



UNIVERSIDAD POLITÉCNICA DE MADRID
FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

Aplicación Integral de Grafos: Búsquedas

AUTOR: Juan Carlos Delgado Núñez
TUTOR: Gregorio Hernández Peñalver

ÍNDICE

1 INTRODUCCIÓN Y ALCANCE	1
2 TEORÍA DE GRAFOS	2
2.1 Introducción.....	2
2.2 Historia	2
2.3 Conceptos teóricos.....	3
2.3.1 Vértice.....	3
2.3.2 Arista.....	4
2.3.3 Grafo	4
2.3.4 Grafo no dirigido	5
2.3.5 Grafo dirigido	5
2.3.6 Propiedades de los grafos.....	6
2.3.7 Algunos grafos básicos.....	7
2.3.7.1 Grafo completo.....	7
2.3.7.2 Ciclo.....	7
2.3.7.3 Cubo	8
2.3.7.4 Árbol.....	10
2.3.7.5 Grafo bipartito	11
3 ALGORITMOS IMPLEMENTADOS	12
3.1 DFS (Depth-First Search)	12
3.1.1 Descripción.....	12
3.1.2 Estrategia	13
3.2 BFS (Breadth-First Search).....	16
3.2.1 Descripción.....	16
3.2.2 Estrategia	17
3.2.3 Aplicaciones	18
3.2.4 Ejemplo práctico: Jugar con ruedas	19
3.3 Problema del camino mínimo.....	21
3.3.1 Algoritmos	22
3.3.2 Algoritmo de Dijkstra.....	22
3.3.3 Estrategia	24
3.4 Bloques.....	24
3.4.1 Intersección de bloques.....	25
3.4.2 Algoritmo de bloques	25
4 DISEÑO DE LA APLICACIÓN	28
4.1 Lenguaje de programación.....	28
4.2 Diseño de Alto Nivel.....	29
4.2.1 Definición de los límites del sistema	29
4.2.2 Diagramas de casos de uso	30
4.2.2.1 Generar Grafo	33
4.2.2.2 EstablecerConfiguración	36

4.2.2.3 EjecutarAlgoritmo	37
4.2.3 Descripción de los casos de uso en formato extendido	38
4.3 Diseño de Bajo Nivel	57
4.3.1 Diagrama de clases	57
4.3.1.1 Visión general	58
4.3.1.2 Grafo	60
4.3.1.3 Algoritmo	62
5 MANUAL DE USUARIO	71
5.1 Partes de la aplicación	71
5.1.1 Barra de Menús	71
5.1.1.1 Archivo	72
5.1.1.2 Editar	73
5.1.1.3 Grafos	73
5.1.1.4 Herramientas	74
5.1.1.5 Ayuda	75
5.1.2 Barra de Iconos (botones)	76
5.1.3 Información	79
5.1.4 Posición	82
5.1.5 Lienzo	82
5.2 Operaciones genéricas	85
5.2.1 Crear nuevo grafo automático	85
5.2.2 Crear nuevo grafo personalizado	87
5.2.3 Guardar grafo	89
5.2.4 Abrir grafo	90
5.2.5 Cerrar	91
5.2.6 Limpiar algoritmos	91
5.2.7 Exportar grafo	92
5.2.8 Salir	93
5.2.9 Editar nodo	93
5.2.10 Insertar vértices	95
5.2.11 Insertar arista	95
5.2.12 Eliminar nodo	95
5.2.13 Suprimir arista	96
5.2.14 Cambiar las opciones	96
5.2.15 Imagen de fondo	100
5.2.16 Estadísticas	100
5.2.17 Matriz de adyacencia	101
5.2.18 Matriz de ponderación	102
5.2.19 Grafos tipo	103
5.3 Ejecución	105
5.3.1 Ejecución automática	108
5.3.2 Ejecución interactiva	109
5.3.2.1 Coloración: Algoritmo Secuencial Básico	110
5.3.2.2 Coloración: Algoritmo Secuencial de Welsh-Powell	113
5.3.2.3 Coloración: Algoritmo Secuencial de Matula-Marble-Isaacson	114

5.3.2.4	Coloración: Algoritmo de Coloración de Brelaz	115
5.3.2.5	Coloración: Algoritmo de Independencia MCI	116
5.3.2.6	Búsqueda: Algoritmo DFS.....	120
5.3.2.7	Búsqueda: Algoritmo BFS.....	123
5.3.2.8	Búsqueda: Conectividad.....	125
5.3.2.9	Búsqueda: Algoritmo de Dijkstra	127
6	CONCLUSIONES.....	130
7	BIBLIOGRAFÍA.....	131

LISTADO DE ILUSTRACIONES

Figura 1. Problema de los puentes de Königsberg	3
Figura 2. Grafo no dirigido.....	5
Figura 3. Grafo dirigido	6
Figura 4. Ejemplo K_4	7
Figura 5. Ejemplo C_5	7
Figura 6. Cubo de dimensión 1	8
Figura 7. Cubo de dimensión 2	8
Figura 8. Cubo de dimensión 3	9
Figura 9. Cubo de dimensión 4	9
Figura 10. Ejemplo de árbol de 8 vértices.....	10
Figura 11. Dos realizaciones del mismo grafo bipartito.....	11
Figura 12. Grafos $K_{2,3}$ y $K_{3,3}$	11
Figura 13. Ejemplo de laberinto.....	12
Figura 14. Secuencia de ejecución DFS	14
Figura 15. Ejecución DFS finalizada sobre un digrafo	15
Figura 16. Ejemplo de grafo sobre el que aplicaremos BFS	16
Figura 17. Ejemplo anterior tras la ejecución de BFS.....	17
Figura 18. Árbol de búsqueda generado por BFS	18
Figura 19. Dibujo ilustrativo del ejemplo	20
Figura 20. Ejemplo de ejecución del algoritmo de Dijkstra	23
Figura 21. Diagrama de nivel 0	32
Figura 22. Diagrama de nivel 1: GenerarGrafo.....	33
Figura 23. Diagrama de nivel 2: GenerarGrafoAleatorio	33
Figura 24. Diagrama de nivel 2: GenerarGrafoPersonalizado.....	34
Figura 25. Diagrama de nivel 2: GenerarGrafoTipo	35
Figura 26. Diagrama de nivel 1: EstablecerConfiguración.....	36
Figura 27. Diagrama de nivel 1: EjecutarAlgoritmo	37
Figura 28. Diagrama de Clases que muestra la Visión Genera del Proyecto	59
Figura 29. Diagrama de Clases con la clase Grafo (y las clases con las que se relaciona)	61
Figura 30. Diagrama de Clases que muestra los Algoritmos DFS y BFS.....	63
Figura 31. Diagrama de Clases que muestra los Algoritmos de Bloques y Dijkstra.....	65
Figura 32. Diagrama de Clases con los Algoritmos de Coloración (Secuencial y Brelaz).....	67
Figura 33. Diagrama de Clases con el Algoritmo de Conjuntos Independientes	68
Figura 34. Diagrama de Clases que muestra la Configuración	70
Figura 35. Menú Archivo	72
Figura 36. Menú Editar	73
Figura 37. Menú Grafos	74

Figura 38. Menú Herramientas.....	74
Figura 39. Menú Ayuda	75
Figura 40. Barra de Iconos.....	76
Figura 41. Árbol de un grafo.....	79
Figura 42. Leyenda del algoritmo de Bloques.....	80
Figura 43. Información de la ejecución de un algoritmo	81
Figura 44. Cuadro de posición del cursor.....	82
Figura 45. Lienzo de la aplicación mostrando un grafo	82
Figura 46. Menú contextual de edición de grafo.....	83
Figura 47. Menú contextual de ejecución de algoritmo.....	83
Figura 48. Ventana de información de vértice sobre el lienzo	84
Figura 49. Ventana de creación de un nuevo grafo automático	85
Figura 50. Detalle de la aplicación una vez creado un grafo	86
Figura 51. Ventana de confirmación para salvar el grafo.....	86
Figura 52. Ventana de creación de un nuevo grafo personalizado	87
Figura 53. Edición de nodos en la creación de un grafo (no dirigido) personalizado	87
Figura 54. Edición de nodos en la creación de un grafo (dirigido) personalizado	88
Figura 55. Ventana de selección del archivo (grafo) a cargar en la aplicación.....	90
Figura 56. Menú Archivo → Exportar a.....	92
Figura 57. Ventana de edición de un grafo no dirigido.....	93
Figura 58. Ventana de edición de un grafo dirigido.....	94
Figura 59. Ventana de opciones.....	97
Figura 60. Detalle de un elemento de la ventana de opciones.....	97
Figura 61. Ventana de selección de color.....	98
Figura 62. Ventanas de selección de algoritmos para la generación de estadísticas.....	101
Figura 63. Ventana de la matriz de adyacencia.....	102
Figura 64. Ventana de la matriz de ponderación.....	103
Figura 65. Ventana de selección de algoritmos de coloración	105
Figura 66. Detalle de la tabla del algoritmo de Brelaz.....	106
Figura 67. Ventana de selección de algoritmos de búsqueda	107
Figura 68. Pestaña del algoritmo de Dijkstra en la ventana de selección de algoritmos	108
Figura 69. Menú contextual de selección de color (algoritmos de coloración).....	110
Figura 70. Detalle de un vértice coloreado	110
Figura 71. Ventana de alerta: color incorrecto.....	111
Figura 72. Ventana de alerta: color no adecuado.....	111
Figura 73. Ventana de alerta: vértice ya coloreado	112
Figura 74. Cuadro de información del algoritmo de coloración secuencial básico.....	112
Figura 75. Ventana de alerta: vértice incorrecto.....	113
Figura 76. Ventana de alerta: vértice no adecuado	114

Figura 77. Ventana de alerta: vértice incorrecto.....	115
Figura 78. Detalle de la tabla del algoritmo de Brelaz.....	115
Figura 79. Menú contextual en la ejecución del algoritmo MCI.....	116
Figura 80. Detalle del lienzo en el algoritmo MCI, mostrando vértices no disponibles.....	117
Figura 81. Detalle del lienzo en el algoritmo MCI, después de recalcular el grafo.....	118
Figura 82. Detalle del lienzo en el algoritmo MCI, con los vértices coloreados.....	119
Figura 83. Ventana de algoritmo de búsqueda DFS. Selección de vértice de inicio	120
Figura 84. Menú contextual en la ejecución del algoritmo DFS	120
Figura 85. Ventana de alerta: vértice incorrecto.....	121
Figura 86. Ventana de alerta: vértice no adecuado.....	122
Figura 87. Ventana de algoritmo de búsqueda BFS. Selección de vértice de inicio.....	123
Figura 88. Menú contextual en la ejecución del algoritmo BFS.....	123
Figura 89. Ventana de alerta: vértice incorrecto.....	124
Figura 90. Ventana de alerta: vértice no adecuado.....	124
Figura 91. Menú contextual en la ejecución del algoritmo de Bloques.....	125
Figura 92. Ventana de alerta: vértice no determinado correctamente	126
Figura 93. Bloques en que queda descompuesto el grafo inicial tras la ejecución.....	126
Figura 94. Ventana de algoritmo de Dijkstra. Selección de vértice de inicio.....	127
Figura 95. Menú contextual de Dijkstra. Seleccionar vértice	127
Figura 96. Detalle del lienzo en el algoritmo de Dijkstra	128
Figura 97. Ventana de alerta: vértice seleccionado incorrecto.....	128
Figura 98. Menú contextual de Dijkstra. Establecer distancia	129
Figura 99. Ventana de alerta: distancia incorrecta.....	129

AGRADECIMIENTOS

En primer lugar quiero dar las gracias al tutor del proyecto, **Gregorio Hernández Peñalver**, del Departamento de Matemática Aplicada de la Facultad de Informática de Madrid. Ha sido quien me ha guiado por el camino correcto cuando he tenido dudas a lo largo de todo este tiempo.

Quiero mencionar a las personas que han hecho posible mi formación académica, y agradecerles todo el esfuerzo que han puesto para que haya podido llegar hasta aquí. En primer lugar a mi madre **Guadalupe**, por dármelo todo; a mis abuelas, donde estéis, **Carmen** y **Avelina**, a mi tía **Mari Carmen**, **gracias a todas de corazón**. Sin vuestro esfuerzo no lo hubiera conseguido.

Quiero dar las gracias a **Yolanda**, por estar ahí en todo momento, facilitándome las cosas, animándome a no rendirme cuando lo fácil hubiera sido abandonar, por insistirme en acabar este proyecto.

Gracias **Abraham**, compañero de aventuras y amigo, desde el principio, desde aquella mañana en que nos conocimos en la cola de Secretaría para matricularnos de primer curso. Ha sido un verdadero placer trabajar a tu lado. Gracias por tu paciencia. Y no lo dudes, seguiremos haciendo cosas, porque si no, nos aburriremos.

Quiero dar las gracias al resto de gente que me ha apoyado todo este tiempo. A **Marina** e **Ignacio**, por todos los años de amistad, a **Raúl** y **Laura**, por hacerme más llevadera la travesía por el desierto, a **Miguel** y **Ana**, a **Fernando**, **Ángela**, **Sara**. Gracias a todos, por ser como sois, porque en algún momento habéis sido una pieza clave, gracias.

Finalmente, quiero agradecer a **Consuelo** su trabajo desinteresado en el proyecto. Gracias por traducir, traducir, traducir...

1 INTRODUCCIÓN Y ALCANCE

El objetivo de este proyecto es desarrollar una aplicación software que permita generar grafos, sobre los que posteriormente podrán aplicárseles una serie de algoritmos. La aplicación se ha denominado IAGraph.

El usuario podrá generar grafos de forma manual o automática, así como hacer uso de una serie de grafos predefinidos, muchos de ellos de sobra conocidos. Los algoritmos que se le podrán aplicar podemos dividirlos en dos grupos: algoritmos de **búsqueda** y algoritmos de **coloración**.

De ahora en adelante nos centraremos principalmente en los algoritmos de búsqueda, aunque haremos referencia en algún momento a los algoritmos de coloración.

La aplicación AIG tiene implementados los siguientes algoritmos de búsqueda: Depth First Search (o **DFS**, búsqueda en profundidad), Breadth First Search (o **BFS**, búsqueda en anchura), algoritmo de **Bloques** (enfocado a la conectividad de un grafo) y algoritmo de **Dijkstra** (búsqueda de caminos mínimos).

El usuario podrá ejecutar los algoritmos de forma automática (paso a paso o *de un tirón*) o de forma interactiva. En este último caso, la aplicación se transforma en una sencilla pero potente herramienta de aprendizaje, donde el usuario deberá ir decidiendo el siguiente paso (siguiente vértice, distancia,...) a medida que se avanza en la ejecución, lo que ayudará a afianzar su conocimiento. Por otra parte es altamente personalizable, por lo que se adaptará a cualquier usuario.

Una curiosa funcionalidad añadida a la aplicación es la posibilidad de poder visualizar entornos de trabajo real, tales como mapas cartográficos, diagramas de redes, planos, etc., sobre los que es posible estudiar las diferentes estrategias de búsqueda mediante los algoritmos ya comentados.

En el aspecto puramente de técnico, IAGraph tiene detrás más de 23.000 líneas de código abierto y ha sido diseñada con un bajo nivel de acoplamiento entre sus módulos. Esto, unido al hecho de que ha sido implementada como un motor de ejecución de pasos, hará que sea muy sencilla su reutilización, en previsión de futuras ampliaciones y mejoras.

2 TEORÍA DE GRAFOS

2.1 Introducción

La teoría de grafos es el estudio de los grafos, estructuras matemáticas utilizadas para modelizar relaciones entre objetos de un conjunto. En este contexto, un grafo es un conjunto no vacío de vértices o nodos y una colección de aristas que conectan pares de vértices. Un grafo puede ser dirigido, es decir, sus aristas conectan puntos de origen con puntos de destino, o no dirigido, entendiendo que no hay distinción entre los dos nodos que conecta cada una de ellas.

Gráficamente, un grafo se representa mediante una serie de puntos (los vértices) conectados por líneas (las aristas).

2.2 Historia

El primer artículo científico relativo a grafos fue escrito por el matemático suizo Leonhard Euler en 1736. Euler se basó en su artículo en el problema de los puentes de Königsberg. La ciudad de Kaliningrado, originalmente Königsberg, es famosa por sus siete puentes que unen ambas márgenes del río Pregel con dos de sus islas. Dos de los puentes unen la isla mayor con la margen oriental y otros dos con la margen occidental. La isla menor está conectada a cada margen por un puente y el séptimo puente une ambas islas. El problema planteaba lo siguiente: ¿es posible, partiendo de un lugar arbitrario, regresar al lugar de partida cruzando cada puente una sola vez?

Abstrayendo este problema y planteándolo con la (entonces aún básica) teoría de grafos, Euler consigue demostrar que el grafo asociado al esquema de puentes de Königsberg no tiene solución, es decir, no es posible regresar al vértice de partida sin pasar por alguna arista dos veces.

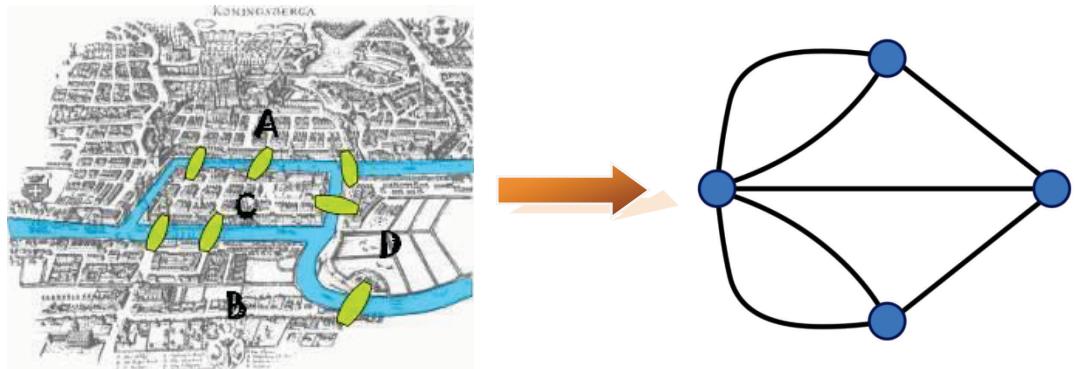


Figura 1. Problema de los puentes de Königsberg

De hecho, Euler resuelve el problema más general: ¿qué condiciones debe satisfacer un grafo para garantizar que se puede regresar al vértice de partida sin pasar por la misma arista más de una vez? Si definimos como "grado" al número de líneas que se encuentran en un punto de un grafo, entonces la respuesta al problema es que los puentes de un pueblo se pueden atravesar exactamente una vez si, salvo a lo sumo dos, todos los puntos tienen un grado par.

