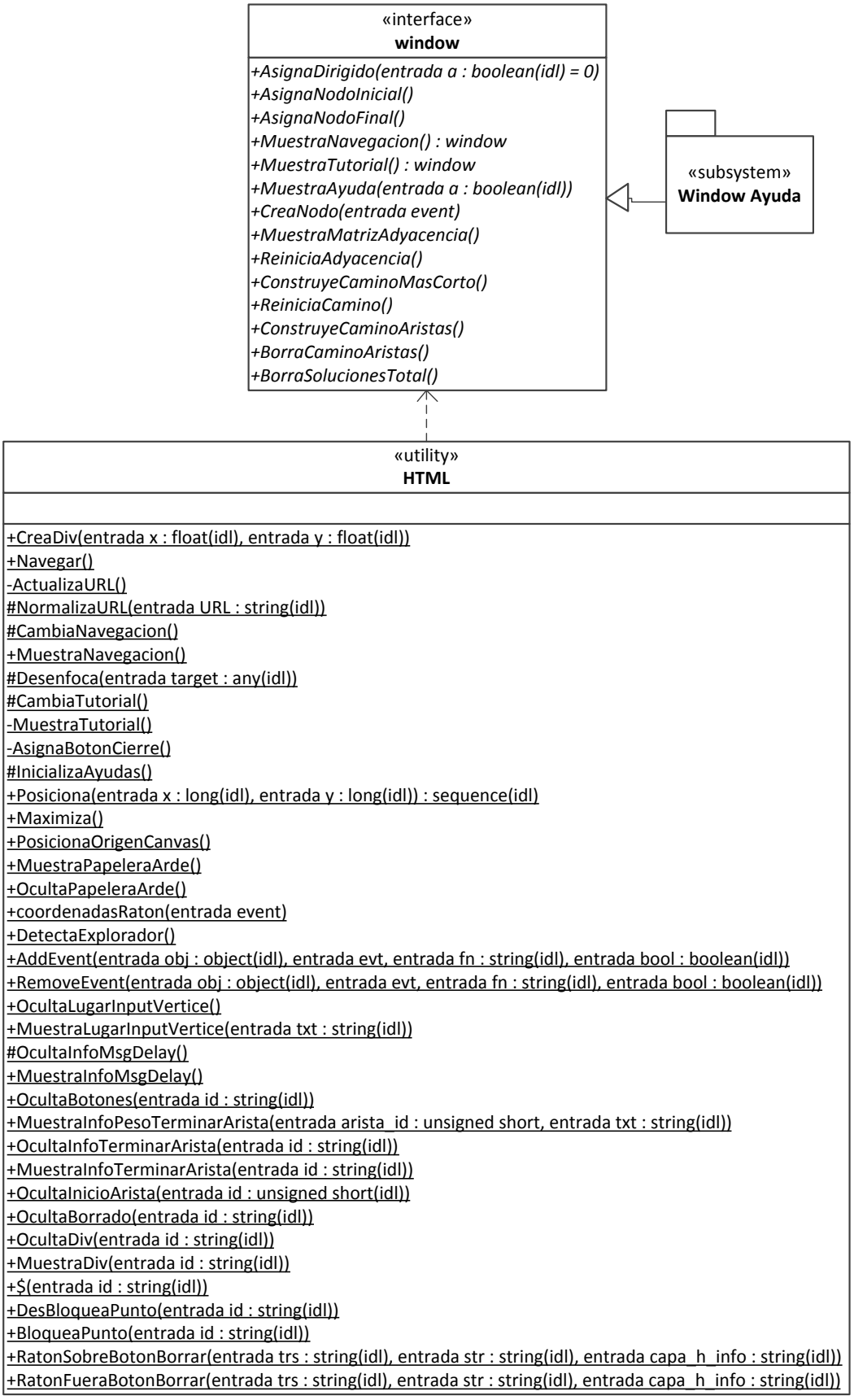
Los diagramas de clases describen las relaciones entre las clases que involucradas en el sistema. Utilizando el estándar UML, el objetivo de estos diagramas es el de presentar las relaciones entre las clases. GrafoMin está implementada con JavaScript como motor de la herramienta. JavaScript no es un lenguaje orientado a objetos propiamente dicho, pero contiene importantes semejanzas. JavaScript es un lenguaje basado en prototipos que no contiene

ninguna declaración de clase, como se encuentra, por ejemplo, en C++ o Java. Esto es a veces confuso para los programadores acostumbrados a los lenguajes con una declaración de clase. En su lugar, JavaScript utiliza funciones como clases. Mediante la herramienta Microsoft Visio 2010 se ha elaborado el diagrama de clases detallado en la página siguiente.

Por cuestiones de capacidad e integridad, con el fin de no perder las relaciones del diagrama, hemos insertado dicho diagrama en diferentes bloques. Podrá observarlo de modo íntegro, utilizando la herramienta Microsoft Visio 2010 con el fichero adjunto en el anexo, donde podrá además ver cada uno de los atributos, el código en JavaScript de cada método. Si desea ampliar la imagen para una visión más cómoda, pulse la tecla CTRL y gire modo ascendente la rueda de su ratón a un mismo tiempo, hasta observar el gráfico a su gusto; podrá volver a su vista anterior, pulsando la tecla CTRL y, sin soltar, girando en modo descendente la rueda de su ratón.


«enumeration»
globales
window
«interface»
window
«subsystem»
Window Ayuda
Fin2
Fin1
1
1
-Fin5
-Fin6
«interface»
Caminos
«implementation class»
grafo
«utility»
HTML
«utility»
midistra
1
-Fin11
-Fin3
1..*
-Fin2
1..*
-Fin4
x:float(idl), y:float(idl), id:char(idl)
«implementation class»
arista
«implementation class»
vertice
1
1..*
-Fin9
-Fin10
1
1
-Fin8
-Fin7
recta
punto
«interface»
Flecha
«utility»
dibuja
«utility»
Flecha
DIAGRAMA DE CLASES


window
-HTML5
-JavaScript
-CSS3
-navegador : <sin especificar> = null
-grafo : grafo = null
-color_canvas : <sin especificar> = #dcdcdc
-color_tablero : <sin especificar> = #bbbbee
-id_tablero : <sin especificar> = origen
-txt_sin_canvas
-G_capa_info : <sin especificar> = capa_info
-G_capa_hover : <sin especificar> = capa_hover
-posicion : <sin especificar> = 0
-alto_tablero : <sin especificar> = 600
-ancho_tablero : <sin especificar> = 800
-TimeOut : <sin especificar> = 0
-globales : sequence(idl) = new Array()
«enumeration»
globales
+debug = false
+origenCanvasX
+origenCanvasY
+K_X = globales.origenCanvasX
+K_Y = globales.origenCanvasY
+pantallaResolucionX
+pantallaResolucionY
+elMovimiento = null
+elComienzoX = null
+elComienzoY = null
Fin2
Fin1
1
-Fin5
1
-Fin6
«implementation class»
grafo
-dirigido : boolean(idl) = false
-ayuda : boolean(idl) = false
-arista_iniciada : boolean(idl) = false
-solucionado : boolean(idl) = false
-elcamino : sequence(idl) = null
-matrix : object(idl) = null
-id_vertice : string(idl) = A
-id_arista : unsigned short(idl) = 0
-paso_camino : unsigned short(idl) = null
-vertice_inicial : char(idl) = null
-vertice_final : char(idl) = null
-vertices : sequence(idl) = new Array()
-aristas : sequence(idl) = new Array()
+InicializaGrafo()
+ReiniciaGrafo()
+EsDirigido() : boolean(idl)
+QuiereAyuda() : boolean(idl)
+ActualizaGrafoSolucion() : boolean(idl)
+SolucionadoCaminos() : boolean(idl)
+ActualizaDirigidoGrafo() : boolean(idl)
+ActualizaAyudaGrafo() : boolean(idl)
-AsignaDirigido(entrada t : string(idl))
-AsignaNodoInicial()
-AsignaNodoFinal()
+EstaAsignadoVerticeInicial() : boolean(idl)
+EstaAsignadoVerticeFinal() : boolean(idl)
-EsVerticeInicialValido() : boolean(idl)
-EsVerticeFinalValido() : boolean(idl)
#EsVerticeInicial() : boolean(idl)
#EsVerticeFinal() : boolean(idl)
+dameArista(entrada id : unsigned long(idl)) : arista
+dameAristaVerticeIni(entrada id : char(idl)) : arista
+dameAristaIniFin(entrada id1 : vertice, entrada id2 : vertice) : arista
+dameAdyacentes(entrada id : arista) : sequence(idl)
#Incrementa(entrada tipo : any(idl))
#IncrementaIndiceVertices()
#dameSiguienteLetra(entrada l : char(idl)) : char(idl)
+SonTodosVerticesDefinitivos() : boolean(idl)
+dameVertice(entrada id : char(idl)) : vertice
«interface»
Caminos
+InitCamino()
+empezar(entrada event, entrada id : string(idl)) : HTML
+caminos(entrada event) : HTML
+soltar(entrada event) : HTML
+CreaCajaArista(entrada id : string(idl)) : HTML
+dameCajaArista(entrada tipo : string(idl), entrada id : string(idl)) : HTML
+AsignaEventos(entrada aux : HTML, entrada id : string(idl), entrada tipo : string(idl))
+BloqueaDragArista(entrada id : string(idl))
+DesBloqueaDragArista(entrada id : string(idl))
+DesbloqueaCajasAristas()
+desplazar(entrada event)
+recolocar(entrada event, entrada thisid : string(idl))
+borrar(entrada event) : char
+SuprimeCamino()
+SuprimeNodoDOM()
+esClon() : boolean(idl)
«utility»
midistra
+ReiniciaCamino()
-Camino() : HTML
-ConstruyeCaminoMasCorto() : HTML
+ReiniciaAnalitica()
+MuestraMatrizAnalitica() : HTML
-EscribeMatrizAnalitica() : HTML
+EscribeAristaDistancias(entrada pos : unsigned short(idl)) : HTML
+ReiniciaAdyacencia()
+MuestraMatrizAdyacencia() : HTML
+EscribeMatrizAdyacencia() : HTML
+EscribeAristaAdyacencia(entrada pos : short(idl)) : HTML
+InicializaMatrizAdyacencia()
+InsertaPesosEnMatriz()
+ConstruyeCaminoAristas() : HTML
+BorraCaminoAristas()
+PintaAristaSolucion(entrada arista_id : string(idl)) : HTML
+BorraAristaSolucion()
+dameAristasPorDistanciaMinimas() : HTML
+BorraSolucionesTotal() : window
+EsResultadoUsuarioCorrecto() : HTML
-damePesoCaminoUsuario() : long long(idl)


«interface»
window
+AsignaDirigido(entrada a : boolean(idl) = 0)
+AsignaNodoInicial()
+AsignaNodoFinal()
+MuestraNavegacion() : window
+MuestraTutorial() : window
+MuestraAyuda(entrada a : boolean(idl))
+CreaNodo(entrada event)
+MuestraMatrizAdyacencia()
+ReiniciaAdyacencia()
+ConstruyeCaminoMasCorto()
+ReiniciaCamino()
+ConstruyeCaminoAristas()
+BorraCaminoAristas()
+BorraSolucionesTotal()
«subsystem»
Window Ayuda
«utility»
HTML
+CreaDiv(entrada x : float(idl), entrada y : float(idl))
+Navegar()
-ActualizaURL()
#NormalizaURL(entrada URL : string(idl))
#CambiaNavegacion()
+MuestraNavegacion()
#Desenfoca(entrada target : any(idl))
#CambiaTutorial()
-MuestraTutorial()
-AsignaBotonCierre()
#InicializaAyudas()
+Posiciona(entrada x : long(idl), entrada y : long(idl)) : sequence(idl)
+Maximiza()
+PosicionaOrigenCanvas()
+MuestraPapeleraArde()
+OcultaPapeleraArde()
+coordenadasRaton(entrada event)
+DetectaExplorador()
+AddEvent(entrada obj : object(idl), entrada evt, entrada fn : string(idl), entrada bool : boolean(idl))
+RemoveEvent(entrada obj : object(idl), entrada evt, entrada fn : string(idl), entrada bool : boolean(idl))
+OcultaLugarInputVertice()
+MuestraLugarInputVertice(entrada txt : string(idl))
#OcultaInfoMsgDelay()
+MuestraInfoMsgDelay()
+OcultaBotones(entrada id : string(idl))
+MuestraInfoPesoTerminarArista(entrada arista_id : unsigned short, entrada txt : string(idl))
+OcultaInfoTerminarArista(entrada id : string(idl))
+MuestraInfoTerminarArista(entrada id : string(idl))
+OcultaInicioArista(entrada id : unsigned short(idl))
+OcultaBorrado(entrada id : string(idl))
+OcultaDiv(entrada id : string(idl))
+MuestraDiv(entrada id : string(idl))
+$(entrada id : string(idl))
+DesBloqueaPunto(entrada id : string(idl))
+BloqueaPunto(entrada id : string(idl))
+RatonSobreBotonBorrar(entrada trs : string(idl), entrada str : string(idl), entrada capa_h_info : string(idl))
+RatonFueraBotonBorrar(entrada trs : string(idl), entrada str : string(idl), entrada capa_h_info : string(idl))