2.3 Conceptos teóricos

A continuación se exponen una serie de conceptos relativos a la Teoría de Grafos necesarios para este proyecto.

2.3.1 Vértice

Un **vértice** o **nodo** es la unidad fundamental de la que están formados los grafos. Los vértices son tratados como objetos indivisibles y sin propiedades, aunque puedan tener una estructura adicional dependiendo de la aplicación por la cual se usa el grafo; por ejemplo, una red semántica es un grafo en donde los vértices representan conceptos o clases de objetos.

Los dos vértices que conforman una arista se llaman **puntos finales**, y esa arista se dice que es **incidente** a los vértices. Un vértice w es **adyacente** a otro vértice v si el grafo contiene una arista (v,w) que los une.

2.3.2 Arista

Una **arista** corresponde a una relación entre dos vértices de un grafo. Gráficamente las aristas se representan, para el caso de los grafos no dirigidos, como una línea que une a los dos vértices. Si el grafo es dirigido, entonces la arista se representa como una flecha, que parte del nodo origen y apunta al nodo destino.

Por otro lado, también es normal que las aristas lleven asociadas una etiqueta (un número, una letra o un valor cualquiera) que indica una información asociada a ambos vértices, a veces un *coste* o indicación del trabajo necesario para recorrer el camino de un vértice al otro.

2.3.3 Grafo

Un **grafo** G es un par ordenado $G = (V, E)$, donde:

- V es un conjunto de vértices o nodos, y
- E es un conjunto de aristas o arcos, que relacionan estos nodos.

Normalmente V suele ser **finito**. Muchos resultados importantes sobre grafos no son aplicables para grafos infinitos.

Se llama **orden** del grafo G a su número de vértices $|V|$.

El **grado** de un vértice o nodo V es igual al número de arcos E que se encuentran en él.

Un **bucle** es una arista que relaciona al mismo nodo; es decir, una arista donde el nodo inicial y el nodo final coinciden.

2.3.4 Grafo no dirigido

Un **grafo no dirigido** es un grafo $G = (V, E)$ donde:

- $V \neq \emptyset$
- $E \subseteq \{x \in P(V) : |x| = 2\}$ es un conjunto de pares no ordenados de elementos de V .

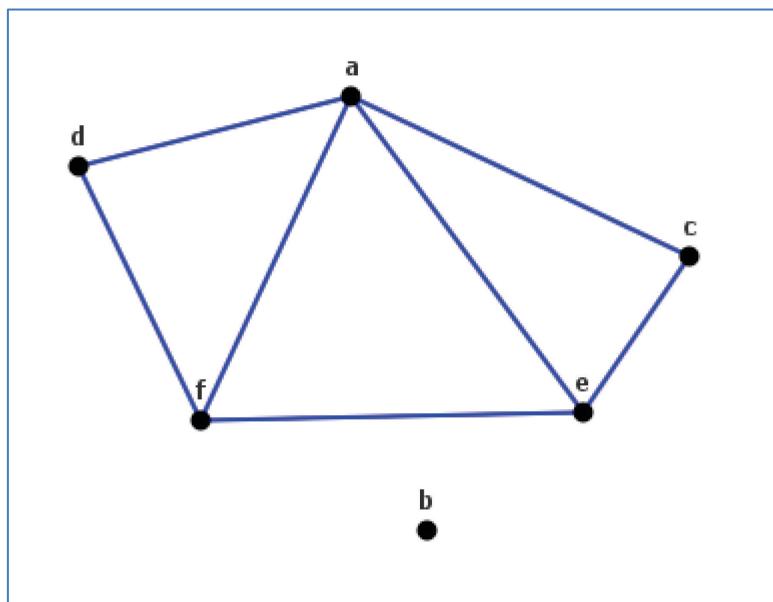


Figura 2. Grafo no dirigido

Un **par no ordenado** es un conjunto de la forma $\{a, b\}$, de manera que $\{a, b\} = \{b, a\}$. Para los grafos, estos conjuntos pertenecen al conjunto potencia de V de cardinalidad 2, el cual se denota por $P(V)$.

2.3.5 Grafo dirigido

Un **grafo dirigido** es un grafo $G = (V, E)$ donde:

- $V \neq \emptyset$
- $E \subseteq \{(a, b) \in V \times V : a \neq b\}$ es un conjunto de pares ordenados de elementos de V .

Dada una arista (a, b) , a es su nodo inicial y b su nodo final.

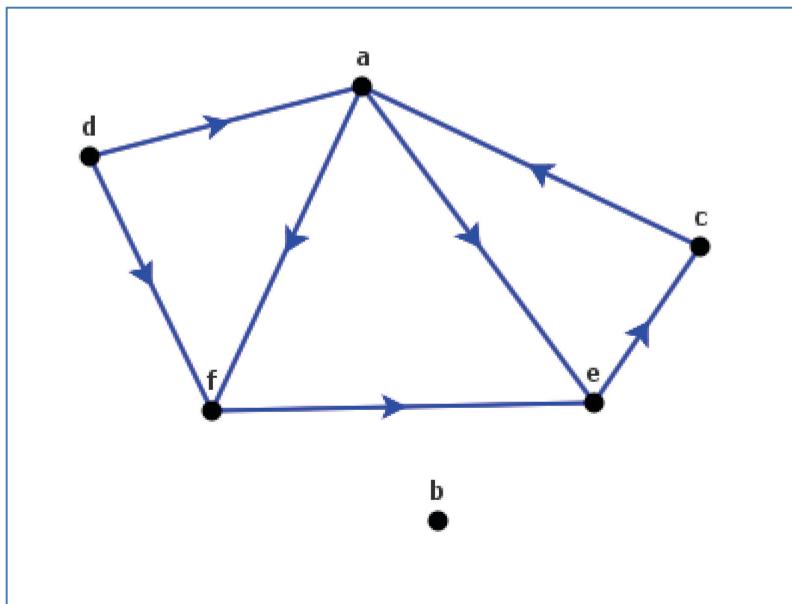


Figura 3. Grafo dirigido

Por definición, los grafos dirigidos no contienen bucles.

Un **grafo mixto** es aquel que se define con la capacidad de poder contener aristas dirigidas y no dirigidas. Tanto los grafos dirigidos como los no dirigidos son casos particulares de éste.

2.3.6 Propiedades de los grafos

Dos aristas son adyacentes si tienen un vértice en común. Dos aristas de un grafo dirigido son consecutivas si el comienzo de una es el fin de la otra. De forma similar, dos vértices son adyacentes si comparten una arista, en cuyo caso se dice que dicha una arista los une.

Un grafo con un vértice y sin aristas se denomina trivial. Un grafo con vértices pero sin aristas se denomina grafo nulo.

En un grafo o digrafo (grafo dirigido) ponderado, cada arista está asociada con un valor, llamado coste, peso, longitud o cualquier otro término relacionado con su aplicación. Es el caso de los problemas de recorridos entre ciudades, donde el valor de la arista es la distancia entre las ciudades que une.

2.3.7 Algunos grafos básicos

2.3.7.1 Grafo completo

Sea G un grafo de orden p . G es completo si dos vértices cualesquiera del grafo son siempre adyacentes, es decir, todos sus vértices tienen valencia o grado $p-1(k_p)$.

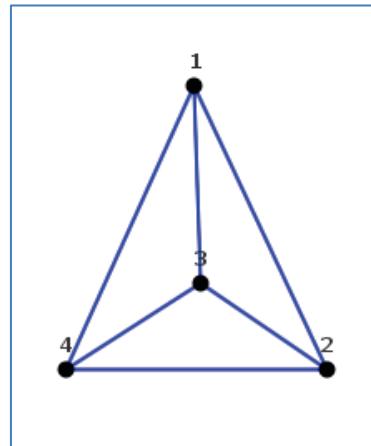


Figura 4. Ejemplo K_4

2.3.7.2 Ciclo

Los ciclos C_n o n -ciclos para $p \geq 3$, son grafos que se asemejan a polígonos de p lados. Un ciclo tiene longitud par si n es par y longitud impar si n es par.

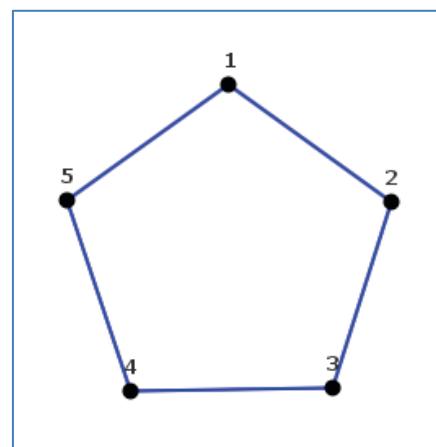


Figura 5. Ejemplo C_5

2.3.7.3 Cubo

Otra de las familias de grafos que suelen considerarse con bastante asiduidad es la de los cubos. Esta familia está formada por los cubos de las distintas dimensiones, desde el cubo de dimensión 1.

Sea K un entero positivo mayor que 1. El k -cubo, Q_k , es el grafo cuyos vértices son las k -tuplas ordenadas de ceros y unos, en el cual dos vértices están unidos por una arista si y sólo si difieren exactamente en una posición.

Así, por ejemplo, para $k=3$, los vértices son $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$, $(0, 1, 1)$, $(1, 0, 0)$, $(1, 0, 1)$, $(1, 1, 0)$, y $(1, 1, 1)$ y, por ejemplo, $(0, 0, 0)$ está unido con $(0, 0, 1)$, $(0, 1, 0)$ y $(1, 0, 0)$ pero no con ningún otro vértice.



Figura 6. Cubo de dimensión 1

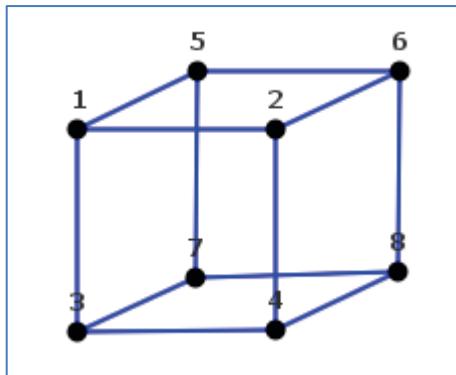


Figura 7. Cubo de dimensión 2

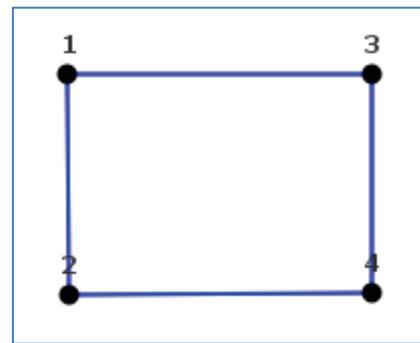


Figura 8. Cubo de dimensión 3

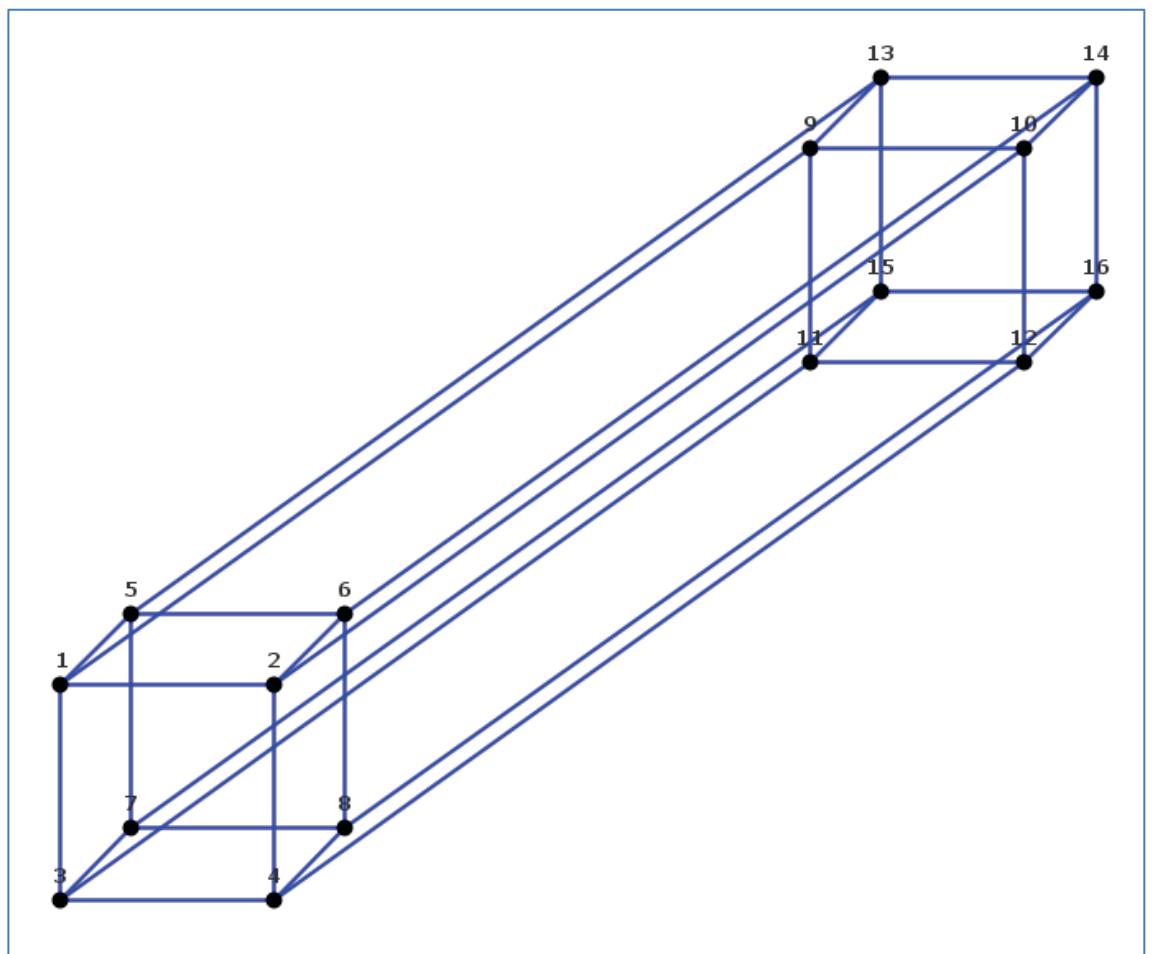


Figura 9. Cubo de dimensión 4

2.3.7.4 Árbol

Un **árbol** de n vértices es un grafo conexo, sin ciclos con exactamente $n-1$ aristas. Además, posee las siguientes propiedades:

- Dos vértices cualesquiera están unidos por un único camino.
- Si al grafo le eliminamos cualquiera de sus aristas queda dividido exactamente en dos componentes conexas dando lugar a un bosque.

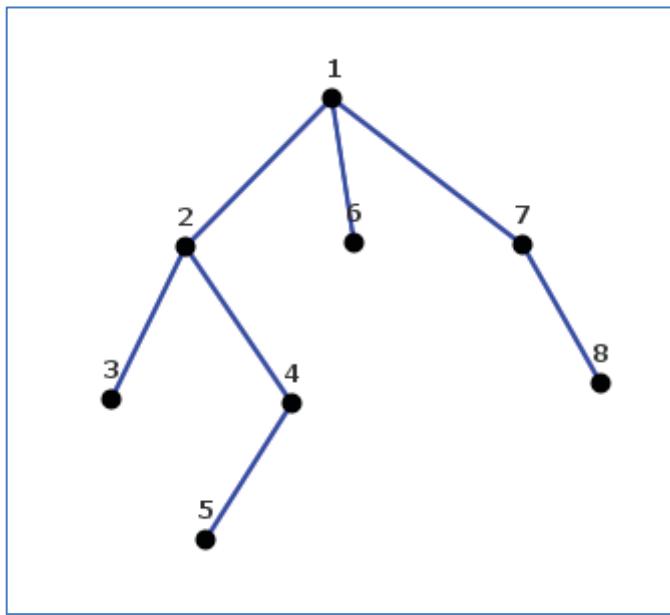


Figura 10. Ejemplo de árbol de 8 vértices

Una de las formas de representación de la información, como estructura de datos, más utilizada es el árbol. Un árbol consiste en un nodo inicial, llamado **raíz**, del que parten un cierto número de aristas, denominadas **ramas**, cada una hasta un nuevo vértice. Los vértices extremos de una arista se llaman también vértice **padre**, aquel que se representa más cerca de la raíz, y el otro vértice **hijo**. A su vez, estos vértices hijos pueden presentar ramas hacia nuevos nodos hijos, que deberán ser siempre nodos diferentes, ya que un árbol no puede presentar ciclos. Por último, aquellos vértices que no presentan ramas hacia ningún hijo, se denominan **hojas**.

2.3.7.5 Grafo bipartito

Los **grafos bipartitos** son aquellos que admiten una partición de sus vértices en dos conjuntos disjuntos $V = X \cup Y$, de manera que las aristas tienen un extremo en cada uno de estos conjuntos (van de vértices en X a vértices en Y). Es el caso de los ciclos C_{2n} de longitud par, de los bosques, o el grafo de la siguiente figura, por ejemplo.

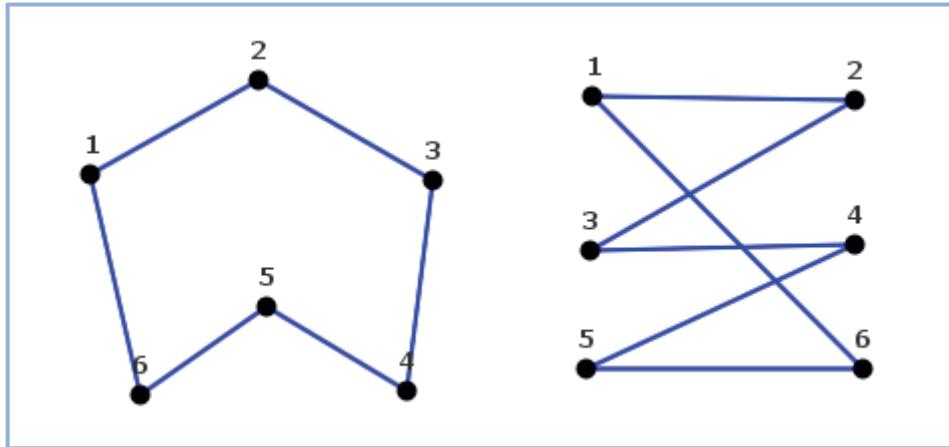


Figura 11. Dos realizaciones del mismo grafo bipartito

Los grafos bipartitos completos $K_{m,n}$ son aquellos de $n + m$ vértices y mn aristas que admiten una partición de sus vértices en sendos conjuntos X e Y de m y n vértices, respectivamente, de manera que cada uno de los m vértices de X es adyacente a todos y cada uno de los n vértices de Y .

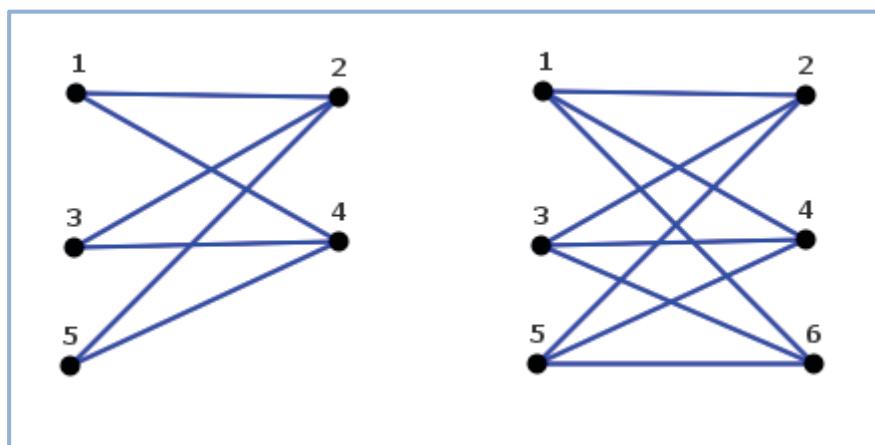


Figura 12. Grafos $K_{2,3}$ y $K_{3,3}$

3 ALGORITMOS IMPLEMENTADOS

3.1 DFS (Depth-First Search)

3.1.1 Descripción

El algoritmo **DFS** es una forma sistemática de encontrar todos los vértices alcanzables de un grafo desde un vértice de origen. En el siglo XIX, el matemático francés **Charles Pierre Tremaux** utilizó una versión de este algoritmo como una estrategia para recorrer laberintos.

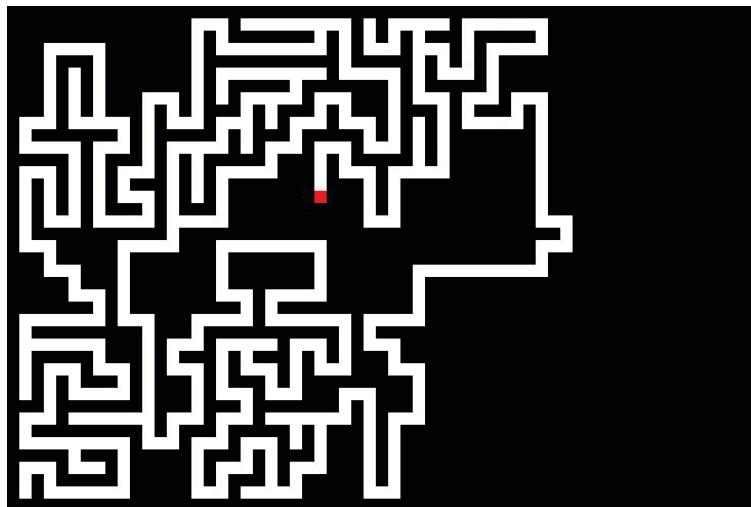


Figura 13. Ejemplo de laberinto

Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

Si G es un grafo un grafo conexo, el algoritmo de búsqueda en profundidad obtiene un árbol **recubridor** de G . Se trata de un grafo en el que aparecen todos los vértices de G , pero no todas sus aristas. El árbol recubridor no es único, depende del vértice de partida.

3.1.2 Estrategia

Existen diferentes formas de implementar este algoritmo. Nosotros hemos escogido la siguiente.

Se utilizan tres posibles colores para representar el estado de un vértice:

- Vértice sin explorar
- Vértice en exploración
- Vértice finalizado

Inicialmente todos los vértices están sin explorar. Una vez determinado un vértice inicial, se ejecuta de la siguiente manera:

1. Se marca el vértice actual u como vértice en exploración (gris).
2. Para cada arista uv , donde u está sin explorar (blanco), ejecutar DFS sobre u de forma recursiva.
3. Marcar el vértice u como finalizado (negro) y regresar a su predecesor (*backtracking*), repitiendo el punto 2.

Distinguiremos dos tipos de aristas:

- Arista del árbol: si nos guía hacia un vértice sin explorar (blanco).
- Arista de retroceso: si nos guía hacia un vértice que está siendo explorado o ya ha sido finalizado (gris o negro).

En la siguiente sucesión de imágenes se puede observar la aplicación de búsqueda en profundidad para recorrer un grafo, partiendo del vértice 1. Se van explorando sucesivamente los vértices hasta llegar al vértice 3, que no tiene más aristas que la que le une con el vértice 5, ya recorrida. Por tanto queda finalizado y se regresa a éste (*backtracking*). El vértice 5 tiene por recorrer la arista que le une con el vértice 4, ya explorado, por lo que estamos ante una arista de retroceso. Una vez finalizado, retrocedemos al vértice 2 y, a continuación, al 4 y al 1. Al ser éste el vértice inicial y estar ya todos los vértices finalizados, se termina la ejecución.

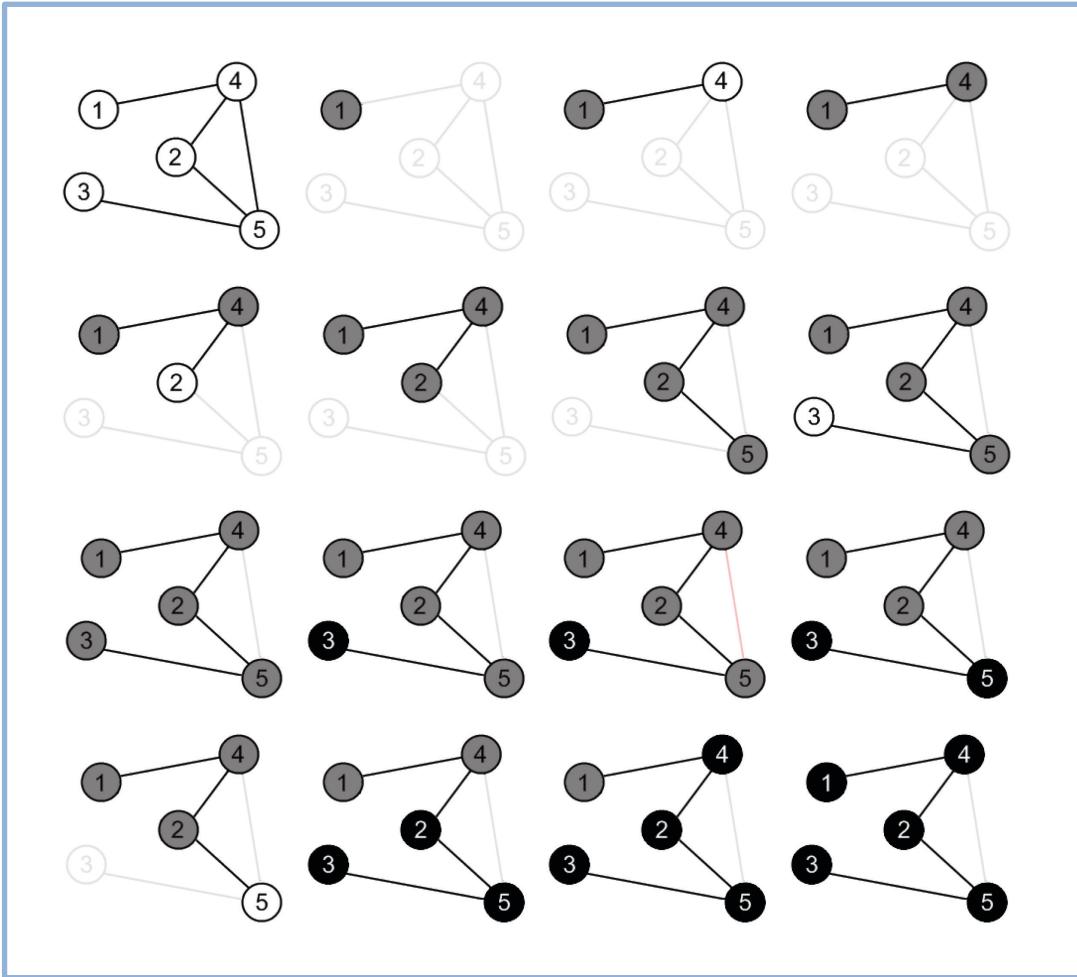


Figura 14. Secuencia de ejecución DFS

Para grafos dirigidos el algoritmo funciona de forma similar, pero considerando únicamente las aristas salientes de cada vértice. Aparece un concepto nuevo, el de predecesor o ancestro de un vértice, que es el recorrido inmediatamente antes del vértice (sucesor o hijo). Además, tendremos en cuenta el orden de búsqueda, es decir, el orden en que los vértices van siendo explorados ($d[v]$).

En este caso podremos distinguir hasta cuatro tipos diferentes de aristas:

- Arista del árbol: si nos guía hacia un vértice sin explorar (blanco).
- Arista de retroceso: si nos guía hacia un vértice que está siendo explorado (gris).

Si una arista nos guía hacia un vértice finalizado (negro), puede ser de dos tipos:

- Arista cross: si el vértice al que nos guía tiene un orden de búsqueda anterior al vértice de origen de la arista ($d[u] < d[v]$).
- Arista forward: si el vértice al que nos guía tiene un orden de búsqueda posterior al vértice de origen de la arista ($d[v] < d[u]$).

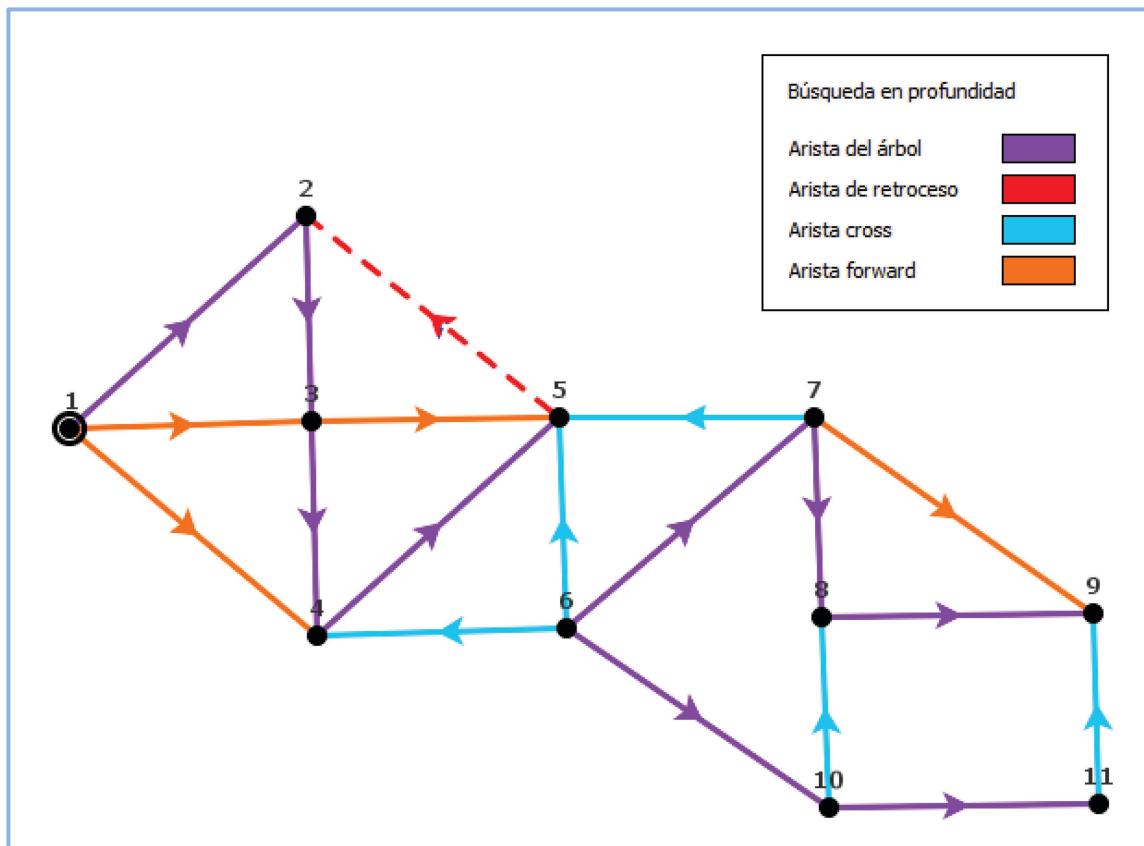


Figura 15. Ejecución DFS finalizada sobre un digrafo

3.2 BFS (Breadth-First Search)

3.2.1 Descripción

El algoritmo **BFS** es una forma de encontrar todos los vértices alcanzables de un grafo partiendo de un vértice origen dado. Como en el algoritmo de búsqueda en profundidad, BFS recorre una componente conexa de un grafo y define un árbol de expansión.

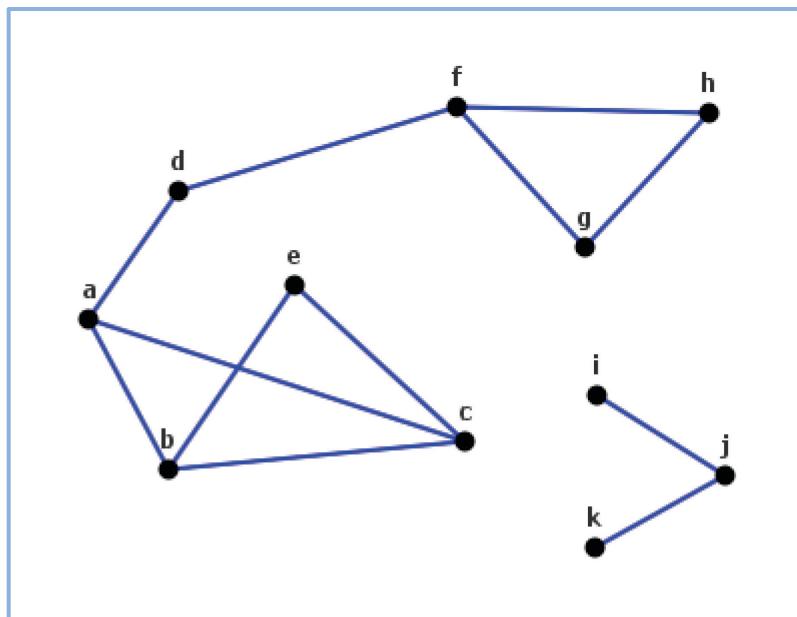


Figura 16. Ejemplo de grafo sobre el que aplicaremos BFS

Intuitivamente, la idea básica de la búsqueda en anchura es la siguiente: lanzar una ola desde el origen s . La ola golpea a todos los vértices situados a una distancia de una arista de s . Desde allí, la ola golpea a todos los vértices situados a una distancia de dos aristas de s , y así sucesivamente.

Este algoritmo de grafos es muy útil en diversos problemas de programación. Por ejemplo halla la ruta más corta entre dos vértices cuando el peso entre todos los nodos es 1, cuando se requiere llegar con un movimiento de caballo de un punto a otro con el menor número de pasos, o para salir de un laberinto con el menor número de pasos, etc.

3.2.2 Estrategia

Utilizaremos una cola, o pila FIFO (First In First Out, el primero en entrar es el primero en salir) donde iremos guardando vértices. El algoritmo, de forma informal, funcionaría de la siguiente manera:

1. Metemos en la cola el vértice origen
2. Sacamos el primer nodo de la cola y lo examinamos
 - a. Si el elemento es el nodo que buscamos, hemos encontrado la solución.
 - b. Si no, metemos en la cola sus vértices vecinos que no hayan sido “descubiertos”.
3. Si la cola está vacía todos los vértices del grafo han sido examinados. Hemos terminado sin encontrar la solución.
4. Si la cola no está vacía repetiríamos el proceso desde el paso 2.

Paso	Estado de la cola
0	
1	a
2	b c d
3	c d e
4	d e
5	e f
6	f
7	h i
8	i
9	

En la figura siguiente se muestra el grafo del ejemplo anterior tras la ejecución del algoritmo. En verde se muestran las aristas del árbol, y en rojo discontinuo aquéllas no del árbol. Se ha detenido la ejecución del algoritmo al recorrer la componente conexa a la que pertenecía el vértice inicial *a*.

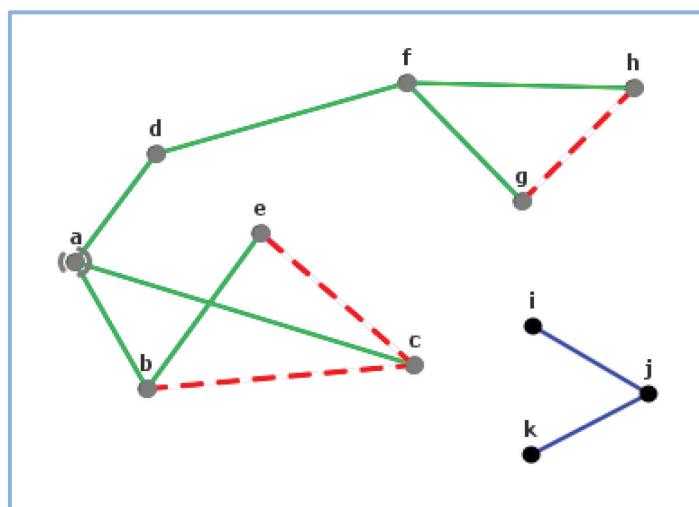


Figura 17. Ejemplo anterior tras la ejecución de BFS

Una de las funcionalidades que ofrece la aplicación dentro de la ejecución de este algoritmo es la posibilidad de generar el árbol de búsqueda resultante, mostrando los vértices en niveles. En la figura siguiente se puede ver un ejemplo.