«utility»
Flecha
#dameDesplazamiento(entrada xActual : long(idl), entrada yActual : long(idl), entrada globales.elComienzoX : long(idl), entrada globales.elComienzoY : long(idl), entrada arista : arista) : sequence(idl)
+ordenaPuntos(entrada x1 : long(idl), entrada x2 : long(idl), entrada y1 : long(idl), entrada y2 : long(idl)) : sequence(idl)
+ComparaAltoIzPuntos(entrada x1 : long(idl), entrada x2 : long(idl), entrada y1 : long(idl), entrada y2 : long(idl)) : arista
x:float(idl), y:float(idl), id:char(idl)
«implementation class»
vertice
-coord_x : float(idl)
-coord_y : float(idl)
-id : char(idl)
-pos : unsigned short(idl)
-bloqueado : boolean(idl) = false
-distancia : unsigned long long(idl) = Infinity
-visitado : boolean(idl) = false
-definitivo : boolean(idl) = false
-precedente : char(idl) = null
-aristacamino : unsigned short(idl) = null
+ReiniciaVertice(entrada Obj : vertice)
+ActualizaVerticesVisitados()
+dameVerticePos(entrada obj : vertice) : unsigned short(idl)
#dameVerticeX(entrada obj : vertice) : float(idl)
#dameVerticeY(entrada obj : vertice) : float(idl)
+dameVerticeXDIV(entrada obj : vertice) : float(idl)
+dameVerticeYDIV(entrada obj : vertice) : float(idl)
+dameVerticeId(entrada obj : vertice) : string(idl)
+dameVerticeBloqueo(entrada obj : vertice) : boolean(idl)
+dameVerticeDistancia(entrada obj : vertice) : long double(idl)
+EstaVerticeVisitado(entrada obj : vertice) : boolean(idl)
+dameVerticePrecedente(entrada obj : vertice) : vertice
+dameVerticeAristaCamino(entrada obj : vertice) : vertice
+damePosicionVerticeTop(entrada obj : vertice) : float(idl)
+damePosicionVerticeLeft(entrada obj : vertice) : float(idl)
#ActualizaVerticePos(entrada obj : vertice, entrada p : unsigned short(idl))
+ActualizaVerticeVisitado(entrada obj : vertice, entrada k : boolean(idl))
+ActualizaVerticeBloqueo(entrada obj : vertice, entrada k : boolean(idl))
+ActualizaVerticeDistancia(entrada obj : vertice, entrada k : long double(idl))
+ActualizaVerticePrecedente(entrada obj : vertice, entrada id : char(idl))
+ActualizaVerticeDefinitivo(entrada obj : vertice, entrada bool : boolean(idl))
+ActualizaVerticeAristaCamino(entrada obj : vertice, entrada id : short(idl))
+EsVerticeDefinitivo(entrada obj : vertice) : boolean(idl)
+ActualizaVerticeCamino(entrada v_act : vertice, entrada aris_id : unsigned short(idl), entrada id_precedente : char(idl), entrada valor : long double(idl), entrada bool : boolean(idl))
+EstaVerticeBloqueado(entrada id_vertice : string(idl)) : boolean(idl)
+vertice(entrada x : float(idl), entrada y : float(idl), entrada id : char(idl))
1
-Fin8
1
-Fin7
punto
-x : long long(idl)
-y : long long(idl)
+Punto(entrada x : long long(idl), entrada y : long long(idl))
+CoordX() : long long(idl)
+CoordY() : long long(idl)
«utility»
dibuja
+InicializaPosicionesDibujo()
-drawBorra(entrada x : unsigned short(idl), entrada y : unsigned short(idl))
-draw(entrada x : unsigned short(idl), entrada y : unsigned short(idl))
+drawResalta(entrada x : unsigned short(idl), entrada y : unsigned short(idl))
+drawArista(entrada x0 : unsigned short(idl), entrada y0 : unsigned short(idl), entrada x1 : unsigned short(idl), entrada y1 : unsigned short(idl))
+drawResaltaArista(entrada x0 : unsigned short(idl), entrada y0 : unsigned short(idl), entrada x1 : unsigned short(idl), entrada y1 : unsigned short(idl))
+drawAristaSolucion(entrada x0 : unsigned short(idl), entrada y0 : unsigned short(idl), entrada x1 : unsigned short(idl), entrada y1 : unsigned short(idl))
+drawAristaSolucionBorra(entrada x0 : unsigned short(idl), entrada y0 : unsigned short(idl), entrada x1 : unsigned short(idl), entrada y1 : unsigned short(idl))

3.3.3 Diseño de la Interfaz

El diseño se mantendrá uniforme con respecto al de las demás páginas Web y herramientas del departamento. Un ejemplo válido será la siguiente imagen, obtenida de los documentos que sobre teoría de grafos tienen alojados los servidores del departamento.

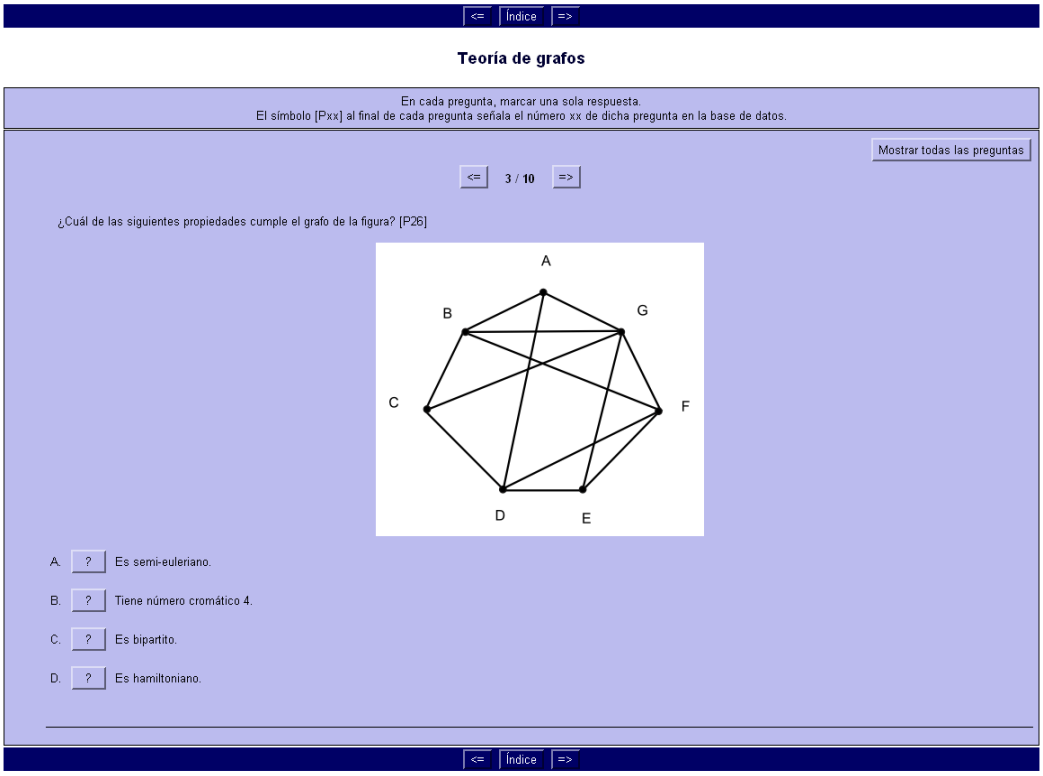


Figura 12 Diseño de la interfaz en la que se apoya el diseño de GrafoMin

Manteniendo los colores básicos y respetando los márgenes y tabulaciones de los diseños de los documentos Web del departamento, se incluirán dos zonas diferenciadas:

- **Zona gráfica**. Esta zona contendrá la etiqueta <CANVAS> la cual, permite dibujar dinámicamente. Se maquetará mediante capas <DIV>, de tal modo que se mantenga la integridad del grafo en su tamaño, forma y posición, aunque la ventana del navegador sea redimensionada antes, durante o después de la construcción del grafo. Para crear un vértice el usuario lo hará mediante doble click del botón izquierdo del ratón sobre esta zona. Cada vértice deberá estar conveniente y unívocamente identificado. Para construir una arista, se mostrará un botón de inicio de construcción sobre un vértice, con el efecto *OnMouseOver*. Una vez iniciada la construcción de la arista, tomado el vértice inicial, se bloquearán la finalización de la arista en los vértices que ya sean adyacentes con el vértice inicial. Se finalizará la construcción de la arista haciendo "*click*" con el botón izquierdo del ratón sobre un vértice permitido. Con el objetivo de que se puedan distinguir la dirección de la arista, si el grafo fuera dirigido, se mostrará mediante una flecha la dirección de dicha arista. Se podrá modificar el peso de una arista en cualquier momento (antes de solicitar la solución del problema a la herramienta). El peso de la arista no impedirá la correcta visión del grafo, ya que será desplazable dicho valor a lo largo de la arista, mediante el efecto "arrastrar y soltar".
- **Zona analítica**. Aquí se mostrarán los resultados analíticos del problema y es donde el usuario podrá construir su solución. Esta zona se utilizará para mostrar un <IFRAME> en el que el usuario podrá acceder a documentación de cualquier otra página de Internet, o bien a los vídeo-tutoriales de GrafoMin, con el fin del que el usuario pueda emplear la herramienta al tiempo que observa un vídeo-tutorial. Con el fin de que el usuario pueda construir el camino más corto entre dos vértices, se creará en esta zona una "*caja arista*" como representación de cada una de las aristas. La caja arista deberá mostrar los valores del vértice inicial, final y el peso de dicha arista. El usuario podrá construir su solución mediante el efecto arrastrar y soltar, tomando una caja arista y desplazándola a la zona de respuesta. Se deberá permitir la reordenación de cada caja arista que se encuentre en el área de solución del usuario.

El correcto funcionamiento deberá estar respaldado por la "usabilidad", de tal modo que, si el ratón señala o se posiciona sobre una arista, deberá resaltarse la línea arista en la zona gráfica y, en cada uno de los elementos representativos que el área analítica contenga dicho elemento, esto es, en las cajas arista.

La solución se mostrará tanto en formato gráfico sobre la zona gráfica, resaltando las líneas de arista, como en la zona analítica, resaltando las cajas arista en su orden correcto.

Se valorará, con texto indicativo, la solución propuesta por el usuario considerando que la solución puede no ser única, pues la herramienta mostrará mediante cajas arista solo una de las posibles soluciones válidas.

3.4 Implementación

3.4.1 Introducción

Los lenguajes de programación elegidos para la implementación de GrafoMin son HTML5, CSS3 y JavaScript.

HTML es un código estándar que es manejado por cualquier navegador y es independiente de la plataforma sobre la que se ejecute. HTML5 es la última versión del importante código HTML y todavía se encuentra en modo experimental, lo cual indica la misma W3C; aunque ya es usado por múltiples desarrolladores web por sus avances, mejoras y ventajas.

Se ha elegido CSS3 por ser la última versión de implementación mediante hojas de estilo en cascada, manteniendo así una separación entre la estructura de la herramienta y su presentación.

JavaScript es necesario para la implementación de las funcionalidades de la etiqueta <CANVAS> disponible en HTML5. Otra opción a <CANVAS> con JavaScript sería la utilización de SVG con ECMAScript o con JavaScript, pero se ha descartado esta opción por diversos inconvenientes:

- la interpretación por parte de la computadora de los gráficos, supone un alto coste de procesador cuando mostramos una gran cantidad de elementos gráficos, lo que implica una limitación en el número de vértices y aristas;
- maneja las imágenes de modo vectorial, mientras que la etiqueta <CANVAS> maneja píxeles, posicionando de modo inmediato el elemento;
- si bien SVG es una recomendación de la W3C desde 2001, los distintos navegadores han incluido esta especificación de modos diferentes –sirva como ejemplo que Google Chrome la incluye con WebKit desde su primera versión, mientras que Mozilla Firefox la incluye de modo nativo desde su versión 1.5–.

Por los inconvenientes anteriormente expuestos, se ha elegido <CANVAS> con JavaScript, si bien se ha tenido que renunciar al manejo de objetos gráficos como elementos del DOM del documento HTML. En todo caso, esto no es un inconveniente insalvable. Se puede controlar cada elemento gráfico guardando su posición y características, dentro de la instancia lógica asociada, y en nuestra implementación, son los valores de los atributos *CoordX* y *CoordY* dentro de cada instancia *vértice*.

Pasaremos a continuación a exponer los aspectos más importantes de la implementación de la herramienta. No pretendemos con ello hacer un manual de desarrollo extenso ya que se ha presentado el diagrama de clases en la sección correspondiente de este manual (ver el apartado 3.3.2.2 Diagrama de clases).

3.4.2 Métodos Globales

Como HTML5 es un estándar todavía en revisión, se han tenido que implementar varios métodos auxiliares pertenecientes a la clase *window* y *document* que sí están incluidos en

versiones anteriores al estándar, así como otras funciones auxiliares que facilitan el proceso de programación:

- *document.getElementsByClassName*: [grafo.js]
- *document.$(id)*: [index.html]
- *window.Cargando(function())*: [index.html]
- *String.prototype.trim*: [index.html]
- *Document.disableSelection(target)*: [index.html]

3.4.3 Ficheros

La herramienta GrafoMin está compuesta por diversos paquetes que engloban las clases (pseudoclases ya que JavaScript no es un lenguaje totalmente orientado a objetos como ya se ha citado anteriormente) descritas en la sección correspondiente (ver el apartado 3.3.2.2 Diagrama de clases), en dicho diagrama se describen los atributos con los tipos de datos utilizados. El fichero adjunto de creación de dicho diagrama, *Diagrama de Clases PFC.vsd*, contiene también el código de cada uno de los métodos. Todos los ficheros están codificados con la tabla de caracteres "*UTF-8*".

Como es habitual en las implementaciones de código orientado a objetos, cuando es necesario, el código se encuentra comentado y "*referenciado*" en lo relativo a restricciones funcionales y no funcionales, con nombres de variables explicativos y métodos cuya nomenclatura es explicativa en sí misma y "*camel sensitive*" [(1)].

A continuación describiremos solo los principales métodos que contenidos en cada fichero o paquete y algunos que por su importancia en la ejecución de la herramienta, consideramos importantes. Muchos de los métodos descritos poseen un inverso: método que ejecuta la acción opuesta en funcionalidad; dichos métodos no están incluidos en la presente descripción.

3.4.3.1 Index.html

Es el fichero principal y contiene la estructura gráfica de la herramienta. Desde este fichero se hacen las peticiones o llamadas al resto de ficheros auxiliares.

Contiene las variables y las siguientes funciones principales.

- *window.Cargando()*. Inicializa todos los valores.
- Métodos globales. Descritos en la sección 3.4.2
- Variables globales. La variable *globales* es de tipo objeto y guarda:
    - la posición del ratón cada vez que se ejecuta un método relativo a dicho dispositivo,
    - el estado de dicha acción –para los métodos arrastrar y soltar–, la resolución de pantalla del usuario,
    - constantes para la modificación de la interfaz –CSS– vía JavaScript,

[1] camel sensitive: nomenclatura utilizada en lenguajes como Java para los métodos; una frase explicativa de su función, sin espacios y con la primera letra de cada palabra en mayúscula; ejemplo: DameIdentificadorDelObjeto

- constantes para la correcta ubicación de los dibujos –dependen del tipo de navegador–,
- un atributo que indica que, si el estado de ejecución es de desarrollo (*globales.debug=true*), mostrará información para los desarrolladores.

- Métodos en etiquetas HTML para la definición del problema.