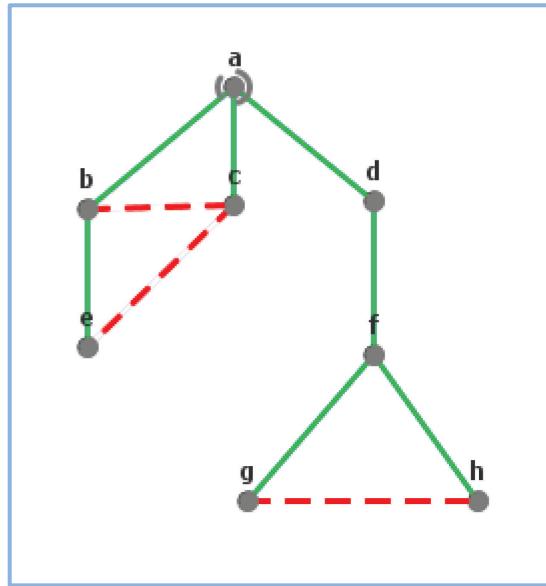


Figura 18. Árbol de búsqueda generado por BFS

3.2.3 Aplicaciones

La búsqueda en anchura puede ser utilizada para solucionar multitud de problemas relacionados con la Teoría de Grafos, además de servir para determinar el camino más corto entre dos nodos u y v de un grafo. Algunos de ellos podrían ser:

- **Encontrar todos los nodos de una componente conexa.**

Si permitimos la aplicación del algoritmo sobre grafos no conexos, cada vez que comenzemos desde un vértice nuevo es porque los vértices que quedan sin visitar no son alcanzables desde los vértices visitados. Entonces lo que estamos encontrando son las diferentes componentes conexas del grafo.

Si ponemos un contador que se incremente luego de la operación elegir un vértice no visitado, lo que obtenemos es la cantidad de componentes conexas del grafo (y con algunas leves modificaciones podemos obtener las componentes en sí).

- **Obtener la distancia de cada uno de los nodos de un grafo al vértice inicial.**

Si partimos desde un vértice v , modificando levemente el código podemos calcular para cada vértice su distancia a v (en saltos, teniendo en cuenta que los pesos de las aristas son siempre 1). Inicialmente todos los vértices tienen rótulo ∞ . Marcamos el vértice v con distancia 0 y luego, al visitar los vecinos de cada vértice w que aún tienen rótulo ∞ , los rotulamos con el rótulo de w más 1 . Al final, cada vértice tendrá marcada la distancia al vértice origen.

- **Probar que un grafo sea bipartito**

Partiendo de un grafo no necesariamente conexo, utilizamos una idea parecida a la anterior en cada componente conexa: rotulamos el vértice v con 0 y luego, al visitar los vecinos de cada vértice w que aún no tienen rótulo, los rotulamos con 1 menos el rótulo de w . De esta manera queda determinada la partición, de ser bipartito el grafo. Queda entonces el verificar que ningún par de vértices con el mismo rótulo sean adyacentes (ese chequeo se puede ir haciendo en cada paso).

3.2.4 Ejemplo práctico: Jugar con ruedas

En este problema vamos a considerar cuatro ruedas alineadas que marcan dígitos del 0 al 9, las cuales están dentro de una caja que muestra únicamente el dígito de la parte superior. En la parte inferior de cada rueda hay dos botones, pulsando cada uno de los cuales haremos que la rueda en cuestión gire en un sentido u otro, mostrando el siguiente dígito. Veamos la siguiente figura para ilustrar el problema:

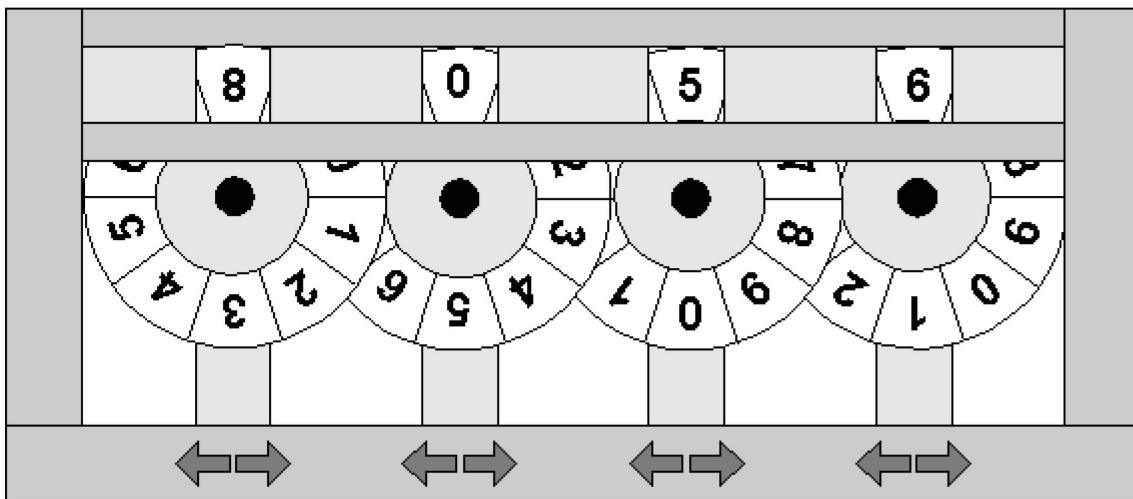


Figura 19. Dibujo ilustrativo del ejemplo

El juego comienza con una secuencia determinada de dígitos, y se trata de alcanzar una secuencia objetivo, utilizando los pulsadores de cada rueda. Sólo se puede utilizar un pulsador cada vez. Al mismo tiempo, existen una serie de combinaciones prohibidas, es decir, por las que no podremos *pasar* para alcanzar nuestro objetivo.

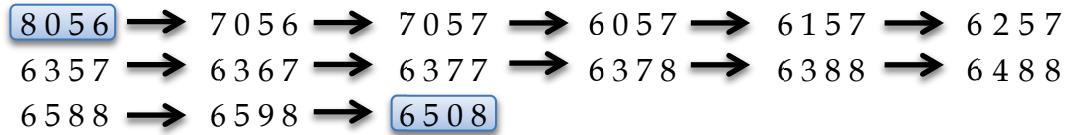
Podemos modelizar el problema como una búsqueda en anchura, en la que las posibles combinaciones son los estados (vértices). El vértice origen es la combinación inicial. Las combinaciones por los que se puede pasar para llegar al vértice destino estarían determinadas por las aristas que unen los vértices, mientras que las combinaciones prohibidas representarían los vértices inalcanzables.

Por ejemplo, desde 8056 puedo ir a 8045, pero no a 3256.

Si establecemos 8 0 5 6 como la combinación inicial y 6 5 0 8 como la combinación objetivo, y establecemos las siguientes combinaciones como prohibidas:

8 0 5 7
8 0 4 7
5 5 0 8
7 5 0 8
6 4 0 8

Podemos alcanzar la combinación objetivo en un mínimo de 14 pasos. Esto quiere decir que nuestro árbol de búsqueda tendría 14 niveles. Una posible solución, sin utilizar combinaciones prohibidas, podría ser:



3.3 Problema del camino mínimo

En teoría de grafos, el problema del camino mínimo es el de encontrar un camino entre dos vértices de un grafo en el que la suma de los pesos (o distancias) de las aristas que componen dicho camino es mínimo.

Formalmente, dado un grafo ponderado (que es un conjunto V de vértices, un conjunto E de aristas y una función de variable real ponderada $f: E \rightarrow \mathbb{R}$) y un elemento $v \in V$ encuentra un camino P de v a $v' \in V$, tal que:

$$\sum_{p \in P} f(p)$$

es el mínimo entre todos los caminos que conectan v y v' .

El problema es también conocido como el problema de los caminos más cortos entre dos nodos, para diferenciarlo de la siguiente generalización:

- **El problema de los caminos más cortos desde un origen** en el cual tenemos que encontrar los caminos más cortos de un vértice origen v a todos los demás vértices del grafo.
- **El problema de los caminos más cortos con un destino** en el cual tenemos que encontrar los caminos más cortos desde todos los vértices del grafo a un único vértice destino, esto puede ser reducido al problema anterior invirtiendo el orden.
- **El problema de los caminos más cortos entre todos los pares de vértices**, el cual tenemos que encontrar los caminos más cortos entre cada par de vértices (v, v') en el grafo.

3.3.1 Algoritmos

Los siguientes algoritmos son los más importantes para resolver este problema:

- **Algoritmo de Dijkstra**, resuelve el problema de los caminos más cortos desde un único vértice origen hasta todos los otros vértices del grafo.
- **Algoritmo de Bellman - Ford**, resuelve el problema de los caminos más cortos desde un origen si la ponderación de las aristas es negativa.
- **Algoritmo de Búsqueda A***, resuelve el problema de los caminos más cortos entre un par de vértices usando la heurística para intentar agilizar la búsqueda.
- **Algoritmo de Floyd - Warshall**, resuelve el problema de los caminos más cortos entre todos los vértices.
- **Algoritmo de Johnson**, resuelve el problema de los caminos más cortos entre todos los vértices y puede ser más rápido que el de Floyd - Warshall en grafos de baja densidad.
- **Teoría perturbacional**, encuentra en el peor de los casos el camino más corto a nivel local.

3.3.2 Algoritmo de Dijkstra

El algoritmo de Dijkstra debe su nombre a su creador, **Edsger Dijkstra**, quien lo concibió en 1956 (fue posteriormente publicado en 1959). Se trata de un algoritmo de búsqueda sobre grafos que soluciona el problema del camino mínimo (de ahí que también sea conocido como **algoritmo de caminos mínimos**) en grafos de aristas con pesos no negativos.

Dado un **grafo conexo** y determinando un vértice de inicio, el algoritmo encuentra el camino de menor coste entre dicho vértice y los restantes (también puede ser usado para determinar el camino mínimo entre dos vértices). Por ejemplo, si los vértices de un grafo representan ciudades y los costes de las aristas la distancia por carretera entre las dos ciudades que une, el algoritmo puede ser usado para determinar la ruta más corta desde una ciudad a las otras.

El algoritmo original no se sirve de colas de prioridad (estructuras de datos en las que los elementos se atienden en el orden indicado por una prioridad asociado a cada uno), teniendo una complejidad de $O(|V|^2)$. La implementación sobre una cola de prioridad llegaría más tarde, y se debe a Fredman y Tarjan (1984). Más concretamente, utiliza un heap de Fibonacci, y consigue rebajar el tiempo de ejecución a $O(|E| + |V|\log |V|)$, siendo E el conjunto de aristas y V el de vértices del grafo (edges y vertex, respectivamente, en inglés).

Este algoritmo se usa bastante en redes de computadores, los nodos corresponden a routers y las aristas entre ellos las conexiones, a cada conexión se le asigna un costo (distancia) y de esta manera algunos protocolos de enrutamiento usan el algoritmo de Dijkstra para encontrar la mejor ruta entre nodos. También es utilizado en todos aquellos sistemas que nos indican las rutas más cortas entre dos puntos (GPS, Google Maps, etc.)

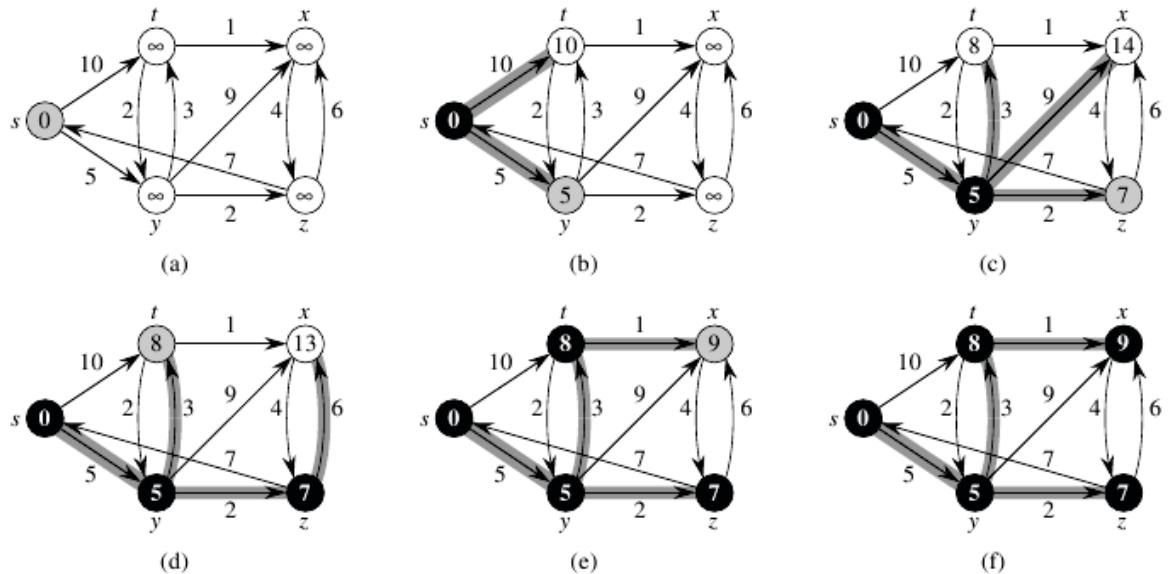


Figura 20. Ejemplo de ejecución del algoritmo de Dijkstra

3.3.3 Estrategia

Tomemos un grafo conexo y ponderado (con pesos no negativos). Partiendo de un vértice inicial, establecemos todas las distancias de dicho vértice al resto en infinito (se establecen al peor caso, el de máximo coste, pues son desconocidas en un principio). Estableceremos como el nodo de inicio el actual.

Recorreremos todos los vértices adyacentes al actual, excepto aquéllos ya marcados:

- Si la distancia guardada entre el vértice actual y su adyacente es mayor que el peso de la arista que las une, sumada a la distancia entre el vértice de origen y el actual, ya calculada, entonces la distancia guardada se sustituye por ésta última, al ser inferior.
- Repetiremos para el resto de adyacentes no marcados. Una vez recorridos todos, marcaremos como terminado el nodo actual.
- El siguiente nodo actual será aquél no marcado cuya distancia guardada sea inferior, y repetiremos el proceso mientras queden nodos por marcar.

Al final, tendremos las distancias mínimas desde el vértice de origen al resto de vértices del grafo.

3.4 Bloques

Los subgrafos conexos sin vértices de corte de un grafo proporcionan una descomposición útil de un grafo. Un bloque de un grafo G es un subgrafo conexo maximal de G sin vértices de corte. Si G mismo es conexo y no tiene vértices de corte, entonces G es un bloque.

Ejemplo. Si H es un bloque de G , entonces H como grafo no tiene vértices de corte, pero algunos de sus vértices pueden ser vértices de corte de G .

3.4.1 Intersección de bloques

Proposición. Dos bloques en un grafo pueden tener como mucho un vértice en común.

Demostración. Supongamos que dos bloques B_1 y B_2 de G tienen al menos dos vértices en común. Probamos que $B_1 \cup B_2$ es un subgrafo conexo sin vértices de cortes, lo que contradice la maximalidad de B_1 y B_2 .

Si eliminamos un vértice v de B_i ($i \in \{1,2\}$), lo que queda de B_i es conexo, por lo que hay un camino desde cualquiera de los vértices restantes de B_i a todos los vértices que queden en $B_1 \cap B_2$. Como $|B_1 \cap B_2| > 1$, al eliminar v queda por lo menos un vértice $w \in B_1 \cap B_2$. Así, al eliminar v , mantenemos los caminos desde cada uno de los vértices de $B_1 \cup B_2 - \{v\}$ a w , por lo que $B_1 \cup B_2$ no puede ser desconectado eliminando sólo un vértice.

3.4.2 Algoritmo de bloques

El algoritmo de bloques de **Jungnickel** (Blockcut en el original) es una adaptación del algoritmo DFS de **Robert Tarjan**, enunciado en el año 72, y que a su vez es una variación ligeramente más eficiente del algoritmo de **Kosaraju**.

Permite hacer una partición de un grafo en los bloques de que está formado. Además, encuentra los puntos de corte del grafo original. Gracias a **Gregorio Hernández Peñalver**, tutor del presente proyecto, se ha llevado a cabo una pequeña adaptación para obtener además los puentes.

El algoritmo hace uso de lo que se ha dado en llamar doble etiquetado, una forma de ir etiquetando cada vértice del grafo mediante dos valores:

- $Nr(v)$ es el orden de búsqueda del vértice v
- $L(v)$ es una función auxiliar que se define como el mínimo valor de $Nr(u)$, para todo vértice u accesible desde v .

Inicialmente, establecemos el doble etiquetado para todos los vértices en la dupla $(0,0)$, y estableceremos el orden de búsqueda a 0 y el vértice de origen como el actual. Además, se hace uso de una pila, inicialmente vacía.

1. Mientras el vértice actual tenga aristas por recorrer (aristas incidentes en vértices vecinos), se escoge una arista, se marca como visitada y se procede de la siguiente manera:

- a) Si el vértice vecino w no ha sido accedido aún ($Nr(w) = 0$)
 - Se incrementa en uno el orden de búsqueda actual ($Nr = Nr + 1$).
 - Se establece el doble etiquetado del vértice ($Nr(w)$ y $L(w)$) como el orden de búsqueda al actual (Nr).
 - Guardamos el vecino en la pila.
 - Establecemos el vecino w como el siguiente vértice activo.
- b) Si el vértice vecino w ya ha sido accedido anteriormente, estamos por tanto ante una arista de retroceso, y se actualizará el valor de función L del vértice activo de la siguiente forma:

$$L(v) \leftarrow \min \{L(v), Nr(w)\}$$

2. Si no tiene aristas por recorrer, cogemos su predecesor.
- a) Si su predecesor **no** es el vértice de inicio (raíz del árbol), procedemos de la siguiente manera:
 - Si el valor de L en el vértice actual v es menor que el orden de búsqueda de su predecesor, establecemos el valor de la función L de éste como el menor de los valores de L en uno y otro:

$$L(p(v)) \leftarrow \min \{L(p(v)), L(v)\}$$

- Si la condición anterior no se cumple, el predecesor del vértice actual v es un punto de corte del grafo, y formamos un nuevo bloque con los vértices que hay insertados por encima del v en la pila, incluyéndole a él (los extraemos previamente de la pila), y al bloque le agregamos $p(v)$.