  - *AsignaDirigido(0)*: el usuario indica si el grafo es o no dirigido.
  - *AsignaNodoInicial()*: el usuario introduce el nombre del vértice inicial.
  - *AsignaNodoFinal()*: el usuario introduce el nombre del vértice final.
  - *MuestraAyuda(0)*: el usuario indica si necesita los globos de ayuda mientras trabaja con la herramienta.
  - *MuestraNavegacion()*: muestra u oculta el área <IFRAME> de navegación.
  - *MuestraTutorial()*: muestra u oculta los video tutoriales de aprendizaje de la herramienta.
  - *CreaNodo(event)*: crea un vértice cuando hace doble click sobre el área de la etiqueta <CANVAS> que contiene este método.

- Métodos en etiquetas HTML para la resolución del problema.

  - *MuestraMatrizAdyacencia()*: muestra la matriz de adyacencia.
  - *ReiniciaAdyacencia()*: borra la matriz de adyacencia y la reinicia.
  - *ConstruyeCaminoMasCorto()*: muestra la matriz analítica, ejecutando también *MuestraMatrizAdyacencia().*
  - *ReiniciaCamino()*: borra la matriz analítica.
  - *ConstruyeCaminoAristas()*: resuelve el problema totalmente ejecutando también *ConstruyeCaminoMasCorto().*
  - *BorraCaminoAristas()*: borra la resolución del problema.
  - *BorraSolucionesTotal()*: reinicia la aplicación.

3.4.3.2 Grafo.js

Este fichero contiene siguientes métodos principales:

- *ReiniciaGrafo()*: trivial.
- *InicializaGrafo()*: trivial.
- *AsignaDirigido(t)* : selecciona si el grafo es dirigido o no.

- *MuestraAyuda(t)* : muestra los globos de ayuda mientras se construye el grafo.
- *AsignaNodoInicial()*: trivial.
- *AsignaNodoFinal()*: trivial.
- *EsVerticeInicialValido()*: determina si el usuario ha escogido un vértice válido.
- *EsVerticeFinalValido()*: determina si el usuario ha escogido un vértice válido.
- *MuestraFlechas()*: si el grafo es dirigido, muestra la dirección de las aristas.
- *IncrementaIndiceVertices()*: automatiza la creación de indicadores para los vértices; valor inicial: A; [A..Za..z].
- *Incrementa (tipo)*: incrementa los indentificadores para nuevos vértices o aristas.
- *Vertice(x,y,id)* : constructor.
- *Arista(v1,id)* : constructor.
- *CreaNodo(event)* : crea un vértice.
- *BorraPunto(event)* : borra un vértice.
- *BloqueaVertices (vertice_id)* : bloquea los vertices con los que no se puede finalizar la arista iniciada.
- *IniciarArista(event)*: inicia la creación de una arista.
- *TerminarArista(punto_id)* : finaliza la creación de una arista.
- *ResaltaArista(arista_id)* : resalta la arista gráficamente.
- *RatonSobrePeso(event)* : actúa con el evento OnMouseOver sobre la caja –capa- que contiene el peso de la arista.
- *dameVertice(id)* : localiza y devuelve un vértice por su identificador.
- *dameArista(id)* : localiza y devuelve una arista por su identificador.
- *dameAdyacentes(id)* : devuelve todos los vértices adyacentes que no estan cerrados (definitivos) al identificado por su id de entrada; válido para grafos con lazo.
- *dameVerticeDistanciaMayor()*: devuelve el vértice con mayor distancia acumulada (dicha distancia no puede ser infinito).
- *dameIdVerticeMasCercano(id_actual)* : retorna el id del vértice con menor distancia acumulada y el id de arista que los une.
- *EsAristaDeCamino(arista_id)*: verifica si la arista pertenece al camino más corto.
- *dameAristaIniFin(id1,id2)* : devuelve la arista que une ambos vértices.

3.4.3.3 Html.js

Este fichero contiene una gran cantidad de funciones auxiliares para la modificación del interfaz y es utilizado fundamentalmente por los métodos residentes en el fichero *grafo.js*.

Sus principales métodos son:

- *CreaDiv(x,y)*: crea la capa para los valores y acciones sobre el nodo-vértice-punto.
- *BloqueaBorrado(id)*: oculta el botón de borrado para un vértice si contiene aristas dicho vértice.
- *BloqueaPunto(id)*: impide acciones sobre un vértice –borrado o finalización de arista sobre dicho vértice–.
- *MuestraInfoTerminarArista(id)*: trivial.
- *MuestraInfoPesoTerminarArista(arista_id,txt)*: si el usuario tiene activa la solicitud de ayuda mediante globos, muestra la información de cómo puede finalizar la arista ya iniciada.
- *MuestraInfoMsgDelay(txt,x,y)*: muestra un mensaje de error si el usuario no ha indicado los vértices inicial o final.
- *MarcaVerticeInicial()*: resalta el vértice indicado por el usuario como inicial.
- *MarcaVerticeFinal()*: ídem anterior con el vértice final.
- *BloqueaSolucionUsuario()*: no permite modificaciones en el resultado del usuario; se utiliza cuando el usuario ha solicitado la resolución del problema.
- *Navegar()*: solicita una dirección web que será mostrada en el <IFRAME> de consulta.
- *MuestraNavegacion()*: muestra u oculta el <IFRAME> de ayuda para navegar por Internet.
- *Desenfoca(target)*: no ejecuta nada, pero retira el foco del ratón del lugar en el que esta: ejecutando así la acción anterior; necesario en algunos navegadores.
- *MuestraTutorial()*: muestra u oculta el <IFRAME> de ayuda para ver los vídeo tutoriales de aprendizaje de la herramienta.
- *InicializaAyudas()*: método auxiliar; no puede ser mostrado el navegador y los vídeo tutoriales a un tiempo.
- *Posiciona(x,y)*: recibe un punto en coordenadas absolutas y lo posiciona en las relativas a la etiqueta <CANVAS>.
- *Maximiza()*: maximiza la ventana de la herramienta.
- *PosicionaOrigenCanvas()*: método de inicialización que posiciona el origen de coordenadas para las capas en el origen de coordenadas de la etiqueta <CANVAS>.
- *AddEvent(obj,evt,fn,bool)*: método auxiliar para ejecutar en cualquier navegador la definición o creación de un evento.

3.4.3.4 Flecha.js

Fichero que contiene los métodos relativos a las etiquetas <CANVAS> que contienen el peso de la arista y la flecha indicadora de dirección de la arista si el grafo es dirigido. También se incluyen los métodos funcionales para el desplazamiento del peso a lo largo de la arista.

Los métodos principales se indican a continuación.

- *PonFlecha(pos_x,pos_y,m,arista)* : llamado por el método *FinalArista* del fichero *grafo.js*, para crear la caja que contiene el peso y la flecha indicativa de dirección de la arista que se muestra sobre la línea de la arista; admite desplazamiento arrastrar y soltar –se llamaría a *comienzoMovimiento(event)*–.
- *CrearDiv(id,padre,pos_x,pos_y,arista,peso)* : método auxiliar llamado por el anterior; crea la capa HTML.
- *CrearInputPeso(id)* : método que crea el *input* de HTML para la toma de datos del peso de la arista –se llamaría a *ModificaPesoArista(event)*–, así como para mostrar su valor sobre la línea arista; es llamado por el método *PonFlecha*.
- *CreaDivInfoArista(arista_id,x,y)* : crea la capa informativa de cómo modificar el valor del peso de la arista y, cómo desplazar la capa que creada por *PonFlecha*.
- *DibujarFlecha(id,canvas_id,angulo)* : crea un objeto <CANVAS> sobre el que se dibujará la flecha de dirección de la arista.
- *CrearCanvas(id,padre)* : crea el objeto <CANVAS>.
- *comienzoMovimiento(event)* : método que inicia el desplazamiento de la caja creada por *PonFlecha*.
- *enMovimiento(event)* : método que inicia renderiza[1] el desplazamiento de la caja creada por *PonFlecha*.
- *finMovimiento(event)* : método que inicia el desplazamiento de la caja creada por *PonFlecha*.
- *dameDesplazamiento(xActual,yActual,arista)*: método auxiliar llamado por el método *enMovimiento*; *xActual*: nueva posición del ratón ("*left*": coordenada x); *yActual*: nueva posición del ratón ("*top*": coordenada y); *arista*: la recta sobre la que se desplaza la capa div. Cambiamos de signo a los valores del eje y ("*top*") para que sea igual que un eje de coordenadas cartesiano; usamos la ecuación de la recta punto-pendiente:

  $$Y - y1 = m(X - x1).$$

- *ordenaPuntos(x1,y1,x2,y2)* : método auxiliar de *dameDesplazamiento* para determinar la posición de los vértices de la arista; el primer punto es (x1,y1); el segundo punto es (x2,y2); el método devuelve el punto que está más arriba; si ambos están a la misma altura, devuelve el punto que está más a la izda. Es un método auxiliar.
- *ComparaAltoIzPuntos(x1,y1,x2,y2)*: x1, x2 son positivos; y1,y2 son negativos; es un método auxiliar llamado por *OrdenaPuntos*.

[1] Rederizar: inglés técnico; proceso computacional de creación de imágenes mediante modelos.

3.4.3.5 Dibuja.js

Este fichero contiene los métodos que dibujan sobre una etiqueta <CANVAS>. Son métodos para mostrar gráficamente el grafo. Principalmente serán llamados por métodos de los ficheros *Flecha.js* y *grafo.js*, para crear el dibujo de la flecha de dirección de la arista, en el caso del primero, y para dibujar los vértices y las líneas de arista, en el segundo. También contiene el método para la creación del objeto punto, utilizado para instanciar los píxeles que ubican los vértices, y la creación del objeto recta, para las aristas.

- *InicializaPosicionesDibujo()*: dependiendo de la resolución de la pantalla del usuario, posicionará el dibujo en un lugar o en otro.
- *draw(x,y)*: dibuja un vértice.
- *drawBorra(x,y)*: borra un vértice dibujado otro de igual color que le fondo del tablero (tablero: capa donde está la etiqueta <CANVAS> principal).
- *drawResalta(x,y)*: resalta un vértice.
- *drawArista(x0,y0,x1,y1)*: dibuja una arista.
- *drawResaltaArista(x0,y0,x1,y1)*: resalta una arista.
- *drawAristaSolucion(x0,y0,x1,y1)*: dibuja una arista solución del camino más corto.
- *drawAristaSolucionBorra(x0,y0,x1,y1)*: borra una arista del camino más corto.

3.4.3.6 Caminos.js

Este fichero contiene los métodos para que el usuario pueda crear, por medio de las llamadas "*cajas arista*", la solución del problema. GrafoMin crea una "*caja arista*" por cada arista y la ubica en la columna llamada "ARISTAS" del área "*Construcción del camino más corto*". Este fichero contiene el método para crear dichas "*cajas arista*" que son de dos tipos: las creadas por la herramienta, y las "*cajas clon*" creadas por el usuario –referencian a una arista–. La herramienta no permite que el usuario cree varias "*cajas clon*" en la columna "CAMINO", para evitar errores triviales. La citada columna contiene la solución propuesta por el usuario. Dicha solución no tiene porqué ser la única. GrafoMin evaluará su solución e indicará si es correcta. Si antes de solicitar la evaluación, el usuario desea reordenar sus "*cajas clon*", podrá hacerlo con el método arrastrar y soltar, y con el mismo método podrá eliminar una de sus "*cajas clon*" enviándola a la papelera.

Citamos seguidamente los métodos contenidos en el fichero.

- *InitCamino()*: asigna los eventos caminos, soltar y borrar a la zona de solución del usuario llamada "CAMINOS" y a la papelera.

- *empezar(event,id)* : inicia el movimiento de una "*caja clon*" sobre la zona "CAMINOS" para crear una copia de la "caja arista" ubicada en la zona "ARISTAS" o bien, inicia el movimiento de una "caja clon" para ser reubicada –reordenada– en la zona solución del usuario llamada "CAMINOS".
- *soltar(e)* : cuando el usuario suelta sobre su zona de solución "CAMINOS", la "*caja arista*" ubicada en la zona "ARISTAS", se creará una copia que llamamos "*caja clon*" y, se bloqueará la "*caja arista*" –no pude volver a copiarse a no ser que sea enviada la "*caja clon*" a la papelera–.
- *desplazar(e)* : efecto del movimiento de una "*caja clon*" sobre otra.
- *recolocar(thisid,event)* : efecto del movimiento de una "caja clon" fuera de otra, pero dentro del área "CAMINOS" –tras "desplazar" sin soltar–
- *dameCajaArista(tipo,id)* : construye la caja que contiene los datos de la arista: para la sección "ARISTAS" y la seccion "CAMINOS" (tipo=arista;tipo=camino) .
- *CreaCajaArista(id)* : inserta una "*caja clon*" de una "caja arista" de la zona "ARISTAS" en la zona "CAMINOS".
- *AsignaEventos(aux,id,tipo)* : asigna los eventos correspondientes a arrastrar y soltar a una "*caja arista*" o a una "*caja clon*", dependiendo del valor pasado en el parámetro "tipo".
- *BloqueaDragArista(id)* : impide crear una "*caja clon*" de una "*caja arista*".
- *borrar(event)* : elimina una "*caja clon*" y, rehabilita la "*caja arista*" ubicada en la zona "ARISTAS" de quien era referencia.

3.4.3.7 Midistra.js

Todo lo contenido en este fichero está orientado al área de solución de la herramienta llamada "*construcción del camino más corto*".

Este fichero contiene la lógica de la resolución del camino más corto aplicando el algoritmo de Dijkstra asó como los métodos de inicialización, reinicialización y creación de las matrices (analítica y de adyacencia) del camino más corto.

Es importante señalar que nuestra "*matriz de adyacencia*" es particular. Por regla general la matriz de adyacencia suelen estar compuestas por ceros y unos –cero: no hay arista, uno: hay arista. Sin embargo, en el caso de GrafoMin el símbolo "-" indicará que no hay arista mientras que otro valor indicará el peso de la arista.