Además, si $L(v)$ es estrictamente mayor que el orden de búsqueda de su predecesor, la arista que los une es una arista puente.

- b) Si su predecesor **sí** es el vértice de inicio, comprobamos si tiene aristas pendientes de recorrer, en cuyo caso sería un vértice de corte y tendríamos un nuevo bloque, formado con los vértices que hay insertados por encima del v en la pila, incluyéndole a él (los extraemos previamente de la pila), y al bloque le agregamos $p(v)$, es decir, la raíz del árbol.

Además, si $L(v)$ es estrictamente mayor que el orden de búsqueda de su predecesor, la arista que los une es una arista puente.

En ambos casos, estableceremos al predecesor del vértice activo v como el siguiente vértice activo.

El proceso continuará ejecutando el punto 1 o el 2 hasta que el predecesor del vértice activo sea 0 y éste no tenga aristas que le unan con sus vecinos pendientes de recorrer.

NOTA: Cuando tratemos con digrafos es importante recalcar que un vértice es vecino de otro si existe una arista entre ambos que tiene como vértice de destino al primero de ellos.

4 DISEÑO DE LA APLICACIÓN

4.1 Lenguaje de programación

Para el desarrollo de la aplicación se ha utilizado el lenguaje de programación Java. Algunas de las características más importantes por las que se ha elegido este lenguaje son las siguientes:

- Es orientado a objetos. Por lo tanto, al igual que otros lenguajes orientados a objetos, tiene la ventaja de que facilita la reutilización y la extensión del código.
- Es multiplataforma, es decir, un programa escrito en Java se pueda ejecutar en múltiples plataformas. El compilador de Java produce un código de bytes que puede ser ejecutado por un intérprete denominado *máquina virtual de Java*; hoy en día, casi todas las compañías de sistemas operativos y de navegadores han implementado máquinas virtuales según las especificaciones publicadas por Sun Microsystems, propietario de Java, para que sean compatibles con el lenguaje Java.
- Evita muchas preocupaciones al programador. La administración de la memoria es automática; el programador no tiene que ocuparse de la liberación de memoria reservada dinámicamente, ya que un *recolector de basura* libera la memoria asignada a un objeto cuando ya no existe ninguna referencia a ese objeto. Y en el proceso de compilación se realizan numerosas comprobaciones que permiten eliminar muchos errores posteriores.
- Entorno de desarrollo gratuito. Sun Microsystems proporciona un entorno de desarrollo Java, *Java Development Kit (JDK)*, de forma gratuita.
- El programador no parte de cero, ya que dispone de una API (Interfaz de Programación de Aplicaciones) que ofrece numerosas clases e interfaces, agrupadas en distintos paquetes, listas para que el programador las utilice en sus propias aplicaciones y que abarcan desde los objetos básicos hasta la generación de XML y el acceso a bases de datos.

4.2 Diseño de Alto Nivel

El objetivo de este apartado es explicar el diseño empleado en la construcción de la aplicación. Como su nombre indica, no consiste en un diseño exhaustivo de cada una de las líneas de código del programa, sino una visión general del proceso realizado. Este proceso viene determinado por los siguientes puntos:

- Definición de los límites del sistema.
- Diagrama de casos de uso.
- Descripción de los casos de uso en formato extendido

Los objetivos de esta memoria no incluyen el realizar una documentación perfeccionista de Ingeniería del Software, por lo que describiremos el proceso de manera que se pueda entender fácilmente y sin estar sujeto a las rigurosas normas de nomenclatura que precisan dichos documentos.

4.2.1 Definición de los límites del sistema

En este apartado se delimitan las funcionalidades que realiza la aplicación. Todo lo que no está definido aquí no lo realiza el sistema.

La herramienta permite crear, editar y borrar grafos, sobre los que se pueden ejecutar algoritmos, tanto de coloración (secuencial, Welsh, Matula, Brelaz e Independencia), como de búsqueda (DFS, BFS, conectividad y Dijkstra).

Además, la aplicación permite obtener datos estadísticos, tanto del grafo como de los algoritmos aplicados.

Como soporte a su utilización, la aplicación permite el guardado de un grafo, incluyendo la posibilidad de almacenar el estado de ejecución de un algoritmo, con el fin de continuar su ejecución en otro momento. El sistema permite generar grafos de forma tanto automática como manual; el usuario puede agregar o borrar aristas y vértices. Como combinación de ambos, una vez generado un grafo, el sistema también permite modificarlo de forma

manual. Para mejorar la visibilidad y la maniobrabilidad, la herramienta permite mover los vértices del grafo sobre el panel de dibujo.

Siempre que el usuario quiera, puede visualizar tanto la matriz de adyacencia como la de ponderación del grafo generado.

El sistema ha incluido 10 colores diferentes, por defecto, para realizar las coloraciones. El usuario podrá modificarlos así como añadir hasta un máximo de otros 10 colores diferentes si lo considerase necesario.

La aplicación es multilenguaje, disponible en español e inglés.

4.2.2 Diagramas de casos de uso

Los casos de uso permiten describir el comportamiento de un sistema desde el punto de vista del usuario basándose en un conjunto de acciones y reacciones. Es por lo tanto una técnica que permite capturar los requisitos funcionales del sistema. De esta forma queda delimitado el alcance del sistema y cuál es su relación con el entorno.

En estos diagramas, el sistema queda reducido a una “caja negra”, ya que no interesa cómo lleva a cabo sus funciones, sino simplemente qué acciones visibles desde el exterior son las que realiza.

Los casos de uso están basados en lenguaje natural, lo que los hace accesibles a cualquier usuario. Además, aquellos casos de uso que resulten muy complejos pueden descomponerse en nuevos casos de uso de un nivel inferior, hasta llegar a un nivel tal que resulten fáciles de analizar.

Los diagramas de casos de uso están formados por tres elementos principales:

- **Actores.** Los actores son los participantes de los casos de uso. Se corresponden con los usuarios que interactúan con el sistema. Pueden ser humanos, dispositivos externos que interactúen con el sistema, o incluso temporizadores que envíen eventos al mismo. Un actor se caracteriza por la forma de interaccionar con el sistema, por lo que un mismo usuario puede ejercer de varios actores, y un actor puede representar a varios usuarios.

- **Casos de uso.** Son los escenarios de interacción de los actores. Representan el comportamiento del sistema en relación con los usuarios. De esta forma, un caso de uso define la secuencia de interacciones entre uno o más usuarios y el sistema.
- **Relaciones.** Representan el flujo de información intercambiada entre los actores y los casos de uso, o entre diferentes casos de uso. Normalmente, se emplean para que un caso de uso obtenga la información necesaria para llevar a cabo alguna acción, o para que el proceso proporcione algún resultado. Estas relaciones pueden ser unidireccionales o bidireccionales.

Los diagramas de casos de uso se clasifican en diferentes niveles, en función del grado de detalle con el que se represente el funcionamiento del sistema. De esta forma, los diagramas de nivel 0 o contexto representan el sistema completo con un nivel de detalle muy bajo, mientras que al aumentar el nivel, el grado de detalle va incrementándose.

A continuación muestra la notación empleada para la representación de los distintos elementos que forman los diagramas de casos de uso.



Los actores son representados mediante figuras de *hombre de palo*, con su correspondiente nombre debajo de la figura.

Los casos de uso, o procesos, se representan mediante una elipse, con su nombre correspondiente debajo de la misma.

Las relaciones se representan mediante flechas que unen los casos de uso, o el caso de uso y el actor, entre los que existe un flujo de información.

A continuación presentamos el diagrama de casos de uso de nivel 0:



Figura 21. Diagrama de nivel 0

Debido a su mayor complejidad, se ha decidido descomponer los siguientes casos de uso en subniveles: GenerarGrafo, EstablecerConfiguración y EjecutarAlgoritmo.

4.2.2.1 Generar Grafo

El caso de uso GenerarGrafo se puede descomponer a su vez en tres casos de uso, que se corresponden con la manera en que el usuario puede generar un grafo desde la aplicación:

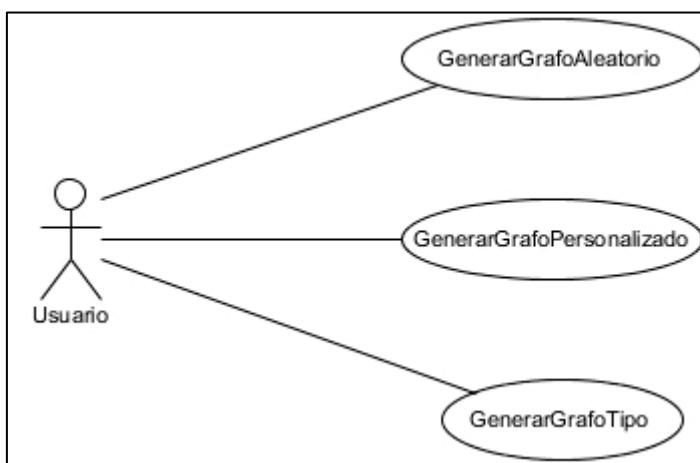


Figura 22. Diagrama de nivel 1: GenerarGrafo

A continuación descomponemos cada uno de estos tres casos de uso, obteniendo los correspondientes diagramas de nivel 2 para cada uno.

GenerarGrafoAleatorio

El usuario introducirá los datos necesarios para generar el grafo (nombre, número de vértices, probabilidad de arista y si es o no dirigido), y el sistema obtendrá un grafo a partir de éstos y lo mostrará de forma gráfica.

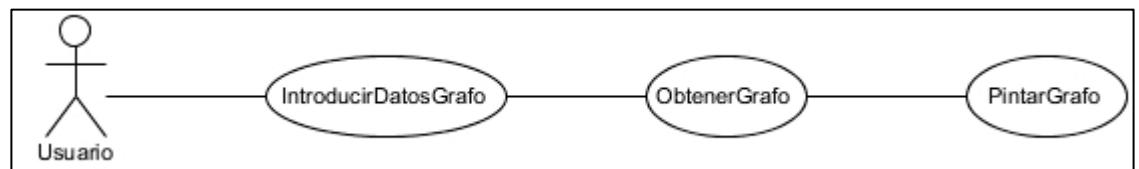


Figura 23. Diagrama de nivel 2: GenerarGrafoAleatorio

GenerarGrafoPersonalizado

El usuario introducirá los datos necesarios para generar el grafo (nombre, número de vértices y si es o no dirigido). A continuación, deberá indicar, para cada vértice, cuáles serán sus vecinos (existirá una arista entre ellos). Dependiendo de si el grafo es o no dirigido, la arista resultante entre dos vértices tendrá o no sentido. El sistema obtendrá un grafo a partir de los datos anteriormente introducidos y lo mostrará de forma gráfica.



Figura 24. Diagrama de nivel 2: GenerarGrafoPersonalizado

GenerarGrafoTipo

Dependiendo del grafo tipo que el usuario desee crear, deberá introducir un determinado tipo de datos u otro (nombre, número de vértices, niveles, nivel de enlace, grado, altura, etc.) para su generación. A continuación el sistema obtendrá un grafo a partir de éstos y lo mostrará de forma gráfica.

Los grafos tipo que el usuario puede generar son:

- Grafo nulo
- Grafo completo: completo k_n , bipartito y tripartito
- Grafo de rejilla: rectangular y triangular
- Grafo platónico: tetraedro, hexaedro, octaedro, icosaedro y dodecaedro
- Grafo enlazado: L_{nr} y L_{nrs}
- Grafo rueda
- Grafo de estrella
- Grafo cubo
- Grafo de Harary

- Grafo de Petersen
- Grafo de Grötzsch
- Grafo de Tutte
- Grafo de Heawood
- Grafo de Herschel

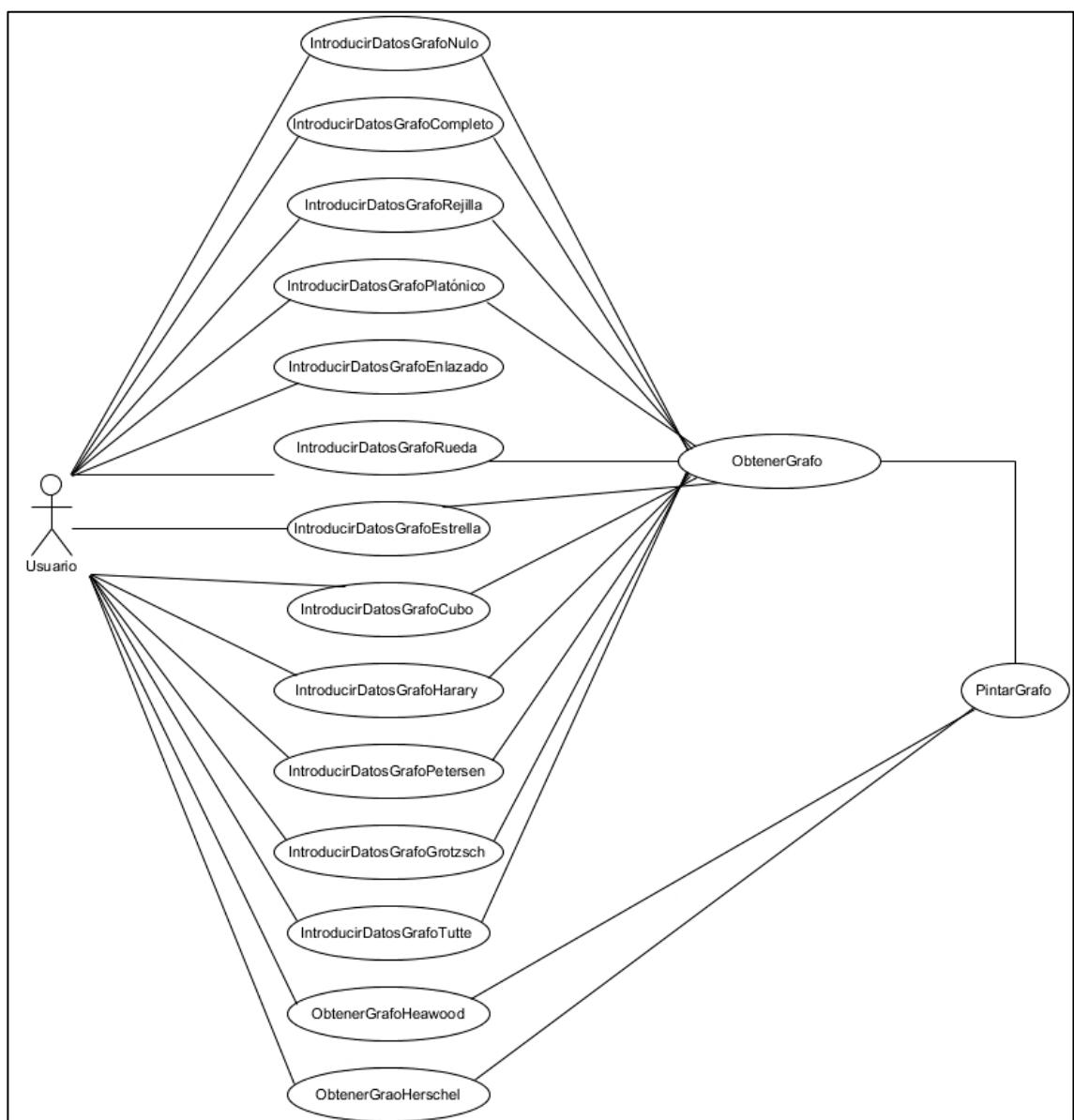


Figura 25. Diagrama de nivel 2: GenerarGrafoTipo

4.2.2.2 EstablecerConfiguración

El caso de uso EstablecerConfiguración lo vamos a descomponer a un nivel inferior para comprender mejor su funcionamiento:

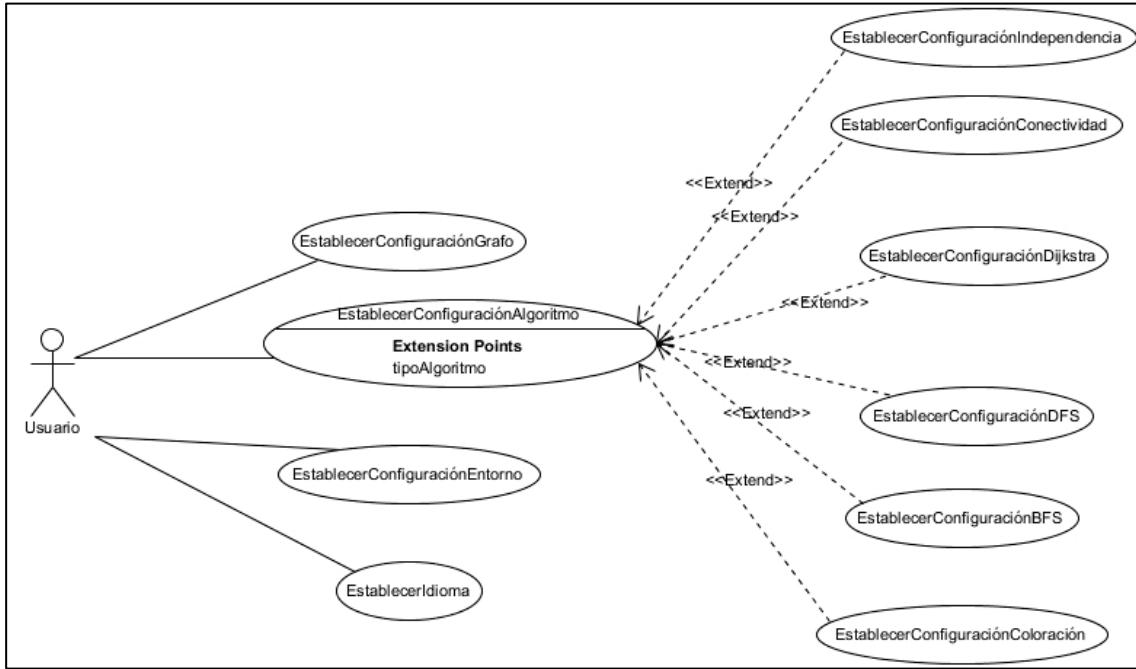


Figura 26. Diagrama de nivel 1: EstablecerConfiguración

Podemos ver que este caso de uso se descompone a su vez en:

- **EstablecerConfiguraciónGrafo**, donde el usuario podrá establecer las opciones características del grafo (color de vértice, color de arista, posición de la flecha para el caso de digrafos, etc.)
- **EstablecerConfiguraciónEntorno**, donde el usuario podrá establecer el número de pasos hacia atrás que el sistema puede dar durante la ejecución de un algoritmo.
- **EstablecerIdioma**, donde el usuario puede modificar el idioma de la aplicación.
- **EstablecerConfiguraciónAlgoritmo**, compuesto a su vez de otros seis casos de uso (que lo extienden), uno por cada algoritmo ejecutable, donde el usuario podrá establecer las opciones características de cada algoritmo.