- *ReiniciaCamino()*: reinicia todos los valores de la herramienta; llama a los métodos *ReiniciaAdyacencia()*, *ReiniciaAnalitica()* y *ReiniciaGrafo()*.
- *ConstruyeCaminoMasCorto()*: verifica si se cumplen las condiciones para el algoritmo de Dijkstra, emitiendo los mensajes de error correspondientes; en caso de que se cumplan las condiciones, llama al método *Camino()*,

- *Camino()*: ejecuta el algoritmo de Dijkstra propiamente dicho; calcula el camino mínimo desde el nodo inicial hasta todos los vértices por los que no se haya pasado; se inicializa [0,*id_propio_vértice*]; por ejemplo para el vértice identificado con el valor 3 sería: [0,3]; *distancia* es una variable del vértice que guarda el peso desde el vértice origen hasta dicho vértice; la distancia al vértice inicio se toma 0.
- *ReiniciaAnalitica()*: trivial.
- *EscribeMatrizAnalitica()*: escribe y muestra la matriz analítica.
- *ReiniciaAdyacencia()*: trivial.
- *MuestraMatrizAdyacencia()*: muestra la matriz de adyacencia creada por el método descrito a continuación.
- *EscribeMatrizAdyacencia()*: calcula y escribe en código HTML la matriz de adyacencia.
- *InicializaMatrizAdyacencia()*: método inicializador que construye la matriz de adyacencia valorando todos los pesos con el valor infinito.
- *InsertaPesosEnMatriz()*: valora los pesos de las aristas existentes toda vez que ha sido ejecutado el método anterior.
- *ConstruyeCaminoAristas()*: calcula la solución del problema planteado valorando la solución del usuario y muestra los datos. Método principal.
- *PintaAristaSolucion(id_arista)*: marca resaltando, en el área de resolución gráfica, una arista que es solución al problema.
- *dameAristasPorDistanciaMinimas()*: devuelve las aristas del camino mínimo: debe estar creada la matriz analítica.
- *BorraSolucionesTotal()*: método principal de reiniciado de la Herramienta; recarga la página.
- *EsResultadoUsuarioCorrecto()*: método que calcula y muestra el resultado, de la solución propuesta por el usuario.

3.4.3.8 Navegador.html

Este fichero es el presentado cuando es llamado el método *MuestraNavegacion()* contenido en el fichero *html.js*; esto sucede cuando el usuario solicita que se muestre el área de navegación contenido en la etiqueta <IFRAME>, con el fin de poder consultar información en Internet, pudiendo ver al mismo tiempo el grafo que ha construido o que está construyendo.

El contenido de dicho fichero no es más que una explicación de cómo puede navegar el usuario. Esto es, dónde debe de escribir la dirección del documento a buscar.

Es necesario que el usuario tenga conexión a Internet.

3.4.3.9 Ayuda.html

Este fichero contiene el índice de los vídeo tutoriales de aprendizaje de la herramienta. El usuario podrá acceder a ellos y verlos en el área de resultados analíticos, de tal modo que no interfiera en la construcción de su grafo.

De modo similar al fichero navegador.html, este fichero es llamado por el método *MuestraTutorial()* contenido en el fichero *html.js*.

Si el usuario no tiene conexión a Internet, pero se ha bajado los archivos tipo flash de los vídeo tutoriales, podrá acceder a ellos sin problemas.

Es necesario contar con el complemento Adobe flash player instalado en el navegador para ver los vídeo tutoriales, tal y como se ha indicado en los requisitos funcionales en esta memoria.

3.4.4 Pruebas

3.4.4.1 Introducción

El objetivo principal es que la herramienta cumpla con las especificaciones requeridas, además de eliminar los posibles errores. El objetivo es doble: verificar si la herramienta hace lo que debe hacer y no hace lo que no debe hacer. Por definición y por experiencia, todas las herramientas de software son imperfectas. Siempre hay uno o más errores no contemplados o no experimentados. La madurez del programa irá decrementando este número, pero podemos elaborar un producto de buena calidad si establecemos un plan de pruebas iniciado en el proceso de desarrollo del software para controlar, corregir y prever errores. Para ello, contamos con las herramientas utilizadas en dicho proceso de desarrollo y con las personas que han efectuado el mismo. No es óptimo que sean las mismas personas que han desarrollado el software las que ejecuten el plan de pruebas, pero por problemas de coste, nos hemos visto obligados a ello.

Una de las herramientas empleadas para la corrección de errores es Firebug. Existe una versión para Mozilla Firefox y también para Google Chrome pero la primera funciona mejor y es más clara. Firebug es una extensión creada y diseñada especialmente para desarrolladores y programadores Web; un paquete de utilidades con el que se puede analizar (revisar la velocidad de carga, la estructura DOM), editar, monitorizar y depurar el código fuente (CSS, HTML,XML y JavaScript) de una página web de manera instantánea (en tiempo de ejecución). La versión utilizada es la 1.9.1, específica para Mozilla Firefox versión 11.

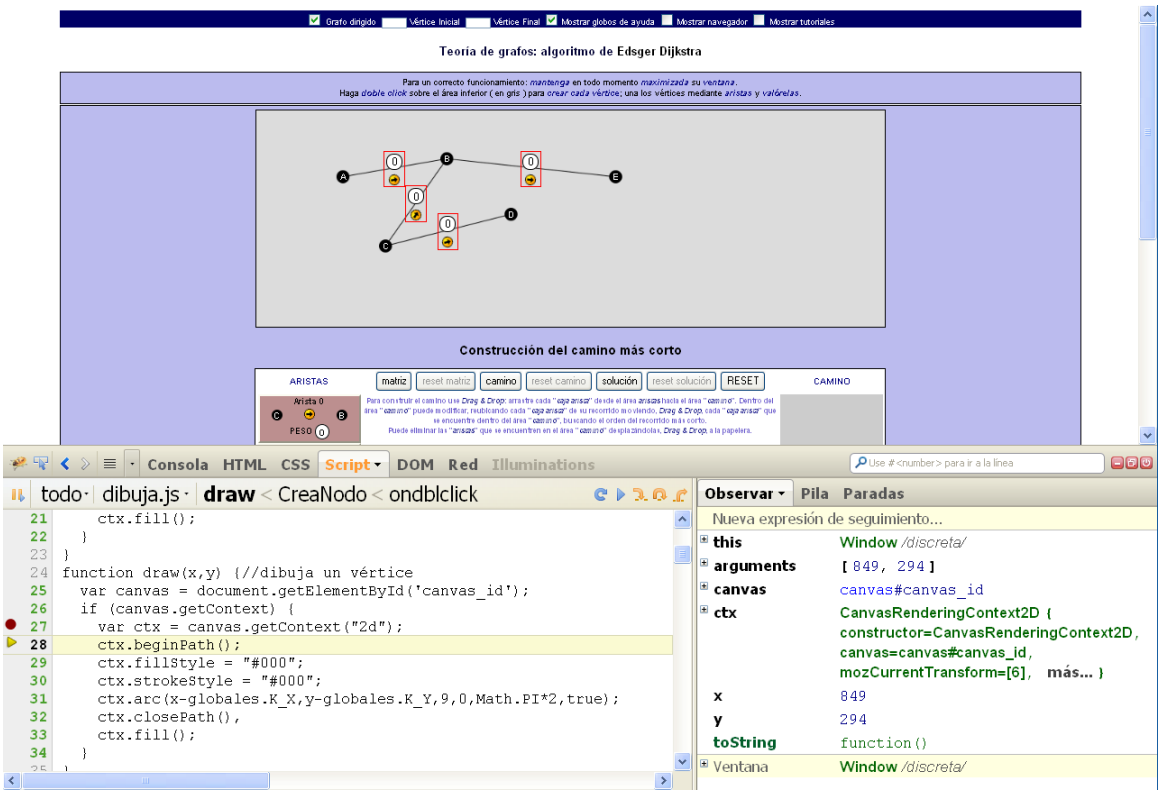


Figura 13 Seguimiento paso a paso de un método con Firebug

Para poder establecer una metodología de trabajo correcta para la detección y búsqueda de errores, se ha realizado como hemos dicho, el plan de pruebas. Éste explicita el alcance, enfoque, recursos requeridos, calendario, responsables y manejo de riesgos del proceso de pruebas. Incluye identificación del plan de acción y fecha del plan, el alcance que es el tipo de prueba y las propiedades o elementos del software a ser probado, los ítems a probar que son los elementos a probar y las condiciones mínimas que debe cumplir para comenzar a aplicarle el plan.

3.4.4.2 Planes de Pruebas

3.4.4.2.1 Identificadores

Se han identificado los planes de pruebas para la fase de desarrollo y puesta en marcha con los siguientes nombres:

- TP_Requisitos_1: plan de verificación de satisfacción de requisitos.
- TP_Interfaz_1: plan de verificación de las funcionalidades de la interfaz.

3.4.4.2.2 Alcance

Las pruebas se han llevado a cabo efectuando peticiones HTTP a los discos duros de modo local en los propios equipos clientes, y desde los clientes a los equipos servidor.

El hardware y entorno requerido para la verificación de ambos planes, TP_Requisitos_1 e TP_Interfaz_1, es el siguiente:

- Equipo Cliente_1
  - AMD Sempron a 2 GHz
  - 2,75 GB RAM tipo SDRAM
  - Resoluciones de pantalla
    - 1024 x 768 píxeles
    - 1280 x 1024 píxeles
    - 800 x 600 píxeles
  - Windows XP, Services Pack 3
  - Google Chrome V.18 + Firebug
  - Mozilla Firefox V.11 + Firebug

- Equipo Cliente_2
  - Intel Pentium
  - 2 GB memoria RAM tipo RIMM
  - 2,4 GHz
  - Resoluciones de pantalla
    - 1024 x 768 píxeles
    - 1280 x 1024 píxeles
    - 800 x 600 píxeles
  - Windows XP, Services Pack 3
  - Mozilla Firefox V.11 + Firebug

- Equipo Servidor_1
  - Xeon Intel E3 4 Cores
  - 12 GB de RAM
  - Ubuntu 10.04 LTS (32 bit)
  - Apache 2.2.22 estable desde el 31 de enero de 2012

3.4.4.2.3 Técnicas de prueba

Se utiliza la siguiente nomenclatura para las pruebas:

1. G{VÉRTCES:A,B;ARISTAS:1[A(peso:3)B]} Origen A; Fin: F. Queremos indicar un grafo compuesto por los vértices A y B, con una arista de peso 3 desde el vértice A al vértice B; se busca el camino más corto entre el vértice inicial A y el vértice final B.
2. Si en lugar de un identificador de vértice o de peso, se muestra el carácter "-" indica vacío, ausente o no definido.

3.4.4.2.4 Técnicas de caja negra o funcionales

Se ha verificado y probado las funcionalidades

- grafo definido y grafo no definido: comprobación de su entrada y de su título,
- solicitud de ayuda y de navegación: comprobación de su entrada y de título;
- solicitud de globos informativos y ausencia de ellos: comprobación de su entrada y de y de texto;
- Asignación de vértices inicial y final
    - asignaciones válida e inválida vértice inicial,
    - asignaciones válida e inválida vértice final,
    - coloración –resaltado- del vértice inicial
    - coloración –resaltado- del vértice final
- solicitud de creación de aristas:
    - de un vértice a otro (ambos no bloqueados),
    - de un vértice a otro (ambos bloqueados),
    - de un vértice no bloqueado a otro vértice bloqueado,
    - fuera de vértice,
- solicitud de creación de vértices:
    - un vértice,
    - 50 vértices,
    - fuera del área <CANVAS>
- solicitud de matriz de adyacencia:
    - número de vértice par,
    - número de vértice impar,
    - vértices sin aristas y con aristas,
    - grafo dirigido y no dirigido,
    - reseteado y regeneración,
- solicitud de matriz analítica:
    - número de vértice par,
    - número de vértices impar,
    - número vértice inicial sin aristas,
    - vértice final sin aristas y con aristas,
    - vértice final no accesible,
    - reiniciado y regeneración,
- solicitud de resolución del problema;
    - para 3, 4, 5, 6 y 7 vértices;
    - con aristas y sin aristas,
    - con vértices inicial y final,
    - sin vértices inicial ni final,
    - con solución del usuario,
    - sin solución del usuario,
    - sin posible solución,
    - con varias soluciones,
    - reiniciado y regenerado
- objetos con la funcionalidad "arrastrar y soltar"; se ha verificado arrastrando el objeto por los vértices, por su imagen y por la zona sin texto:
    - una "*caja arista*" para crear un una caja clon,
    - una "*caja clon*" sobre otra,
    - una "*caja arista*" sobre cualquier área,
    - una "*caja clon*" sobre cualquiera área,
    - una "*caja clon*" sobre la papelera,
    - una "*caja clon*" fuera de la ventana,

    - una "*caja arista*" fuera de la ventana.
    - caja peso:
        - resaltado y desresaltado de la línea arista, de las "*cajas arista"* y "*cajas clon*" y resaltado de los vértices,
        - modificación del valor del peso con letras, símbolos y números positivos y negativos, (hay que verificar que no admita negativos y muestra al usuario un mensaje de error).
        - arrastrar y soltar la caja peso en aristas de diferentes pendientes, tanto positivas como negativas,
- botones:
    - matriz,
    - reiniciar matriz,
    - camino,
    - reiniciar camino,
    - solución,
    - reiniciar solución,
    - reiniciar

Tipos de problemas planteables para cada una de las pruebas:

- G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C], 4[C(2)D], 5[D(7)E], 6[D(2)F], 7[E(1)F],} Origen A; Fin: F
- G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C], 4[C(2)D]} Origen -; Fin: F
- G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(-)C], 4[C(2)D], 5[D(7)E], 6[D(2)F], 7[E(1)F],} Origen A; Fin: -
- G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C], 4[C(2)D]} Origen -; Fin: -
- G{VÉRTCES:A,B,C;ARISTAS:1[A(peso:3)B], 2[A(-)C], 3[B(5)C]} Origen A; Fin: C

3.4.4.2.5 Técnicas de caja blanca o estructurales

Se ha verificado y probado todos los caminos de flujo de datos con las siguientes pruebas:

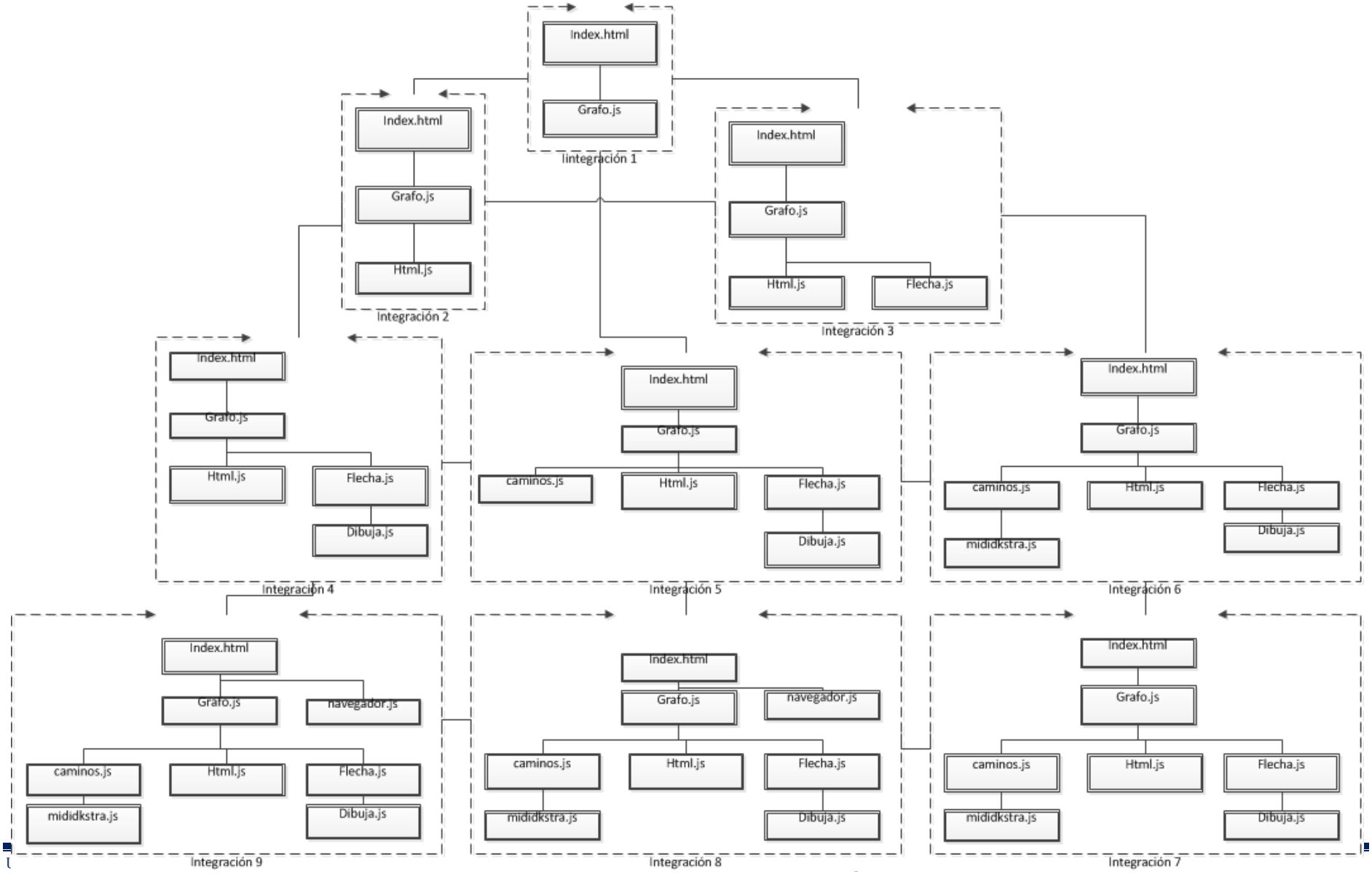
- G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C], 4[C(2)D], 5[D(7)E], 6[D(2)F], 7[E(1)F],} Origen A; Fin: F
- G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(-)C], 3[B(5)C], 4[C(2)D]} Origen -; Fin: F
- grafo G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C], 4[C(2)D], 5[D(7)E], 6[D(2)F], 7[E(1)F],} Origen A; Fin: -
- grafo G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C], 4[C(2)D]} Origen -; Fin: -
- grafo G{VÉRTCES:A,B,C;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C]} Origen A; Fin: C
- grafo G{VÉRTCES:A,B,C,D,E,F;ARISTAS:1[A(peso:3)B], 2[A(3)C], 3[B(5)C], 4[C(2)D]} Origen A; Fin: C

3.4.4.2.6 Pruebas unitarias y de integración

Durante el desarrollo se han ido realizando pruebas incrementalmente a la creación de los diferentes paquetes o ficheros. Como se ha descrito en las secciones de desarrollo, cada paquete está codificado en un fichero diferente, uniendo en un paquete varias clases. Las clases, como

la correcta programación orientada a objetos indica, poseen métodos de tipo "*set*"[(1)] y "*get*"[(2)] de tal modo que no se vean alterados los datos de una instancia, si no es desde dentro de la propia clase. Tras cada implementación de código se ha realizado las siguientes pruebas de integración como se describe en el gráfico de la página siguiente.

[(1)] set: inglés técnico, programación orientada a objetos; prefijo de los métodos que asignan valores a una variable.
[(2)] get: inglés técnico, programación orientada a objetos; prefijo de los métodos que devuelven el valor de una variable u objeto, o el objeto propiamente dicho.


Index.html
Grafo.js
Iintegración 1
Index.html
Grafo.js
Html.js
Integración 2
Index.html
Grafo.js
Html.js
Flecha.js
Integración 3
Index.html
Grafo.js
Html.js
Flecha.js
Dibuja.js
Integración 4
Index.html
Grafo.js
caminos.js
Html.js
Flecha.js
Dibuja.js
Integración 5
Index.html
Grafo.js
caminos.js
Html.js
Flecha.js
Dibuja.js
mididkstra.js
Integración 6
Index.html
Grafo.js
navegador.js
caminos.js
Html.js
Flecha.js
midikstra.js
Dibuja.js
Integración 9
Index.html
Grafo.js
navegador.js
caminos.js
Html.js
Flecha.js
midikstra.js
Dibuja.js
Integración 8
Index.html
Grafo.js
caminos.js
Html.js
Flecha.js
midikstra.js
Dibuja.js
Integración 7

3.4.5 Documentación

3.4.5.1 Introducción

La documentación relativa a la herramienta GrafoMin se presenta en diversos formatos y está dirigida a diferentes usuarios:

- Jefe de proyecto
- Programador
- Usuario

3.4.5.2 Carpeta profesional

La propia memoria actúa como carpeta profesional y está dirigida al jefe de proyecto. En la memoria está descritos los requisitos, la metodología de desarrollo y de pruebas, y el estudio de costes en tiempo de desarrollo y económicos.

3.4.5.3 Carpeta técnica

Los capítulos segundo, tercero de la presente memoria y los ficheros adjuntos con los diagramas de clases, donde los programadores podrán encontrar las bases de la codificación y parte de la implementación, configuran esta carpeta. Los archivos relativos son ficheros ejecutables en la herramienta Microsoft Visio 2010 y son:

- *Diagrama de Clases PFC.vsd*
- *Casos de Uso PFC.vsd*
- *Prueba incremental de integración PFC.vsd*

3.4.5.4 Carpeta económica

Se conforma con el capítulo 4 de la presente memoria y con los ficheros adjuntos ejecutables en la herramienta Microsoft Project:

- *Grafomin.mpp*

3.4.5.5 Carpeta de usuario final

Esta carpeta está conformada con la documentación necesaria para la utilización de la herramienta GrafoMin que está presentada en los ficheros ejecutables en cualquier navegador que cuente con la extensión Adobe flash player:

- Aprendizaje rápido

- Introducción: descripción de botones y uso de GrafoMin
- Creación del grafo
    - Construcción | Borrado de vértices
    - Asignación de grafo Dirigido | No dirigido
    - Construcción de Aristas
    - Desplazamiento y valoración de pesos
    - Asignación de vértice inicio | final
- Construcción del Camino más corto
    - Selección de Aristas
    - Reordenación de aristas seleccionadas
    - Borrado de aristas asignadas
- Construcción | Borrado de matrices
    - Creación | borrado de la matriz de adyacencia
    - Creación | borrado de la matriz Analítica
- Obtención de la solución y borrado
    - Obtener la solución
    - Tipos de borrado de la solución

3.4.6 Mantenimiento

3.4.6.1 Prevención

Será necesario designar una persona responsable de la coordinación de las tareas de mantenimiento. El responsable tomaría el plan de mantenimiento del presente proyecto o bien, utilizando técnicas de reingeniería, diseñaría un nuevo plan. Con el objetivo de no hacer dos veces la misma tarea y con el conocimiento fehaciente de que no es tarea del equipo responsable del proyecto la elaboración de tal plan, pues así ha sido acordado con el cliente, en esta sección del capítulo describiremos las líneas deseables de actuación de dicho plan, sin explicitarlo.

Tras la designación de un responsable de mantenimiento, el siguiente paso será asignar un método para capturar los errores detectados por los propios usuarios. Lo más sencillo y fácil será crear un enlace dentro de la herramienta a un formulario que recoja las incidencias. Se puede añadir una dirección de correo electrónico para indicar a los usuarios otro método de contacto donde puedan enviar sus solicitudes de corrección de errores, ampliación de funcionalidades, o bien cambios en dichas funcionalidades.

El responsable de mantenimiento deberá designar al programador o programadores para la resolución de errores y para implementación de nuevas funcionalidades, si se solicitaran por parte del cliente. Dicho equipo deberá utilizar la documentación correspondiente, descrita en la sección anterior, para el conocimiento de GrafoMin.

Cualquier cambio introducido en la herramienta deberá ser documentado y se deberá añadir dicha documentación al presente proyecto.

Todo cambio introducido en el proyecto deberá contar con una evaluación de errores. Si los cambios producidos son debidos a tareas de corrección de errores, se deberá documentar dicha corrección añadiendo las modificaciones y las pruebas de integración de software pertinentes. Si los cambios producidos son de carácter de ampliación de funcionalidades, se deberá efectuar una prueba de regresión.

Capítulo 4

Planificación y presupuesto

4.1 Planificación

4.1.1 Introducción

Este apartado engloba la organización, planificación y estimación del proyecto. Nos referiremos en esta sección a diversas tareas, con la nomenclatura especificada en el diagrama de Gantt adjunto. Para ello utilizamos la herramienta Microsoft Project 2010.

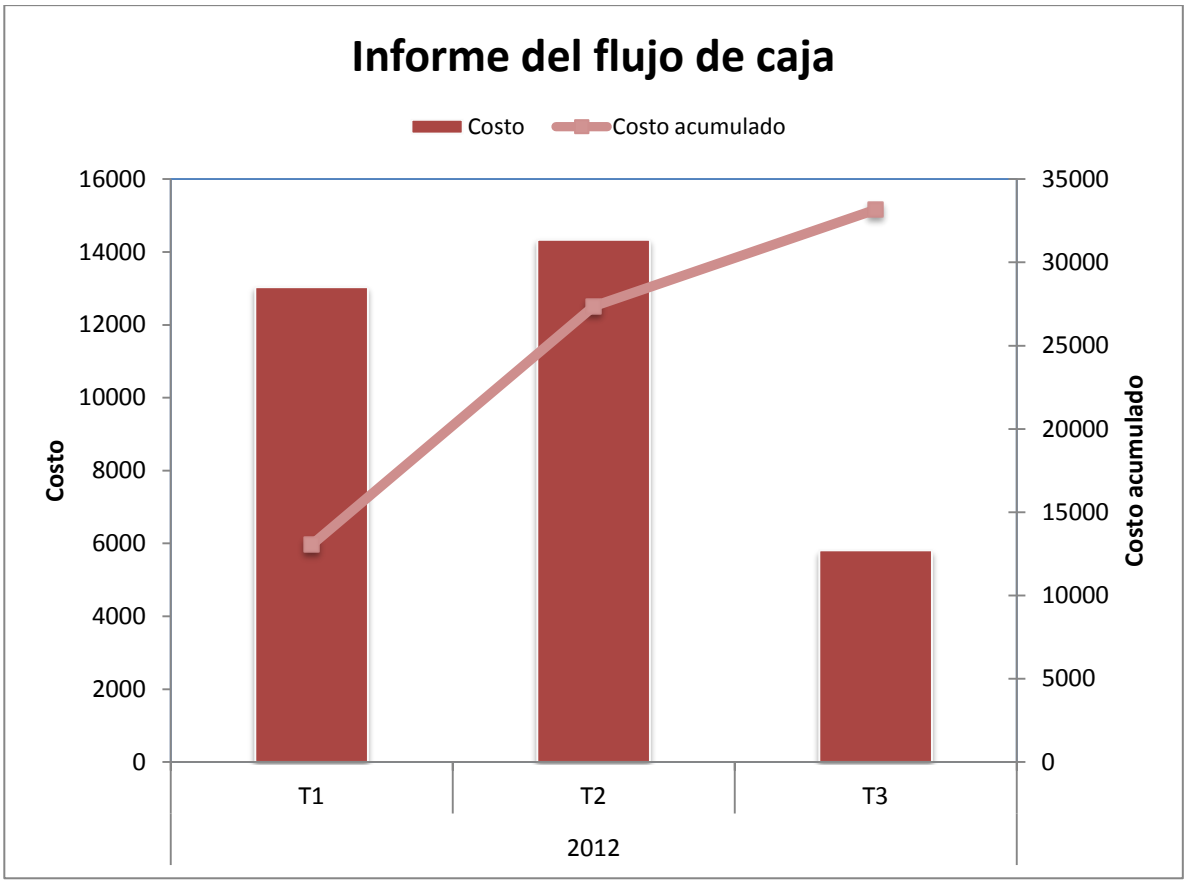
Se reflejan en el citado diagrama de Gantt todas las tareas a realizar para la elaboración y ejecución del presente proyecto. Será necesario plantear objetivos parciales, a fin de controlar los tiempos de demora o elasticidad, así como las propias fases de la ejecución del mismo, reflejadas como hitos o tareas finalizadas necesariamente antes de la ejecución de las subsiguientes fases. También se describen los recursos necesarios en cuanto a personal y los costes de los mismos, de tal modo que podremos obtener la valoración del coste total del proyecto.