4.2.2.3 EjecutarAlgoritmo

El caso de uso EstablecerConfiguración lo vamos a descomponer a un nivel inferior para comprender mejor su funcionamiento:

El caso de uso EjecutarAlgoritmo se puede descomponerse en un nivel inferior para una mayor comprensión:

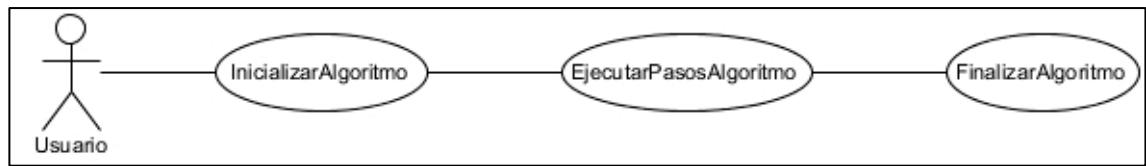


Figura 27. Diagrama de nivel 1: EjecutarAlgoritmo

Como puede apreciarse, el usuario inicializará el algoritmo seleccionado, introduciendo los datos iniciales (si fuesen requeridos). El sistema ejecutará los pasos necesarios hasta que el algoritmo finalice.

4.2.3 Descripción de los casos de uso en formato extendido

En esta sección expondremos los casos de uso con un nivel de detalle mayor, especificando cuáles son las interacciones habituales entre los actores de la aplicación.

El formato que utilizaremos, emplea los siguientes campos:

- **Caso de Uso:** Nombre del caso de uso.
- **Objetivo:** Explicación del caso del objetivo del caso de uso
- **Actor principal:** En este caso el Usuario
- **Precondiciones:** Condiciones que deben cumplirse antes de comenzar el escenario principal de éxito.
- **Escenario principal de éxito:** Pasos detallados de las interacciones entre actores.
- **Garantías de éxito:** Qué debe cumplirse cuando el caso de uso se acaba con éxito.
- **Garantías de fracaso:** Qué debe cumplirse cuando el caso de uso se abandona.

Caso de uso	AbrirGrafo
Objetivo	Crear un nuevo grafo cargado desde fichero
Actor principal	Usuario
Precondiciones	El fichero XML seleccionado debe seguir el formato correcto de descripción de un grafo (puede haber sido generado a partir de la funcionalidad de salvar grafo)
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción Abrir grafo. 2. El sistema pide la ruta de acceso al fichero que contiene los datos del grafo. 3. El usuario indica la ruta 4. El sistema verifica la coherencia del fichero. 5. El sistema genera un nuevo grafo a partir del fichero suministrado. 6. El sistema muestra el grafo pintado sobre el lienzo.
Garantías de éxito	Se habrá creado un nuevo grafo a partir del fichero suministrado y se mostrará pintado sobre el lienzo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	SalvarGrafo
Objetivo	Guardar el grafo, representado sobre el lienzo, en un fichero.
Actor principal	Usuario
Precondiciones	Debe existir un grafo
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción Salvar grafo. 2. El sistema pide la ruta donde se almacenará el fichero que contendrá los datos del grafo. 3. El usuario indica la ruta 4. El sistema serializa el grafo en una cadena XML. 5. El sistema genera un fichero XML con la cadena generada y lo guarda en la ruta especificada.
Garantías de éxito	Se habrá creado un fichero XML que contendrá el grafo serializado.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	CerrarGrafo
Objetivo	Eliminar el grafo representado sobre el lienzo.
Actor principal	Usuario
Precondiciones	Debe existir un grafo
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción Cerrar grafo. 2. El sistema pregunta si se desea salvar el grafo. 3.a. El usuario decide guardar el grafo (ver caso de uso relacionado; continuará en el punto 4 tras la ejecución del caso de uso) 3.b. El usuario no decide guardar el grafo 4. El sistema elimina el grafo de memoria y deja el lienzo en blanco.
Garantías de éxito	Se habrá eliminado el grafo de memoria y el lienzo se mostrará en blanco.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	EditarNodo
Objetivo	Editar las características del vértice (aristas de entrada y salida)
Actor principal	Usuario
Precondiciones	Debe existir un grafo
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona el vértice a editar. 2. El sistema muestra la pantalla de edición de nodos. 3. El usuario añade o elimina vértices vecinos. 4. El sistema añade o elimina aristas del grafo en función de las decisiones tomadas por el usuario, y muestra el grafo resultante sobre el lienzo.
Garantías de éxito	Se habrá modificado el grafo y el lienzo reflejará los cambios.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	ExportarGrafo
Objetivo	Exportar la información del grafo a un fichero Excel.
Actor principal	Usuario
Precondiciones	Debe existir un grafo
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de Exportar grafo. 2. El sistema pide la ruta donde se almacenará el fichero que contendrá los datos del grafo. 3. El usuario indica la ruta 4. El sistema obtiene los datos del grafo. 5. El sistema genera un fichero Excel con los datos y lo guarda en la ruta especificada.
Garantías de éxito	Se habrá creado un fichero Excel que contendrá los datos exportados del grafo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	LimpiarAlgoritmo
Objetivo	Eliminar toda la información relativa al algoritmo que se está ejecutando sobre el grafo, tanto en memoria como sobre el lienzo.
Actor principal	Usuario
Precondiciones	Debe existir un grafo y se debe estar ejecutando un algoritmo.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de Limpiar algoritmos. 2. El sistema elimina los datos relativos al algoritmo. 3. El sistema muestra sobre el lienzo el grafo original.
Garantías de éxito	Se habrá eliminado toda información relativa al algoritmo que se estaba ejecutando y se mostrará el grafo original sobre el lienzo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	CambiarTipoProyecto
Objetivo	Cambiar el tipo de proyecto (búsquedas o coloración) para que el usuario pueda ejecutar otro conjunto de algoritmos.
Actor principal	Usuario
Precondiciones	No debe existir un algoritmo en ejecución.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona otro tipo de algoritmo al actual. 2. El sistema registra la información del nuevo tipo de proyecto a ejecutar.
Garantías de éxito	Se habrá cambiado el tipo de proyecto a ejecutar.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	ObtenerMatrizAdyacencia
Objetivo	Mostrar la matriz de adyacencia del grafo.
Actor principal	Usuario
Precondiciones	Debe existir un grafo.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona de Mostrar matriz de adyacencia. 2. El sistema genera la matriz de adyacencia. 3. El sistema muestra en una nueva ventana la

	matriz de adyacencia del grafo.
Garantías de éxito	Se muestra en una nueva ventana la matriz de adyacencia del grafo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	ObtenerMatrizPonderación
Objetivo	Mostrar la matriz de ponderación del grafo.
Actor principal	Usuario
Precondiciones	Debe existir un grafo.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona de Mostrar matriz de ponderación. 2. El sistema genera la matriz de ponderación. 3. El sistema muestra en una nueva ventana la matriz de ponderación del grafo.
Garantías de éxito	Se muestra en una nueva ventana la matriz de ponderación del grafo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	AñadirVértice
Objetivo	Añadir un nuevo vértice.
Actor principal	Usuario
Precondiciones	Ninguna.
Escenario de éxito	<ol style="list-style-type: none">1. El usuario selecciona la opción de Añadir vértice.2. El usuario selecciona sobre el lienzo la posición del nuevo vértice.3. El sistema añade un nuevo vértice al grafo en la posición indicada y muestra el resultado sobre el lienzo.
Garantías de éxito	Se añade el nuevo vértice en la posición indicada.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	BorrarVértice
Objetivo	Eliminar el vértice indicado.
Actor principal	Usuario
Precondiciones	Debe existir un grafo.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de Borrar vértice. 2. El usuario selecciona sobre el lienzo el vértice a borrar. 3. El sistema elimina todas las aristas, entrantes y salientes, del vértice a borrar. 4. El sistema elimina el vértice a borrar. 5. El sistema muestra sobre el lienzo el grafo resultante.
Garantías de éxito	Se modifica el grafo sin el vértice eliminado y sus aristas entrantes y salientes.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	MoverVértice
Objetivo	Mover el vértice indicado a la posición elegida.
Actor principal	Usuario
Precondiciones	Debe existir un grafo.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de Mover vértice. 2. El usuario selecciona sobre el lienzo el vértice a mover. 3. El usuario arrastra el vértice seleccionado a la posición deseada. 4. El sistema cambia la posición del vértice seleccionado por la nueva. 5. El sistema muestra sobre el lienzo el grafo resultante.
Garantías de éxito	Se modifica el grafo con el vértice seleccionado en la nueva posición.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	AñadirArista
Objetivo	Añadir una nueva arista entre el vértice origen y el de destino.
Actor principal	Usuario
Precondiciones	Debe existir un grafo con al menos dos vértices.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de Añadir arista. 2. El usuario selecciona sobre el lienzo el vértice de origen. 3. El usuario arrastra el puntero del ratón hasta el vértice destino. 4. El sistema añade una nueva arista entre el vértice origen y el de destino del grafo y muestra el resultado sobre el lienzo.
Garantías de éxito	Se añade una nueva arista entre los vértices indicados.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	BorrarArista
Objetivo	Eliminar la arista indicada.
Actor principal	Usuario
Precondiciones	Debe existir un grafo que contenga al menos una arista.
Escenario de éxito	<ol style="list-style-type: none">1. El usuario selecciona la opción de Borrar arista.2. El usuario selecciona sobre el lienzo la arista a borrar.3. El sistema elimina la arista del grafo.4. El sistema muestra sobre el lienzo el grafo resultante.
Garantías de éxito	Se modifica el grafo sin la arista eliminada.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Como hemos visto anteriormente este caso de uso ha sido descompuesto en niveles para mejor comprensión. Se ha tomado la decisión de explicar los casos de uso que conforman el nivel 1: Generar grafo aleatorio, Generar grafo personalizado y Generar grafo tipo.

Caso de uso	GenerarGrafoAleatorio
Objetivo	Crear un grafo.
Actor principal	Usuario
Precondiciones	Ninguna.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de Generar un grafo de forma aleatoria. 2. El sistema muestra una ventana de introducción de datos. 3. El usuario completa la información (número de vértices, probabilidad de arista y si es dirigido). 4. El sistema cierra la ventana y calcula un grafo en base a los datos introducidos por el usuario. 5. El sistema muestra el grafo generado sobre el lienzo.
Garantías de éxito	Se genera un grafo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	GenerarGrafoPersonalizado
Objetivo	Crear un grafo.
Actor principal	Usuario
Precondiciones	Ninguna.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de Generar un grafo de forma personalizada. 2. El sistema muestra una ventana de introducción de datos. 3. El usuario completa la información (número de vértices y si es dirigido). 4. El sistema cierra la ventana y genera internamente un grafo nulo con el número de vértices introducido. 5. El sistema muestra por pantalla una ventana similar a la de edición de nodos (ver caso de uso) 6. El usuario añade vértices vecinos para cada nodo del grafo y pulsa el botón de Aceptar. 7. El sistema cierra la ventana, añadiendo al grafo las aristas conforme a las opciones seleccionadas por el usuario. 8. El sistema muestra el grafo generado sobre el lienzo.
Garantías de éxito	Se genera un grafo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	GenerarGrafoTipo
Objetivo	Crear un grafo (de los predefinidos en el sistema).
Actor principal	Usuario
Precondiciones	Ninguna.
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona el grafo tipo a generar. 2. El sistema muestra una ventana de introducción de datos. 3. El usuario completa la información (dependiendo del grafo tipo se mostrará una información u otra). 4. El sistema cierra la ventana y calcula el grafo tipo en base a los datos introducidos por el usuario. 5. El sistema muestra el grafo generado sobre el lienzo.
Garantías de éxito	Se genera un grafo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	EstablecerConfiguración
Objetivo	Modificar la configuración del sistema
Actor principal	Usuario
Precondiciones	Ninguna
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción Opciones. 2. El sistema muestra la ventana de opciones. 3. El usuario modifica tantas opciones del sistema como desee 4. El sistema almacena la nueva configuración. 5. El sistema aplica la nueva configuración en el entorno de trabajo.
Garantías de éxito	Se habrá modificado la configuración del sistema, según lo especificado por el usuario. Dichos cambios serán visibles en el entorno de trabajo.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

Caso de uso	EjecutarAlgoritmo
Objetivo	Ejecutar un algoritmo sobre un grafo
Actor principal	Usuario
Precondiciones	Debe existir un grafo
Escenario de éxito	<ol style="list-style-type: none"> 1. El usuario selecciona la opción Ejecutar. 2. El sistema muestra la ventana de ejecución de algoritmos en función del tipo de proyecto seleccionado actualmente. 3. El usuario selecciona el algoritmo a ejecutar e introduce los datos necesarios para inicializar el algoritmo. 4. El sistema carga el estado inicial del algoritmo seleccionado. 5. El usuario inicia la ejecución del algoritmo. 6. El sistema ejecuta los pasos indicados por el usuario (dependiendo del tipo de ejecución), modificando el grafo por pantalla, hasta que se cumple la condición de fin del algoritmo.
Garantías de éxito	Se habrá ejecutado el algoritmo seleccionado sobre el grafo existente.
Garantías de fracaso	El estado anterior al caso de uso en el que estaba el sistema.

4.3 Diseño de Bajo Nivel

Para realizar el diseño de bajo nivel utilizaremos **diagramas de clases**, que nos permitirán tener una visión global de los diferentes elementos que conforman la aplicación

4.3.1 Diagrama de clases

El diseño del proyecto se ha realizado mediante una metodología orientada a objetos. Algunas de las ventajas que tiene el uso de esta metodología son la **reutilización** y la posibilidad de realizar una **mayor abstracción durante la etapa de diseño**. El uso de esta metodología permite abordar problemas más complejos facilitando el mantenimiento y evolutivo de los sistemas. Además es una metodología apropiada para desarrollos iterativos como el aplicado en este proyecto.

El desarrollo del proyecto se ha realizado utilizando el lenguaje de programación **Java**. La elección de este lenguaje permite desarrollar siguiendo la metodología **OO** con independencia de la plataforma, y además permite la utilización de sus librerías gráficas (Swing). Se ha optado por seguir el modelo vista-controlador como patrón de diseño.

En los siguientes diagramas de clases se ha incluido la información más relevante, excluyendo las librerías propias de Java, ya que no aportan información adicional.

4.3.1.1 Visión general

En este diagrama se muestra una visión general de la aplicación. A continuación se explican las clases que lo conforman:

- **AIGView:** Extiende de la clase FrameView para la construcción de la ventana principal de la aplicación. Aquí es donde se tratarán todos los eventos del sistema.
- **Canvas:** Extiende de la clase JPanel para construir el lienzo de la aplicación, donde se dibujarán los grafos y los algoritmos que sobre ellos se apliquen.
- **Controlador:** Es la clase principal de la aplicación, encargada de recibir información tanto de los eventos lanzados desde la vista como del canvas, sirviendo de puente entre ellos. Además, sirve para gestionar el modelo de datos. Conectados al controlador se encuentran otras clases, que serán explicadas en sus diagramas correspondientes para una mejor comprensión. Estos subdiagramas son:
 - Grafo
 - Configuración
 - Algoritmo

Podemos comprobar que estas tres clases utilizan tipos enumerados:

- **eBotonPulsado:** utilizado para enumerar los botones de la ejecución de los algoritmos (paso previo, ejecución, pausa y paso siguiente).
- **eAccion:** utilizado para enumerar los botones que son de aplicación directa sobre el lienzo (agregar un vértice o arista, eliminar un vértice o arista).
- **eResultadoPaso:** utilizado para enumerar los posibles valores tomados tras la ejecución interactiva del paso de un algoritmo.

Diseño de la aplicación

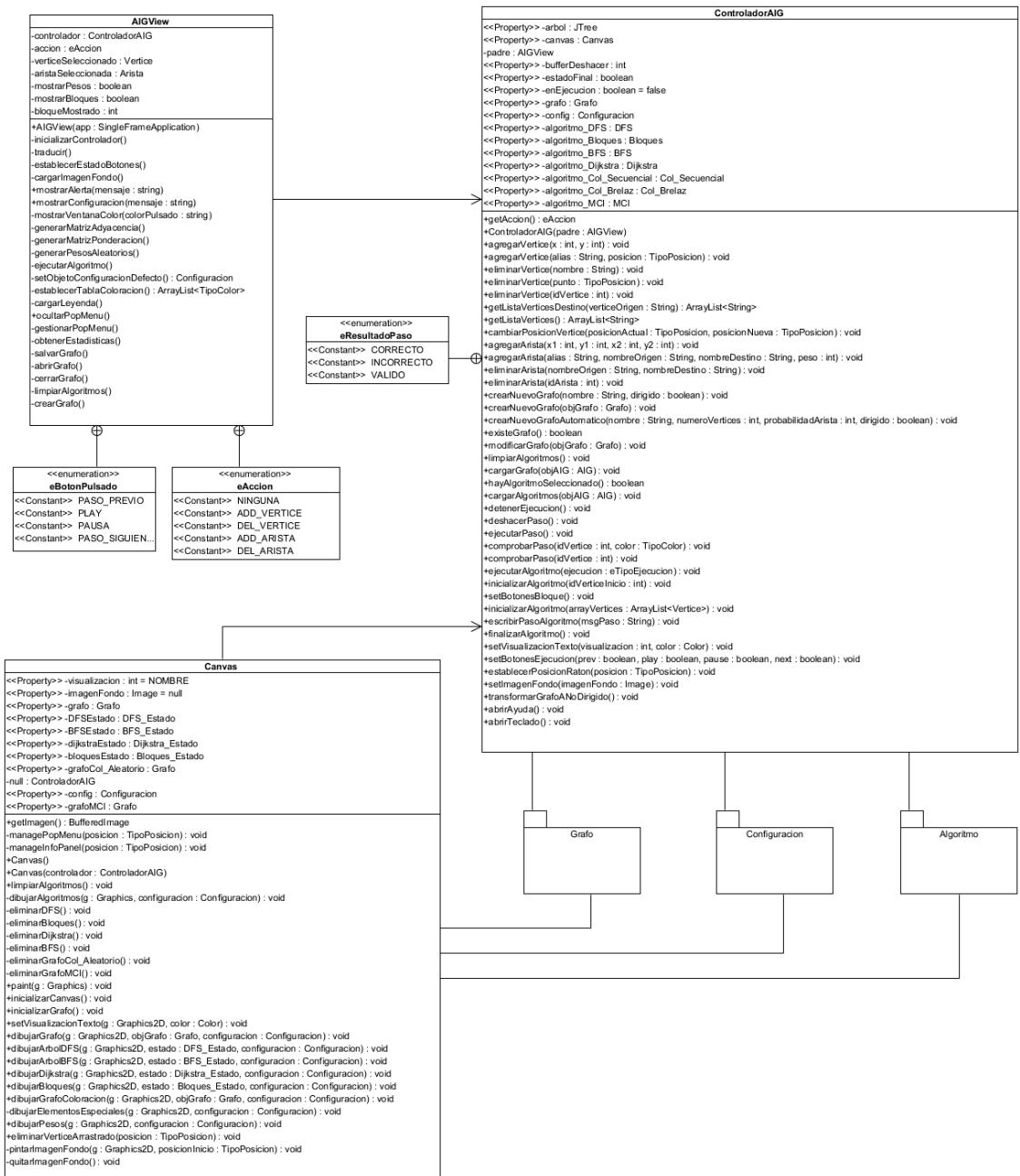


Figura 28. Diagrama de Clases que muestra la Visión Genera del Proyecto

4.3.1.2 Grafo

En este diagrama se explica cómo está formada la clase Grafo y las clases en las que se apoya (Vertice y Arista, principalmente):

- **Grafo:** Clase que sirve para instanciar el objeto que representa a un grafo. Se ha decidido modelizar un grafo mediante una lista con sus vértices y otra con sus aristas. Utiliza un tipo enumerado para representar la forma en que se genera (manual o automática).
- **Vertice:** Clase que sirve para instanciar el objeto que representa un vértice del grafo. En cada objeto tenemos una lista de vecinos (aquellos vértices del grafo a los que se puede ir, unidos a este objeto mediante una arista. Si el grafo no es dirigido, tenemos una lista de aristas. En cambio, si es dirigido, tenemos una lista de aristas de entrada y otra de aristas de salida. Esta clase se relaciona con las clases TipoPosicion y TipoColor.
- **Arista:** Clase que sirve para instanciar el objeto que representa una arista del grafo. Contiene información de los vértices de origen y destino, así como el peso. Se relaciona con la clase TipoColor.
- **TipoPosicion:** Representa una posición sobre el eje cartesiano (x,y) del lienzo.
- **TipoColor:** Representa un color mediante sus componentes RGB.

Diseño de la aplicación



Figura 29. Diagrama de Clases con la clase Grafo (y las clases con las que se relaciona)

4.3.1.3 Algoritmo

Debido a que la aplicación implementa distintos algoritmos, se ha tomado la decisión de separar los algoritmos en bloques para una mejor visión de las clases.

Algoritmos DFS y BFS

- **DFS:** Clase que instancia un objeto que representa al algoritmo DFS. Contiene información relativa tal como vértice de inicio, la estadística de ejecución, el estado actual (representado por una instancia de la clase DFS_Estado) y una lista de estados anteriores, entre otros.
- **DFS_Estado:** Clase que instancia un objeto que representa información de la ejecución de un paso del algoritmo. Entre esta información se encuentra una lista de vértices del árbol que se va formando, los vértices candidatos, las aristas utilizadas, etc. Se apoya en tres clases de tipos enumerados: eTipoVértice, eTipoEstado y eTipoArista.
- **BFS:** Clase que instancia un objeto que representa al algoritmo BFS. Contiene información relativa tal como vértice de inicio, vértice final, la estadística de ejecución, el estado actual (representado por una instancia de la clase BFS_Estado) y una lista de estados anteriores, entre otros.
- **BFS_Estado:** Clase que instancia un objeto que representa información de la ejecución de un paso del algoritmo. Entre esta información se encuentra una lista de vértices candidatos y visitados, las aristas utilizadas y las no del árbol, la distancia de cada vértice al origen, etc. Se apoya en tres clases de tipos enumerados: eEstadoVértice, eEstadoArista y eTipoArista.
- **Estadistica:** Clase que instancia un objeto que representa la información estadística de la ejecución de cada algoritmo.

Diseño de la aplicación

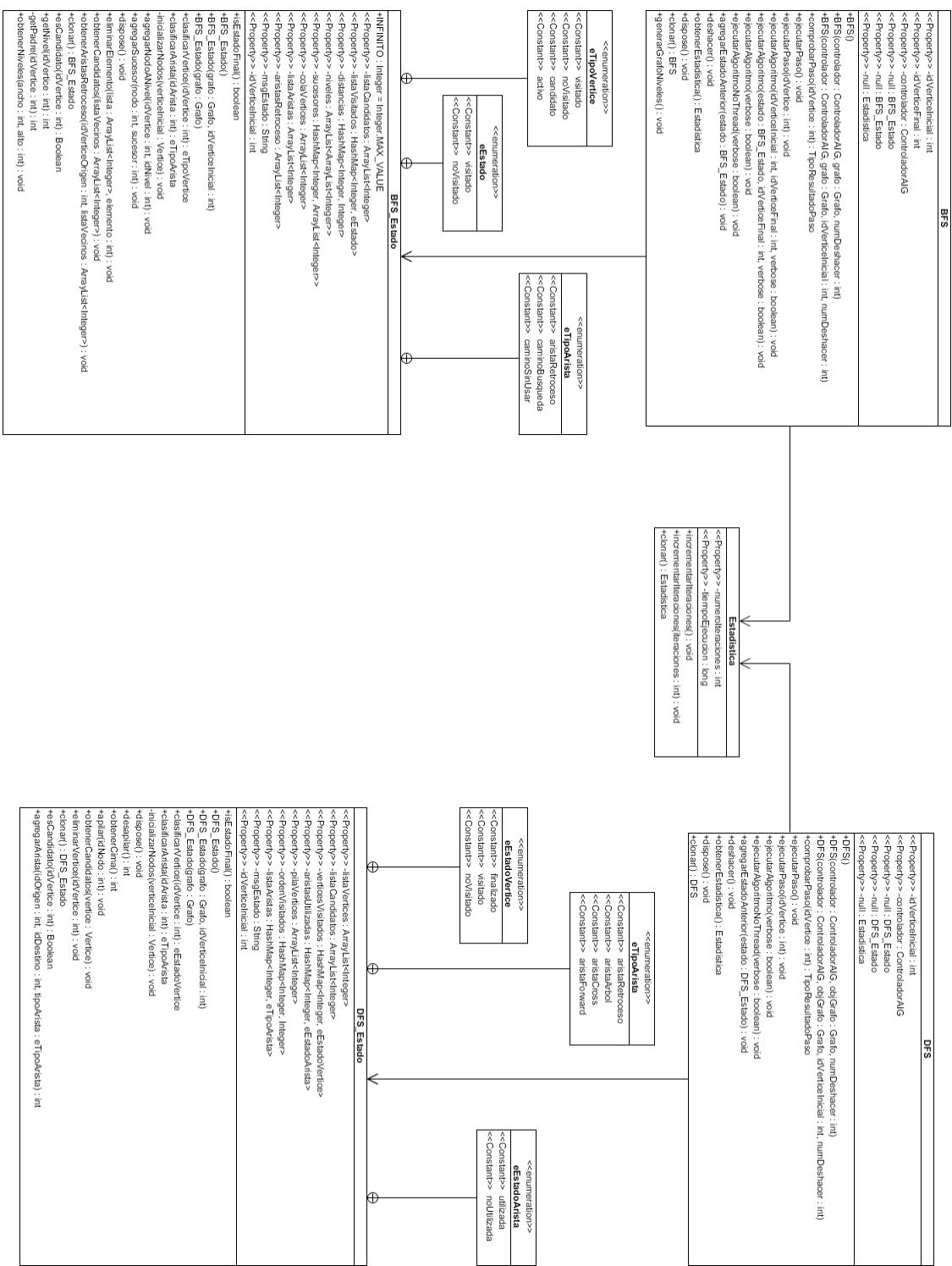


Figura 30. Diagrama de Clases que muestra los Algoritmos DFS y BFS

Algoritmos de Bloques y de Dijkstra

- **Bloques:** Clase que instancia un objeto que representa al algoritmo de Bloques. Contiene información relativa tal como vértice de inicio, la estadística de ejecución, el estado actual (representado por una instancia de la clase Bloques_Estado) y una lista de estados anteriores, entre otros.
- **Bloques_Estado:** Clase que instancia un objeto que representa información de la ejecución de un paso del algoritmo. Entre esta información se encuentra una lista de vértices de corte y una serie de listas de aristas (visitadas, de retroceso y puentes), los bloques generados, etc. Se apoya en dos clases de tipos enumerados: eTipoVértice y eEstado.
- **Dijkstra:** Clase que instancia un objeto que representa al algoritmo Dijkstra. Las propiedades más importantes son el vértice de inicio, la estadística de ejecución, el estado actual (representado por una instancia de la clase Dijkstra_Estado) y la lista de estados anteriores.
- **Dijkstra_Estado:** Clase que instancia un objeto que representa información de la ejecución de un paso del algoritmo. Entre esta información se encuentra una lista de vértices candidatos, visitados y predecesores, la distancia de cada vértice al origen y el camino mínimo. Se apoya en cuatro clases de tipos enumerados: eEstado, eTipoVertice, eAccion y eTipoArista.

Diseño de la aplicación

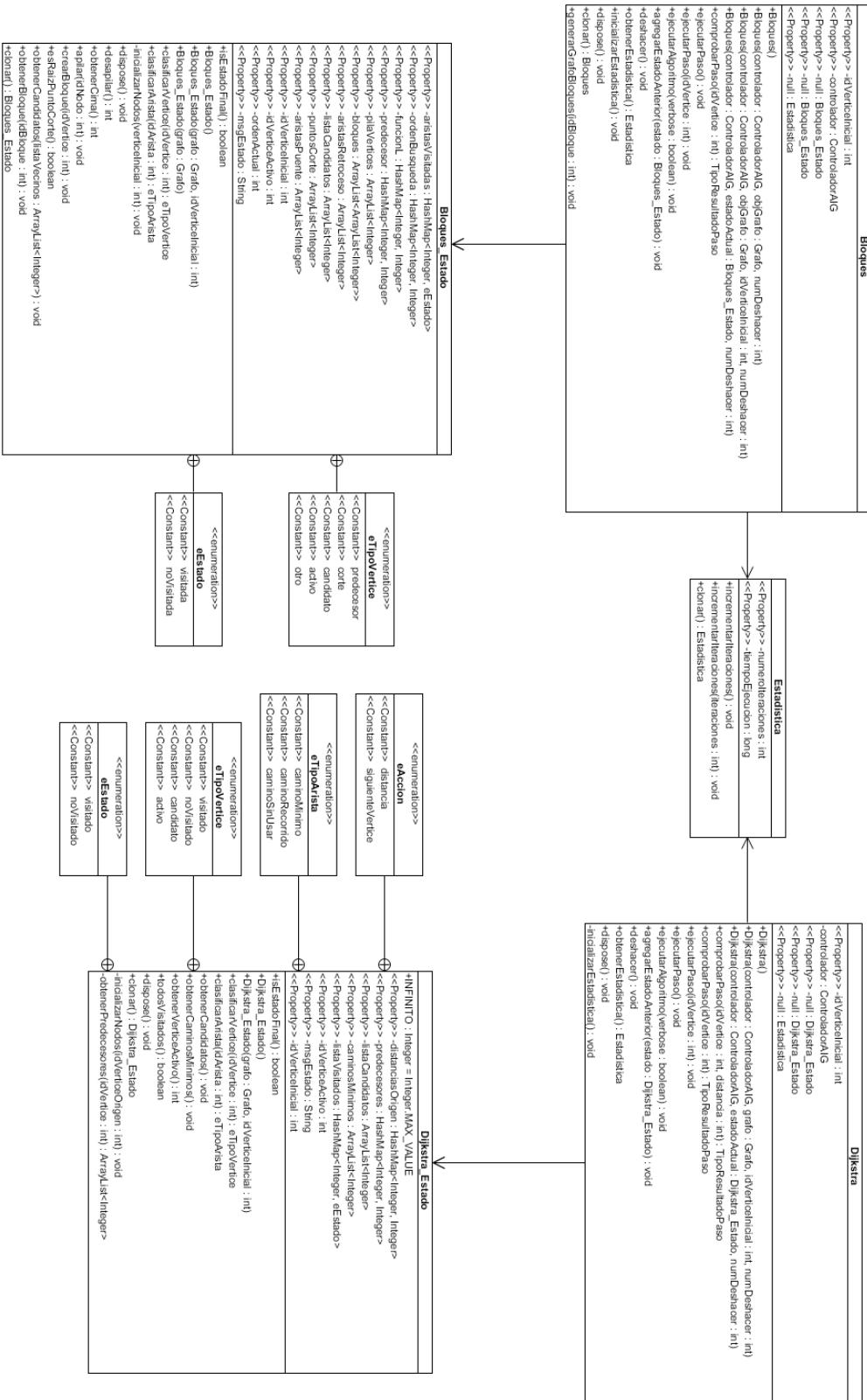


Figura 31. Diagrama de Clases que muestra los Algoritmos de Bloques y Dijkstra

Algoritmos de Coloración (Brelaz y Secuencial)

- **Col_Secuencial:** Clase que instancia un objeto que representa a los algoritmos de coloración secuencial. Se utiliza esta clase para implementar los algoritmos de coloración aleatoria, Welsh y Matula. Contiene información del estado actual y la estadística de ejecución.
- **Col_Secuencial_Estado:** Clase que instancia un objeto que representa información de la ejecución de un paso del algoritmo. Entre esta información se encuentra una lista de nodos (representa la lista en la que se irán coloreando los vértices del grafo, dependiendo del algoritmo que se esté ejecutando) y una matriz de grupos de colores.
- **Nodo_Secuencial:** Clase que instancia un objeto que representa un nodo del grafo en la ejecución del algoritmo.
- **Col_Brelaz:** Clase que instancia un objeto que representa al algoritmo de Brelaz. La información más relevante de esta clase viene representada por el estado actual y la estadística de ejecución.
- **Col_Brelaz_Estado:** Clase que instancia un objeto que representa información de la ejecución de un paso del algoritmo. Entre esta información se encuentra la lista de los nodos del grafo y una matriz de grupos de colores.
- **Nodo:** Clase que instancia un objeto que representa un nodo del grafo en la ejecución del algoritmo. Para cada nodo del grafo tendremos representado el grado del Vértice, el grado de Saturación y el orden en que ha sido coloreado.

Diseño de la aplicación



Figura 32. Diagrama de Clases con los Algoritmos de Coloración (Secuencial y Brelaz)

Algoritmos de coloración de conjuntos independientes

- **MCI:** Clase que instancia un objeto que representa al algoritmo de independencia. Contiene información del estado actual, la lista de estados anteriores y la estadística de ejecución.
- **MCI_Estado:** Clase que instancia un objeto que representa información de la ejecución de un paso del algoritmo. Entre esta información se encuentra el grafo de la iteración, la lista de vértices que conforman el conjunto independiente actual, una lista de conjuntos independientes, una lista de vértices no disponibles y los vecinos del nodo actual.