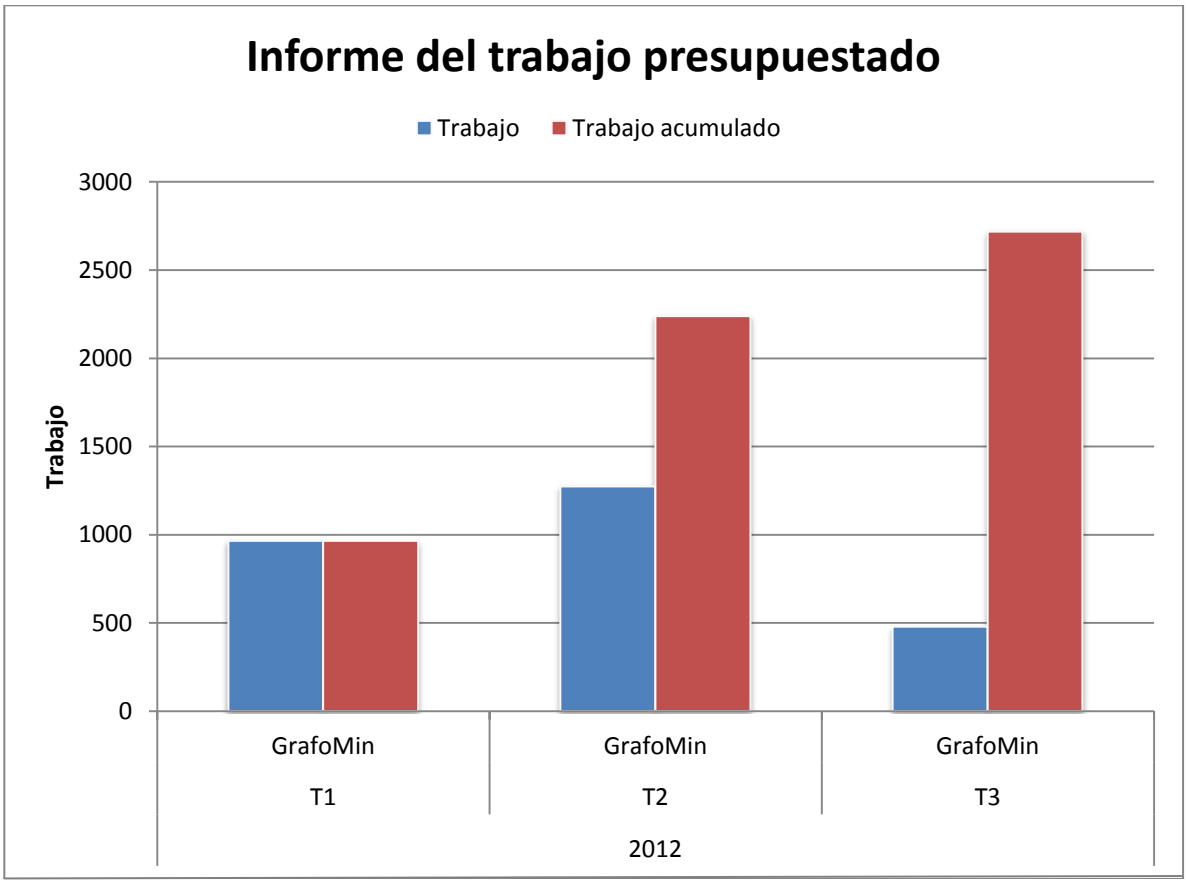
4.1.2 Estimación

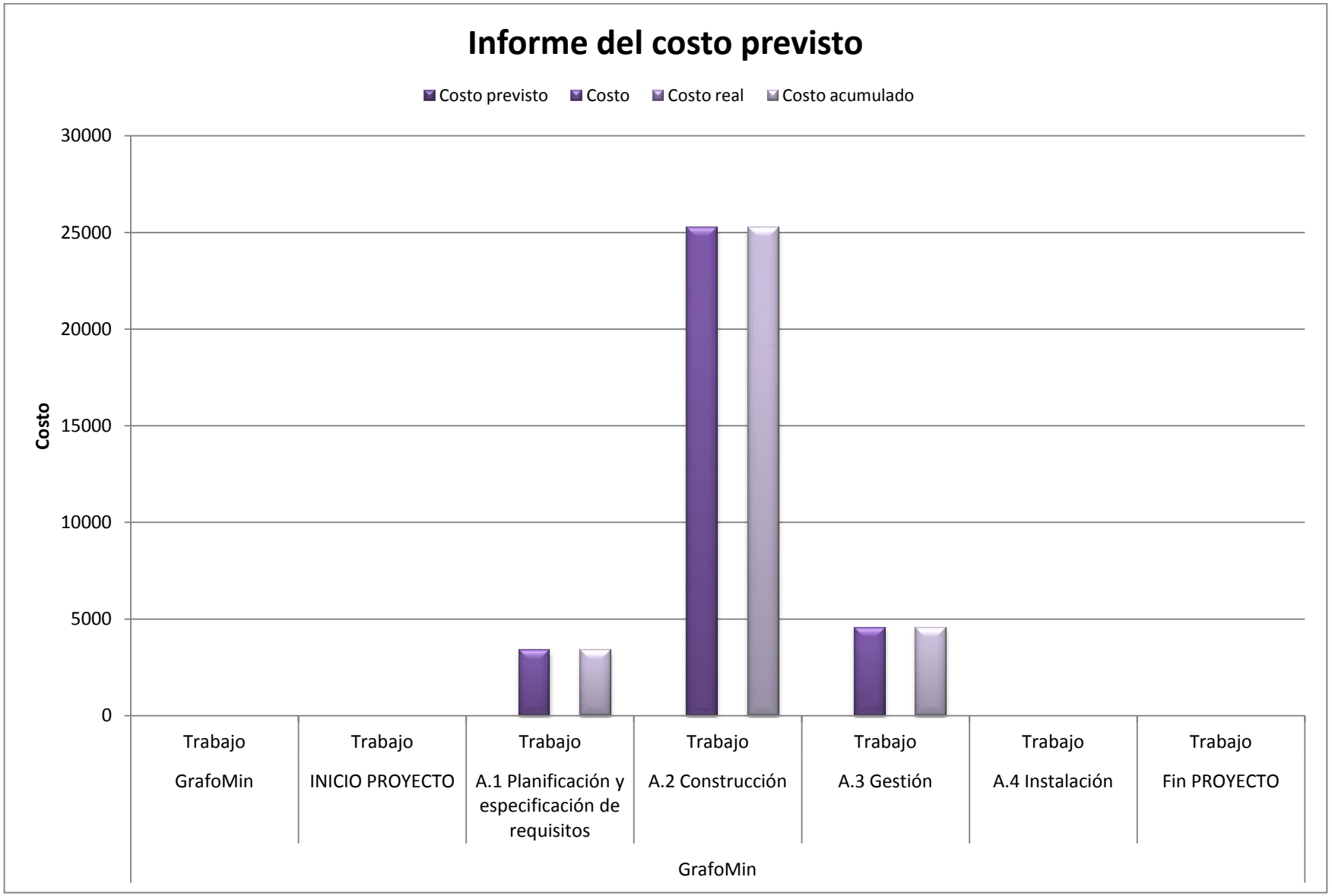
Conforme a los estudios realizados y así reflejados en el diagrama de Gantt adjunto y el archivo *Grafomin.mpp*.

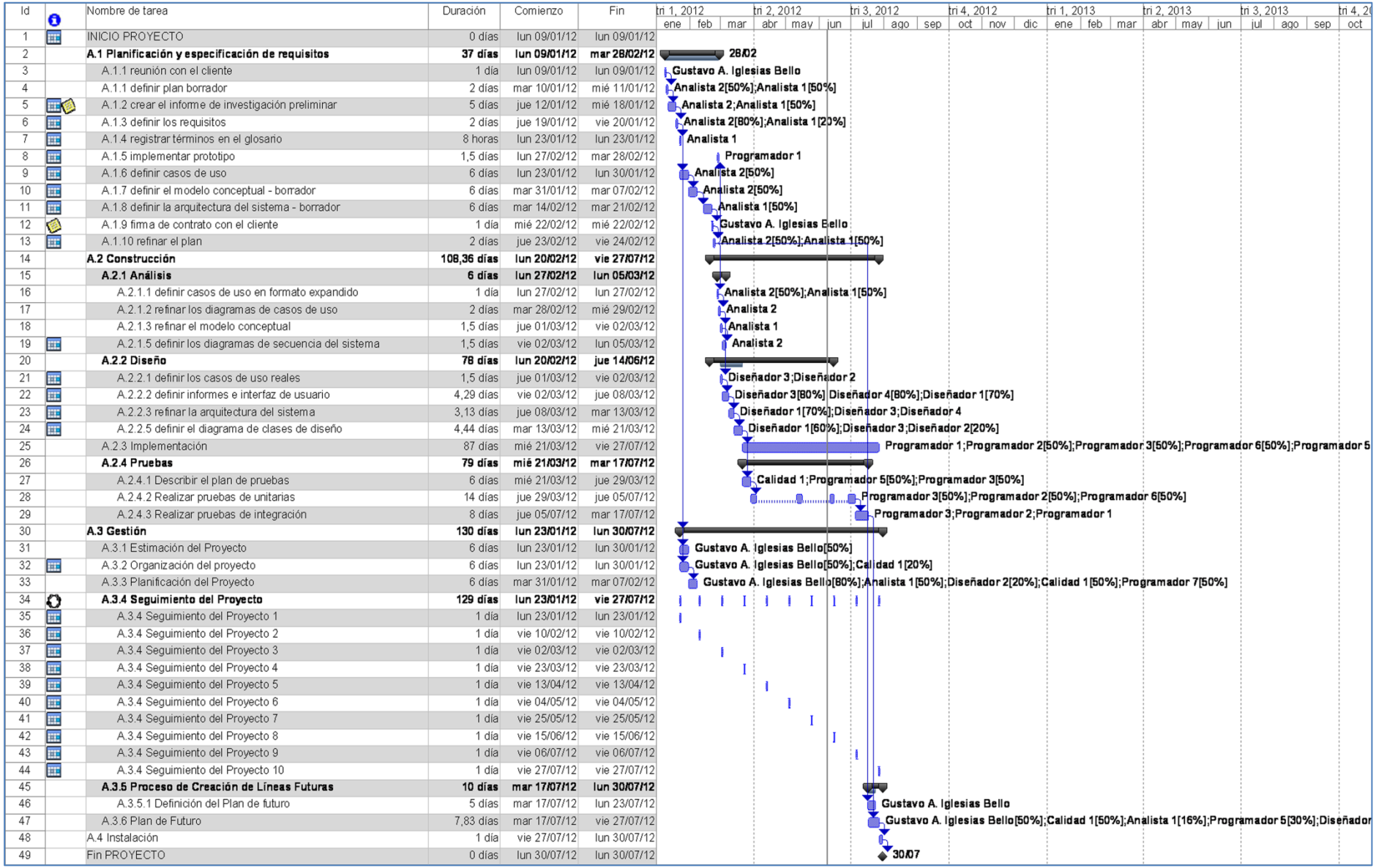
La duración del proyecto, iniciado el lunes 9 de Enero de 2012 y finalizado el lunes 30 de Julio de 2012, es de 140 días laborales trabajados.

Año	Trimestre	Semana	Costo	Costo acumulado	Trabajo	Trabajo acumulado
2012	T1	Semana 2	775,00	775,00	48,00	48,00
		Semana 3	812,50	1587,50	52,00	100,00
		Semana 4	1322,50	2910,00	76,00	176,00
		Semana 5	1717,50	4627,50	109,60	285,60
		Semana 6	739,00	5366,50	48,00	333,60
		Semana 7	250,00	5616,50	16,00	349,60
		Semana 8	525,00	6141,50	32,00	381,60
		Semana 9	1101,00	7242,50	85,20	466,80
		Semana 10	977,87	8220,37	87,43	554,23
		Semana 11	783,58	9003,95	74,88	629,12
		Semana 12	1480,01	10483,96	123,30	752,42
		Semana 13	2551,95	13035,91	213,42	965,83
	Total T1		13035,91	13035,91	965,83	965,83
	T2	Semana 14	1127,92	14163,84	100,25	1066,08
		Semana 15	1575,00	15738,84	140,00	1206,08
		Semana 16	1575,00	17313,84	140,00	1346,08
		Semana 17	1575,00	18888,84	140,00	1486,08
		Semana 18	945,00	19833,84	84,00	1570,08
		Semana 19	1662,07	21495,91	147,75	1717,83
		Semana 20	1056,82	22552,74	93,92	1811,75
		Semana 21	675,00	23227,74	60,00	1871,75
		Semana 22	675,00	23902,74	60,00	1931,75
		Semana 23	675,00	24577,74	60,00	1991,75
		Semana 24	1215,00	25792,74	108,00	2099,75
		Semana 25	675,00	26467,74	60,00	2159,75
		Semana 26	897,08	27364,81	79,75	2239,50
	Total T2		14328,90	27364,81	1273,67	2239,50
	T3	Semana 27	2174,02	29538,84	166,58	2406,08
		Semana 28	1575,00	31113,84	140,00	2546,08
		Semana 29	1334,29	32448,12	117,58	2663,67
		Semana 30	723,64	33171,76	53,92	2717,57
		Semana 31	0,00	33171,76	0,00	2717,57
	Total T3		5806,95	33171,76	478,08	2717,57
Total general			**33171,76**	**33171,76**	**2717,58**	**2717,57**


Informe del flujo de caja
Costo
Costo acumulado
16000
14000
12000
10000
8000
6000
4000
2000
0
35000
30000
25000
20000
15000
10000
5000
0
Costo
Costo acumulado
T1
T2
T3
2012


Informe del trabajo presupuestado
Trabajo
Trabajo acumulado
3000
2500
2000
1500
1000
500
0
Trabajo
GrafoMin
T1
GrafoMin
T2
GrafoMin
T3
2012


Informe del costo previsto
Costo previsto
Costo
Costo real
Costo acumulado
30000
25000
20000
15000
10000
5000
0
Costo
Trabajo GrafoMin
Trabajo INICIO PROYECTO
Trabajo A.1 Planificación y especificación de requisitos
Trabajo A.2 Construcción
Trabajo A.3 Gestión
Trabajo A.4 Instalación
Trabajo Fin PROYECTO
GrafoMin



Id	Nombre de tarea	Duración	Comienzo	Fin
1	INICIO PROYECTO	0 días	lun 09/01/12	lun 09/01/12
2	**A.1 Planificación y especificación de requisitos**	**37 días**	**lun 09/01/12**	**mar 28/02/12**
3	A.1.1 reunión con el cliente	1 día	lun 09/01/12	lun 09/01/12
4	A.1.1 definir plan borrador	2 días	mar 10/01/12	mié 11/01/12
5	A.1.2 crear el informe de investigación preliminar	5 días	jue 12/01/12	mié 18/01/12
6	A.1.3 definir los requisitos	2 días	jue 19/01/12	vie 20/01/12
7	A.1.4 registrar términos en el glosario	8 horas	lun 23/01/12	lun 23/01/12
8	A.1.5 implementar prototipo	1,5 días	lun 27/02/12	mar 28/02/12
9	A.1.6 definir casos de uso	6 días	lun 23/01/12	lun 30/01/12
10	A.1.7 definir el modelo conceptual - borrador	6 días	mar 31/01/12	mar 07/02/12
11	A.1.8 definir la arquitectura del sistema - borrador	6 días	mar 14/02/12	mar 21/02/12
12	A.1.9 firma de contrato con el cliente	1 día	mié 22/02/12	mié 22/02/12
13	A.1.10 refinar el plan	2 días	jue 23/02/12	vie 24/02/12
14	**A.2 Construcción**	**108,36 días**	**lun 20/02/12**	**vie 27/07/12**
15	**A.2.1 Análisis**	**6 días**	**lun 27/02/12**	**lun 05/03/12**
16	A.2.1.1 definir casos de uso en formato expandido	1 día	lun 27/02/12	lun 27/02/12
17	A.2.1.2 refinar los diagramas de casos de uso	2 días	mar 28/02/12	mié 29/02/12
18	A.2.1.3 refinar el modelo conceptual	1,5 días	jue 01/03/12	vie 02/03/12
19	A.2.1.5 definir los diagramas de secuencia del sistema	1,5 días	vie 02/03/12	lun 05/03/12
20	**A.2.2 Diseño**	**78 días**	**lun 20/02/12**	**jue 14/06/12**
21	A.2.2.1 definir los casos de uso reales	1,5 días	jue 01/03/12	vie 02/03/12
22	A.2.2.2 definir informes e interfaz de usuario	4,29 días	vie 02/03/12	jue 08/03/12
23	A.2.2.3 refinar la arquitectura del sistema	3,13 días	jue 08/03/12	mar 13/03/12
24	A.2.2.5 definir el diagrama de clases de diseño	4,44 días	mar 13/03/12	mié 21/03/12
25	A.2.3 Implementación	87 días	mié 21/03/12	vie 27/07/12
26	**A.2.4 Pruebas**	**79 días**	**mié 21/03/12**	**mar 17/07/12**
27	A.2.4.1 Describir el plan de pruebas	6 días	mié 21/03/12	jue 29/03/12
28	A.2.4.2 Realizar pruebas de unitarias	14 días	jue 29/03/12	jue 05/07/12
29	A.2.4.3 Realizar pruebas de integración	8 días	jue 05/07/12	mar 17/07/12
30	**A.3 Gestión**	**130 días**	**lun 23/01/12**	**lun 30/07/12**
31	A.3.1 Estimación del Proyecto	6 días	lun 23/01/12	lun 30/01/12
32	A.3.2 Organización del proyecto	6 días	lun 23/01/12	lun 30/01/12
33	A.3.3 Planificación del Proyecto	6 días	mar 31/01/12	mar 07/02/12
34	**A.3.4 Seguimiento del Proyecto**	**129 días**	**lun 23/01/12**	**vie 27/07/12**
35	A.3.4 Seguimiento del Proyecto 1	1 día	lun 23/01/12	lun 23/01/12
36	A.3.4 Seguimiento del Proyecto 2	1 día	vie 10/02/12	vie 10/02/12
37	A.3.4 Seguimiento del Proyecto 3	1 día	vie 02/03/12	vie 02/03/12
38	A.3.4 Seguimiento del Proyecto 4	1 día	vie 23/03/12	vie 23/03/12
39	A.3.4 Seguimiento del Proyecto 5	1 día	vie 13/04/12	vie 13/04/12
40	A.3.4 Seguimiento del Proyecto 6	1 día	vie 04/05/12	vie 04/05/12
41	A.3.4 Seguimiento del Proyecto 7	1 día	vie 25/05/12	vie 25/05/12
42	A.3.4 Seguimiento del Proyecto 8	1 día	vie 15/06/12	vie 15/06/12
43	A.3.4 Seguimiento del Proyecto 9	1 día	vie 06/07/12	vie 06/07/12
44	A.3.4 Seguimiento del Proyecto 10	1 día	vie 27/07/12	vie 27/07/12
45	**A.3.5 Proceso de Creación de Líneas Futuras**	**10 días**	**mar 17/07/12**	**lun 30/07/12**
46	A.3.5.1 Definición del Plan de futuro	5 días	mar 17/07/12	lun 23/07/12
47	A.3.6 Plan de Futuro	7,83 días	mar 17/07/12	vie 27/07/12
48	A.4 Instalación	1 día	vie 27/07/12	lun 30/07/12
49	Fin PROYECTO	0 días	lun 30/07/12	lun 30/07/12

4.1.3 Organización

Conforme a los plazos de entrega y ejecución establecidos en el apartado anterior, se han establecido los recursos necesarios.

Para la ejecución y planificación de GrafoMin, son necesarios dos tipos de recursos:

- Materiales: ya especificados en la sección 3.4.4.2.2
- Humanos:
  - 1 jefe de proyecto: Gustavo A. Iglesias Bello,
  - 2 analistas:
    - uno de ellos como arquitecto del sistema,
    - otro como líder del equipo de desarrollo,
  - 7 programadores,
  - 4 diseñadores:
    - uno como responsable de materiales y diseñador de la arquitectura,
    - otro como diseñador de la arquitectura,
    - dos como diseñadores gráficos,
  - 1 responsable de calidad y de la verificación de pruebas.

4.1.4 Planificación

Dentro de todas las tareas a ejecutar, la propia planificación del proyecto se especifica en los grupos de tareas A.1 Planificación y especificación de requisitos y, en el grupo de tareas A.3 Gestión, donde se estipula la realización, estimación, planificación y seguimiento del proyecto. Esta última tarea, el seguimiento del proyecto, es una tarea repetitiva que, iniciada tras la creación del documento de requisitos, se ocupa de verificar el cumplimiento de la planificación, resolver los problemas, dudas y soluciones posibles, a todo el proceso de creación de la herramienta: para ello se celebran reuniones semanales de todo el equipo de trabajo.

La primera tarea será siempre, la reunión con el cliente que, en nuestro caso es la tarea A.1.1 reunión con el cliente del diagrama de Gantt adjunto donde se refleja la reunión con D. Eduardo Jesús Sánchez-Villaseñor y D. Jesús Salas Martínez. Es esta una tarea fundamental, pues en ella se define por parte del cliente las necesidades que el equipo deberá satisfacer. Solo ha sido necesario una segunda reunión, en la que los clientes aceptan, en la tarea A.1.8 definir la arquitectura del sistema -borrador-, la ejecución del proyecto de acuerdo con el prototipo creado (tarea A.1.5 implementar prototipo) y el documento de requisitos ya definido (A.1.3 definir los requisitos).

En el grupo segundo, A.2 Construcción, se especifican las tareas a realizar para la creación de GrafoMin. Hay tres secciones principales: A.2.1 Análisis, A.2.2 Diseño y A.2.3 Pruebas. Todas estas tareas han sido descritas detalladamente en el capítulo 3º del presente documento de proyecto.

La fecha de inicio del proyecto ha sido el lunes 9 de Enero de 2012 y el día de finalización el lunes 30 de Julio de 2012. Con un total de 2.718 horas trabajadas.

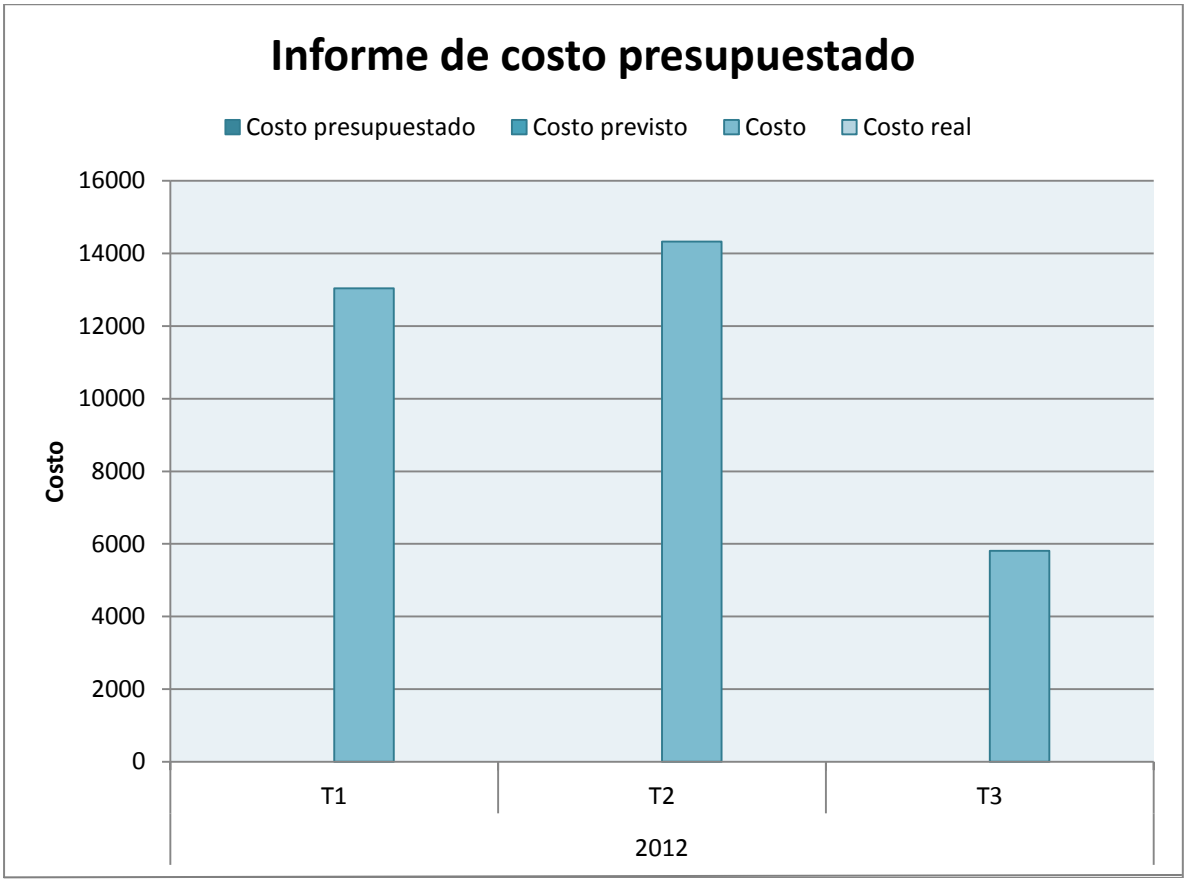
4.2 Presupuesto

Una de las grandes ventajas de utilizar la herramienta Project de Microsoft, es que nos genera de modo automático un presupuesto de costes. Para ello es necesario introducir en el programa, los costes de los recursos materiales y humanos.

Describimos los costes por trabajador para la empresa (incluido sueldo bruto y costes de Seguridad Social y mutualidad) con los que hemos valorado el proyecto. Si bien se citan los costes por hora extra, el proyecto está correctamente proyectado y no requiere de las mismas:

- Jefe de proyecto: 2.200 €/mes – 80 €/hora extra
- Analista: 2.500 €/mes – 60 €/hora extra
- Responsable de calidad: 2.700 €/mes – 70 €/hora extra
- Diseñadores gráficos: 1.500 €/mes – 20 €/hora extra
- Diseñadores de arquitectura: 2.000 €/mes – 50 €/hora extra.
- Programador: 1.800 €/mes – 50 €/hora extra

El costo total de proyecto es de 33.171,76 €



Capítulo 5

Conclusiones y líneas de trabajo

5.1 Introducción

En este capítulo mostramos las conclusiones y las posibles líneas de trabajo futuras que permitirían mejorar la herramienta, así como las diferentes dificultades encontradas al llevar a cabo el proyecto.

5.2 Conclusiones

Durante su fase de prueba (BETA), en la que fue presentado a los usuarios de Internet a través del enlace creado en wikipedia:

http://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

GrafoMin ha obtenido una respuesta aceptable por parte de la comunidad. Esta versión ha sido probada con 245 accesos, de los cuales el 185 son nuevos visitantes y 87 son visitantes que repiten la experiencia -ver gráfico de Google Analytics-. Podemos concluir que la herramienta no solo es aceptada por su interés, sino que tiene un aprovechamiento real dentro de la comunidad. Esto es verificable no solo por las cifras de visitas nuevas y antiguas, sino por la duración de las mismas.

Google Analytics

http://www.yottabyte.es-http://www....
www.yottabyte.es PREDETER...

Páginas de destino

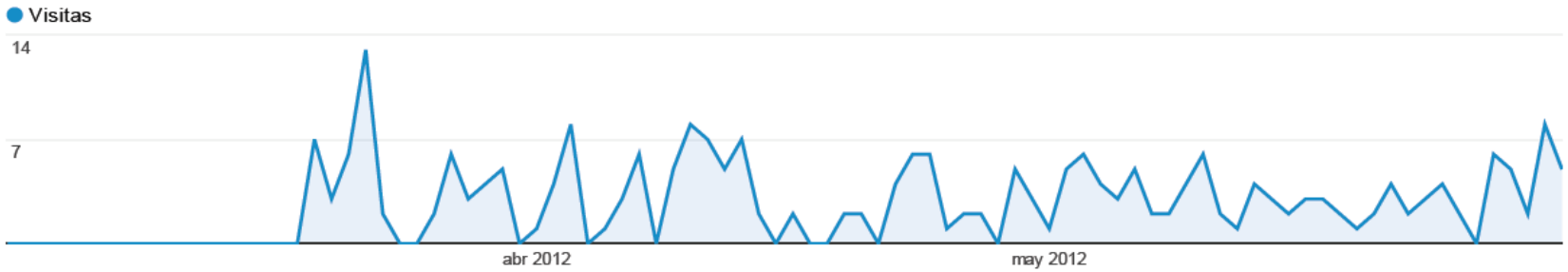
01/03/2012 - 31/05/2012

TODAS » PÁGINA DE DESTINO: /discreta/

% de accesos : 84,78%

Explorador

Uso del sitio



Visitas	Páginas / Visita	Duración media de la visita	Porcentaje de visitas nuevas	Porcentaje de rebote
245	**3,56**	**00:06:51**	**64,49%**	**44,90%**
% del total: 84,78% (289)	Promedio del sitio: 3,35 (6,49%)	Promedio del sitio: 00:06:06 (12,18%)	Promedio del sitio: 66,44% (-2,93%)	Promedio del sitio: 49,83% (-9,89%)

			Total		1. /discreta/	
	Página de destino	Tipo de visitante	Visitas	Duración media de la visita	Visitas	Duración media de la visita
1.	/discreta/	New Visitor	**158**	00:02:34	158	00:02:34
2.	/discreta/	Returning Visitor	**87**	00:14:39	87	00:14:39

5.3 Dificultades

La principal dificultad encontrada ha sido el uso de HTML5, que no siendo un estándar, ha necesitado del conocimiento de su nueva etiqueta <CANVAS> y de la implementación de métodos ya estandarizados en las versiones anteriores. Los citados métodos se muestran en la sección 3.4.2.Métodos globales.

Ha sido necesario un elevado tiempo de programación para el desplazamiento de las cajas indicativas de peso, donde se ha utilizado la ecuación de la recta punto-pendiente, para el desplazamiento de la misma. Esta dificultad ha estribado en que la caja peso (etiqueta <DIV>) tapaba la salida en pantalla de alguno de los vértices de la arista a la que la caja peso hacía referencia. Por tanto, ha sido necesaria la depuración del código para en elevado número de pendientes diferentes (hasta 12 valores diferentes para la pendiente han sido probados). Ha sido, por tanto, especialmente difícil la codificación del método *dameDesplazamiento(xActual,yActual,globales.elComienzoX,globales.elComienzoY,arista)*, que se ocupa de la correcta ubicación de la caja informativa del peso de la línea arista. Se recomienda en futuras actualizaciones no tocar dicha codificación, por su gran dificultad.

No ha sido ajena a la implementación la ya habitual problemática de las diferentes presentaciones que los navegadores hacen de las páginas Web. En este punto, debido al tiempo necesario para su resolución, nos hemos encontrado con un problema insalvable con Microsoft Internet Explorer.

Se ha tratado en todo momento de mantener el diseño líquido de la herramienta. Se llama diseño líquido a aquel diseño que conserva su estructura y aspecto, aunque el navegador cambie el tamaño de la ventana. El principal problema que causa la redimensión de la ventana es, que el aspecto del grafo que hubiera sido dibujado con anterioridad, no mantenga su estructura idéntica, las líneas de arista queden desplazadas con respecto a los vértices y, lo más difícil de evitar, que las cajas informativas con el peso no queden alineadas con la línea arista por lo que, en posteriores desplazamientos a lo largo de dicha línea, no se efectuará correctamente la reubicación de la caja peso.

Un problema insalvable ha sido el no indicar el tamaño de la etiqueta <CANVAS> por medio de CSS, teniendo que hacerlo directamente en la etiqueta con loa atributos with y heigth. Esto está causado porque todavía no está dentro del estándar la correcta codificación de HTML5. En concreto, Microsoft Internet Explorer necesita conocer dichos atributos desde la etiqueta, y no desde la hoja de estilos.

Otro de los inconvenientes que hay que superar con diversa dificultad es el valor de las coordenadas de cualquier objeto: tanto respecto a la propia ventana del navegador, como en la etiqueta <CANVAS>. Este último caso hace especialmente incómoda la programación y, sobre todo, la depuración del código, relativo a posiciones de elementos gráficos. Hemos citado en el párrafo anterior que el método *dameDesplazamiento* es especialmente azaroso en su desarrollo e implementación; pues debemos añadir el inconveniente de la posición relativa de los elementos, atributos en este caso, con respecto al origen de coordenadas. Los orígenes de coordenadas son tomados siempre con respecto al vértice superior izquierdo de cualquier elemento, por tanto, en la mayoría de

los casos, las posiciones de un objeto con respecto a otro en el que está contenido, serán valores del tercer cuadrante de los ejes de coordenadas cartesianos.

5.4 Líneas futuras

Para cualquier modificación que se desee efectuar en el código, se podrá indicar el valor de la variable *globales.debug* con el valor true, para que la herramienta muestre en pantalla diferentes valores que toman, la posición del ratón, el valor de la pendiente de la línea de arista (la última creada), la resolución de la pantalla donde se ejecuta la aplicación y las posiciones de el último vértice con respecto a la etiqueta <CANVAS>.

Se podría ampliar el problema a grafos dirigidos con la posibilidad de incluir dos aristas de sentidos opiestos entre dos vértices. La herramienta solo permite una sola arista, dirigida o no, pero no dos, como se está proponiendo. La herramienta está totalmente preparada para esta ampliación, en lo que se refiere a estructura de los datos y a su ocupación en memoria, solo falta la implementación gráfica en la que se deberá contemplar el caso de que ya exista una arista dirigida que une los dos vértices, supongamos vértices A y B, y donde queremos incluir su opuesta. Para solucionar tal problemática, bastará con borrar la arista gráfica existente dentro de la etiqueta <CANVAS>, usando el método *drawArista(x0,y0,x1,y1)* del fichero *dibuja.js*, el cual dibujará una línea de arista con el mismo fondo que la etiqueta <CANVAS>, quedando así borrada. Se deberá proceder entonces, a dibujar (con el mismo método indicado) ambas aristas opuestas, observando la conveniente separación ente ambas.

Otra posible línea de ampliación consistiría en extender la compatibilidad de la herramienta para Microsoft Internet Explorer. Para ello aconsejamos el uso del depurador que ofrece el fabricante. Cabe notar que, para la ejecución de GrafoMin en los navegadores de Microsoft, será necesario incluir el archivo *excanvas-modified.js*, que permitirá dibujar en la etiqueta <CANVAS>, sin efectuar modificación alguna en el código, ya que utiliza los mismos métodos implementados en la herramienta.

Una evolución muy útil e interesante sería la posibilidad de poder guardar el problema planteado por al usuario para un posterior análisis o reproducción en la misma herramienta. Esto sería especialmente interesante para que los alumnos que no comprendan algún paso o tengan alguna duda, puedan replantearla nuevamente. Para los profesores sería sumamenta práctico para preparar las clases, presentaciónes o explicaciones.

Para el proceso de ejecución de los archivos guardados por los usuarios en sus equipos se debe consultar la especificación correspondiente de la "API v 2011" o bien "API v 2012". Se leerá el archivo local creando un objeto *FileReader*, preferiblemente utilizando el método *FileReader.readAsText(File, opt_encoding)*. Se verificará que se hayan procesado todas las inicializaciones necesarias en el proceso de carga de GrafoMin. Se procesarán los datos leídos ejecutando ya los métodos de la herramienta. Primero se crearán los vértices:

- *draw(tempX,tempY);* //dibuja el punto en canvas donde *TempX* y *TempY* son las coordenadas del vértice a crear.

- *CreaDiv(tempX,tempY);* //crea la capa para los valores y acciones sobre el nodo-vertice-punto.
- *ActualizaVertices(tempX,tempY);* //Guardamos el vértice en el array.

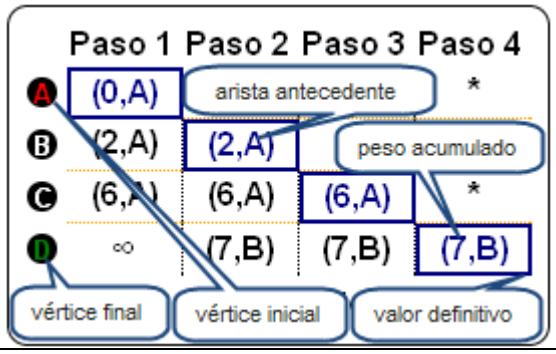
Seguidamente las aristas:

- *var vértice_id_inicial = valor1;//*se tomará del fichero;
- *var vértice_id_final = valor1;//*se tomará del fichero;
- *var arista_id=valor2;//*se tomará del fichero;
- *var j = dameVertice(vertice_id_inicial);//*Punto Inicial.
- *var a = new Arista(j,grafo.id_arista);*
- *BloqueaVertices(vertice_id_inicial);*
- *grafo.arista_iniciada = grafo.id_arista;*
- *ActualizaAristas(a);*
- *FinalArista(vertice_id_final);*
- *CreaCajaArista(dameAristaId(a));//*Incluimos una nueva "*caja arista*".

El en proceso de guardado será necesario guardar en un fichero local los identificadores de los vértices, los identificadores de las aristas y los vértices inicial y final de cada una de las aristas. Como por motivos de seguridad no es posible guardar directamente un fichero en el disco del usuario, se escribirá dicho contenido en una capa <DIV> indicándole al usuario que lo copie y pegue en un archivo con formato de texto plano (tipo "*txt*").

Finalmente, una línea de trabajo adicional sería la creación de un seguimiento analítico más profundo. Este seguimiento consistiría en ir describiendo paso a paso el valor de la matriz analítica en cada paso del algoritmo de Dijkstra, con el fin de que el alumno introdujera los posibles valores de dicha matriz, a cada paso del algoritmo, para así verificar si su conocimiento del citado algoritmo es correcto. Dichos resultados se mostrarán en el área de soluciones, llamado "construcción del camino más corto". Para evaluar correctamente la matriz paso a paso, deberemos modificar o utilizar el método *EscribeMatrizAnalitica()* (ver fichero *midistra.js*) que trata de la creación de la matriz analítica.

Glosario

applet	*Componente de una* aplicación *que se ejecuta en el contexto de otro programa.*
CANVAS	*Elemento HTML5 que permite la generación dinámica de gráficos mediante JavaScript*
DIV	*Elemento capa de HTML*
DOM	*Modelo de Objetos del documento para HTML y XML*
Firebug	*Firebug es una extensión de Firefox creada y diseñada especialmente para desarrolladores y programadores web.*
HTTP	*Protocolo de transferencia de hipertexto utilizado en la World Wide Web*
IFRAME	*Elemento HTML que inserta un marco en un documento*
matriz analítica	*Matriz triangular inferior* $C_{ij} \geq 0$*, descriptiva del valor de los pesos de cada arista, volorado en cada una de las iteraciones del algoritmo de Dijkstra. Donde:* • $i \in \{1 \ldots n\}$ *vértices,* • $j \in \{1 \ldots m\}$ *iteraciones,* • C_{ii} *es el par formado por el valor del camino desde el vértice inicial y el vértice antecedente,* • ∞: *vértice no accesible desde el vértice inicio.* 
SVG	*Especificación para describir gráficos vectoriales bidimensionables -Scalable Gector Graphics-*
UML	*Lenguaje Unificado de Modelado (UML - Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group).*
usabilidad	*El neologismo usabilidad (del inglés "usability") se refiere a la facilidad con que las personas pueden utilizar y aprender el uso de una herramienta particular (o cualquier otro objeto fabricado por humanos) con el fin de alcanzar un objetivo concreto.*

UTF-8	*formato de codificación de caracteres utilizando símbolos de longitud variable*
WebKit	*Plataforma para aplicaciones*
W3C	*Consorcio internacional que produce recomendaciones para la World Wide Web*

Referencias

ASOCIACIÓN ESPAÑOLA DE USUARIOS DE INTERNET. Información corporativa. *Accesibilidad* [en línea]. [Fecha de consulta: mayo de 2012]. Disponible en Internet: http://www.aui.es/index.php?body=asoc_v1article&id_article=350

BIGGS, Norman L. Árboles, ordenación y búsqueda. *Matemática discreta*. 1ª ed. Ediciones Vicens Vives, S.A. Barcelona 1998, p 225-230. ISBN: 84-316-3311-5.

CHAPEL, David. *Simple Object Access Protocol and firewalls* [en línea] [Fecha de consulta : enero 2012]. Disponible en Internet]: http://msdn.microsoft.com/workshop/xml/general/SOAP_White_Paper.asp

FIELDING, R.; Gettys, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, p.; BERNERS-LEE, T. *Hypertext Transfer Protocol -- HTTP/1.1*. Internet Engineering Task Force [en línea]. [Fecha de consulta: enero de 2012]. Disponible en Internet: http://www.ietf.org/rfc/rfc2616.txt

HEWITT, Joe ; ODVARKO, Jan ; FIREBUG WORKING GROUP. Mozilla - Complementos. *Firebug 1.9.2* [en línea]. [Fecha de consulta: 31 de enero de 2012]. Disponible en Internet: https://addons.mozilla.org/es-es/firefox/addon/firebug/

KASSEM, Nicholas; ENTERPRISE TEAM. *Designing enterprise applications with the Java™ 2 Platform, Enterprise Edition*. Addison Wesley, 2000. ISBN: 978-0201702774.

OMG MISSION STATEMET. *Catalog of OMG CORBA/IIOP Specifications* [en línea]. [Fecha de consulta: enero de 2012]. Disponible en Internet: http://www.omg.org/technology/documents/formal/corba_iiop.htm

OMG MISSION STATEMET. *UML - Object Management Group* - Versión: 2.4.1. [en línea]. [Fecha de consulta: enero de 2012]. Disponible en Internet: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

RAPPA, Michael. *Business Models on the Web* [en línea]. Digital Enterprise Report. 2002. [Fecha de consulta: 16 de enero de 2012]. Disponible en Internet: http://digitalenterprise.org/models/models.html

REAL ACADEMIA ESPAÑOLA DE LA LENGUA. *Diccionario de la Lengua Española*. Edición 21ª [en línea]. 1993. ISBN: 84-239-6813-8. [Fecha de última consulta: 3 de Julio de 2012]. Disponible en Internet: http://www.rae.es

RODRÍGUEZ VILLALOBOS, Alejandro. *Grafos - software para la construcción, edición y análisis de grafos.* Bubok Publishing S.L. 2010. ISBN: 978-84-9981-116-1

SALLÁN LEYES, J.M.; SUÑÉ, A.; FERNÁNDEZ, V.; FONOLLOSA, J.B. Teoría de grafos.*Métodos cuantitativos de organización industrial* I. 2ª ed. Ediciones Universitat Politècnica de Catalunya S.L. Barcelona 2005, p. 158-162. [Fecha de consulta: 18 de enero de 2012]. ISBN: 84-8301-795-4. Disponible en Internet: http://books.google.es/books?id=PLSM9yj8tHAC&pg=PA158&lpg=PA158&dq=resoluci%C3%B3n+camino+m%C3%A1s+corto+dijkstra&source=bl&ots=SbjVlggpWJ&sig=Qnq2K285lYOr9NgeeKYWfXB87kE&hl=es&sa=X&ei=pSuQT5b4A8m6hAe46ciZBA&sqi=2&ved=0CE4Q6AEwBQ#v=onepage&q=resoluci%C3%B3n%20camino%20m%C3%A1s%20corto%20dijkstra&f=false

UNIVERSIDAD DE LA LAGUNA. *UNE 50-104-94 Norma de referencias bibliográficas* [en línea]. [Fecha de consulta : 2 de julio de 2012]. Disponible en Internet : http://www.ull.es/view/institucional/bbtk/Referencias_normas_UNE/es

.NET FRAMEWORK 4. *Microsoft Corporation* [en línea]. [Fecha de consulta: enero de 2012]. Disponible en Internet: http://www.microsoft.com/net