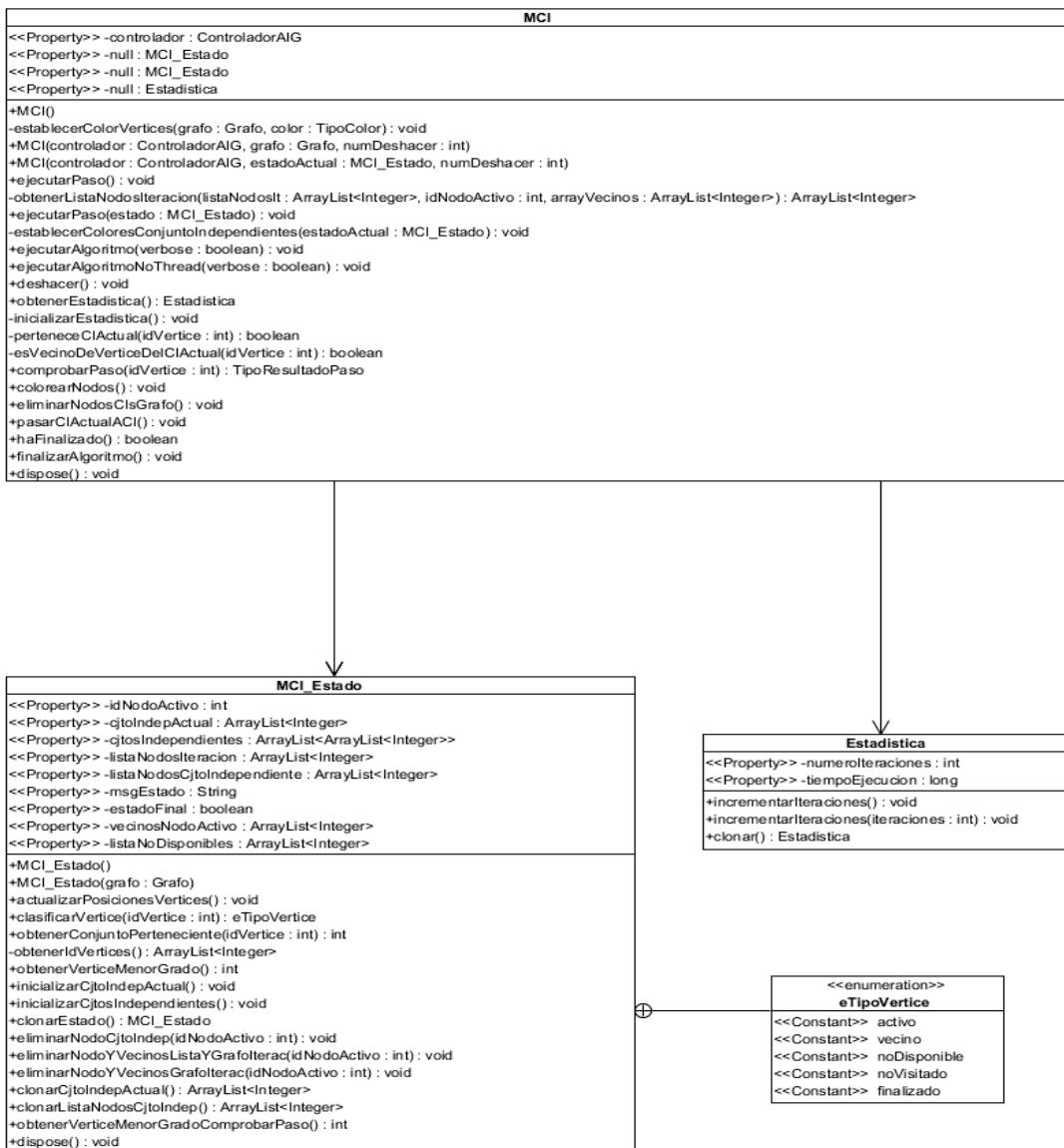


Figura 33. Diagrama de Clases con el Algoritmo de Conjuntos Independientes

Configuración

En este diagrama se explica las clases que implementan los datos configurables de la aplicación:

- **Configuracion:** Clase que sirve para representar la configuración del sistema. Las propiedades de esta clase son instancias de las clases explicadas a continuación.
- **Configuracion_Grafo:** Contiene la información relativa a la configuración del grafo, entre la que se encuentra el color y grosor de los vértices y aristas.
- **Configuracion_Idioma:** Los textos mostrados por la aplicación pueden estar en español o en inglés. Esta clase sirve para determinar el idioma de la aplicación en cada momento.
- **ConfiguracionEntorno:** Se apoya en cuatro tipos enumerados para guardar información relativa al tipo de proyecto (coloración o búsqueda), al algoritmo en ejecución, a la visualización en el lienzo del nombre o alias de los vértices del grafo y al tipo de ejecución del algoritmo (automática o interactiva).

Existen además cinco clases para configurar las características de los elementos mostrados en el lienzo, asociadas a cada uno de los algoritmos de la aplicación.

Son: Configuracion_DFS, Configuracion_BFS, Conguracion_Bloques, Configuracion_Dijkstra y Configuracion_MCI.

AIG: Búsquedas

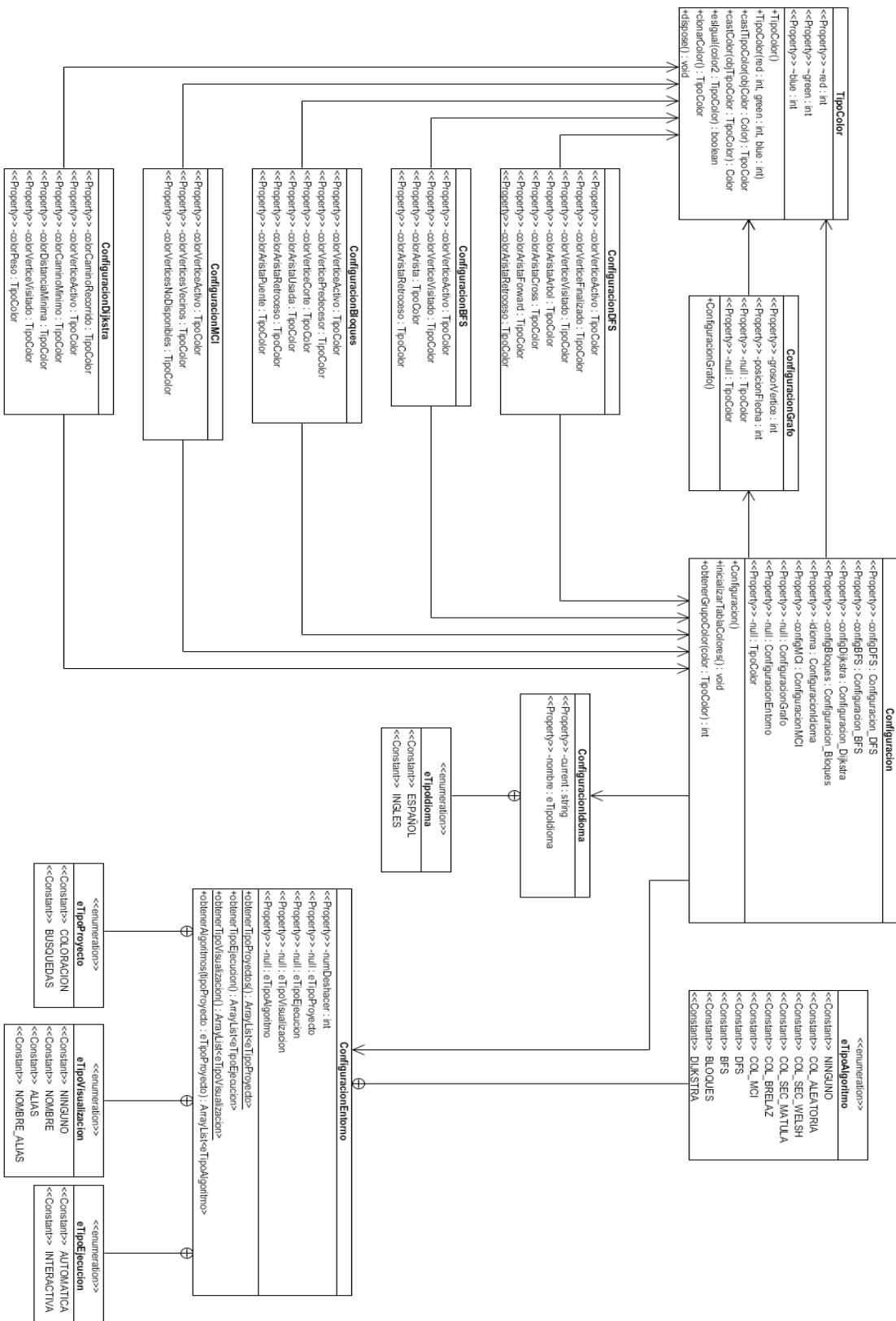


Figura 34. Diagrama de Clases que muestra la Configuración

5 MANUAL DE USUARIO

AIG (Aplicación Integral de Grafos) es una aplicación JAVA distribuida en un paquete JAR y por tanto ejecutable en un sistema con instalación de JVM. La distribución de los elementos en la aplicación es similar a la empleada en cualquier otra herramienta gráfica.

5.1 Partes de la aplicación

La ventana principal de la aplicación AIG se compone de los siguientes elementos:

- Barra de Menús
- Barra de Iconos (Botones)
- Información
- Posición
- Lienzo

5.1.1 Barra de Menús

A continuación describiremos los menús de la aplicación indicando la utilidad de cada una de las opciones. En esta sección no entraremos a explicar en profundidad cada una de las opciones.

Como puede apreciarse, al iniciarse la aplicación existen opciones del menú que están deshabilitadas. Estas se habilitan al abrir o crear un grafo, y en función del grafo activo en cada momento, se activarán unas opciones u otras.

5.1.1.1 Archivo

Como en la mayoría de aplicaciones de ventana, el primer menú que nos encontramos es “Archivo”. A continuación explicaremos brevemente cada una de las funcionalidades que nos encontraremos en este menú.

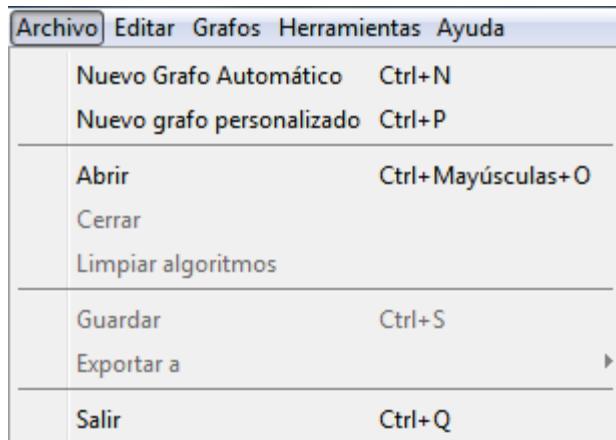


Figura 35. Menú Archivo

- **Nuevo Grafo Automático:** Nos permite crear un grafo de manera automática, tan solo tendremos que especificar el número de nodos y la probabilidad de que exista arista entre los nodos.
- **Nuevo Grafo Personalizado:** A diferencia con la opción “Nuevo Grafo Automático”, esta funcionalidad nos permite especificar manualmente las aristas existentes.
- **Abrir:** Nos permite cargar en la aplicación un grafo previamente salvado. Si en el momento en que el grafo fue guardado, estábamos inmersos en la ejecución de un algoritmo, se podrá continuar con la ejecución del algoritmo desde el punto en que fue almacenado.
- **Cerrar:** Elimina el grafo, dejando la aplicación en la misma situación que cuando se arranca.
- **Limpiar algoritmos:** Esta funcionalidad, elimina de pantalla cualquier referencia a la ejecución de un algoritmo, dejando solamente el estado inicial del grafo.

- **Guardar:** Nos permite la opción de almacenar el grafo con el que estamos trabajando. Dicho grafo se almacenará en formato XML. Si en el momento en que se decide almacenar el grafo, estamos ejecutando un algoritmo, también se almacenará el estado de la ejecución del algoritmo, para permitir posteriormente con la ejecución.
- **Exportar:** La aplicación nos permite exportar toda la información relativa al grafo en formato Excel.
- **Salir:** Nos permite cerrar la aplicación.

5.1.1.2 Editar

En el menú editar podremos editar el grafo en ejecución.

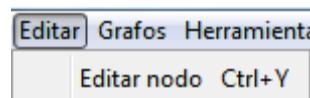


Figura 36. Menú Editar

Este menú contiene la siguiente opción:

- **Editar Nodo:** Esta funcionalidad permite cambiar información relativa a cada nodo. Cambio de alias y modificación (inserción/eliminación) de las aristas de cada vértice.

5.1.1.3 Grafos

En este menú encontraremos una serie de grafos predefinidos que pueden cargarse para trabajar con ellos. Algunos de ellos, como Tutte, son grafos fijos, mientras que otros, como Estrella, permiten que el usuario introduzca el número de vértices que se desea que tenga. Las características de cada uno se explicarán en la sección “Grafos Tipo”

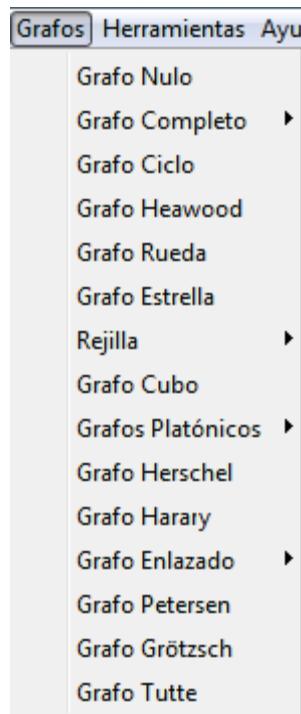


Figura 37. Menú Grafos

5.1.1.4 Herramientas

El siguiente menú ofrece una serie de herramientas que el usuario podrá seleccionar durante su trabajo con la aplicación.



Figura 38. Menú Herramientas

Este menú contiene las siguientes opciones:

- **Ejecución:** Permite al usuario seleccionar un algoritmo para iniciar su ejecución. El usuario podrá seleccionar si desea ejecutar el algoritmo de manera automática o interactiva. Esta opción permanecerá deshabilitada mientras no exista un grafo activo en la aplicación.
- **Opciones:** El usuario podrá configurar una serie de parámetros predefinidos en la aplicación, tales como, el color de los nodos, aristas, tamaño de la arista, etc. La aplicación permite que se pueda modificar el idioma.
- **Imagen de fondo:** Permite que el usuario pueda elegir que el lienzo tenga una imagen de fondo.
- **Limpiar fondo:** Elimina la imagen de fondo cargada, restableciendo el fondo blanco.
- **Estadísticas:** Esta funcionalidad permite que el usuario compare la ejecución de diversos algoritmos respecto al grafo activo en la aplicación. El resultado se generará en formato Excel.
- **Matriz de Adyacencia:** Mostrará la matriz de adyacencia asociada al grafo activo en la aplicación.
- **Matriz de Ponderación:** Mostrará la matriz de ponderación asociada al grafo activo en la aplicación. Muestra el peso de las aristas existentes entre vértices.

5.1.1.5 Ayuda

El siguiente menú ofrece ver la documentación del menú de usuario, los métodos abreviados del teclado e información acerca de los desarrolladores del proyecto.

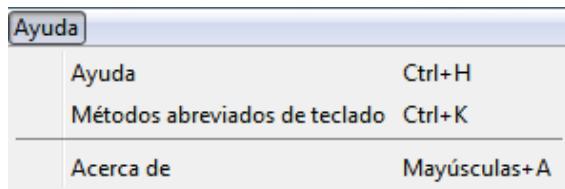


Figura 39. Menú Ayuda

5.1.2 Barra de Iconos (botones)

La barra de iconos (botones) permite al usuario seleccionar las operaciones más frecuentes, además de poder seleccionar una serie de opciones tanto para modificar gráficamente el grafo, como configurar el proyecto sobre el que se está trabajando.



Figura 40. Barra de Iconos

A continuación explicaremos brevemente la funcionalidad de cada elemento de la barra de iconos:

- ❑ **Nuevo Grafo Automático:** Permite crear un Nuevo Grafo Automático. Realiza la misma funcionalidad que si seleccionáramos **Archivo** → **Nuevo Grafo Automático**.
- ❑ **Guardar:** Permite salvar un grafo activo a un archivo en formato XML. Realiza la misma funcionalidad que si seleccionáramos **Archivo** → **Guardar**.
- ❑ **Abrir:** Permite cargar un grafo previamente guardado en la aplicación. Realiza la misma funcionalidad que si seleccionáramos **Archivo** → **Abrir**.
- ❑ **Imagen de fondo:** Inserta una imagen de fondo en el lienzo. Realiza la misma funcionalidad que si seleccionáramos **Herramientas** → **Imagen de fondo**.
- ❑ **Ejecutar algoritmo:** Permite la ejecución de algoritmos sobre el grafo activo en la aplicación. Realiza la misma funcionalidad que si seleccionáramos **Herramientas** → **Ejecución**. Este botón permanecerá deshabilitado mientras no exista un grafo activo en la aplicación.
- ❑ **Opciones:** Permite al usuario configurar una serie de parámetros en la aplicación. Realiza la misma funcionalidad que si seleccionáramos **Herramientas** → **Opciones**.

A continuación tenemos una serie de cuatro botones, que permiten el manejo manual del grafo sobre el lienzo. Estos botones están relacionados entre sí, y no puede haber dos botones seleccionados a la vez, con lo que si tenemos uno seleccionado y pulsamos sobre otro, el primero se

deseleccionará. Inicialmente ninguno de los botones está seleccionado, con lo que solamente se podrán mover los nodos del grafo, para ello el usuario colocará el puntero del ratón sobre el vértice deseado, pulsará con el botón izquierdo del ratón sobre él, y sin soltar el botón izquierdo del ratón, podrá mover libremente el vértice seleccionado hasta la posición deseada. Pasaremos a explicar la funcionalidad de cada uno de estos cuatro botones.

- **Insertar nodo:** Permite al usuario insertar un nodo, únicamente realizando un clic con el botón izquierdo del ratón sobre la posición deseada. Si el lugar deseado coincide con la posición donde está un nodo ya creado, no se insertará un nuevo vértice en dicha posición. Realiza una funcionalidad similar que si seleccionáramos **Editar → Insertar → Nodo**.
- **Eliminar nodo:** Permite al usuario eliminar un nodo, únicamente realizando un clic con el botón izquierdo del ratón sobre el vértice que se desea borrar. Hecho esto, se eliminarán tanto el nodo seleccionado, como todas las aristas que lleguen o salgan de dicho vértice. Realiza una funcionalidad similar que si seleccionáramos **Editar → Suprimir → Nodo**.
- **Insertar arista:** Permite al usuario insertar una arista entre dos nodos, para ello pulsará con el botón izquierdo del ratón sobre el vértice origen de la arista y sin soltar dicho botón, moverá el cursor hasta el vértice destino de la arista, donde soltará el ratón. El usuario podrá comprobar que una vez que ha pulsado sobre el vértice origen y está moviendo el cursor, aparece en pantalla la arista que está insertando. Realiza una funcionalidad similar que si seleccionáramos **Editar → Insertar → Arista**.
- **Eliminar arista:** Permite al usuario eliminar una arista existente entre dos nodos. Para ello, el usuario únicamente tendrá que realizar un clic con el botón izquierdo del ratón, sobre la arista que se desea eliminar. Realiza una funcionalidad similar que si seleccionáramos **Editar → Suprimir → Arista**.

Después de este conjunto de botones podemos ver que tenemos dos cuadros de selección, el primero de ellos, nos permite seleccionar el tipo de proyecto con el que vamos a trabajar: Coloración o Búsquedas, la selección de uno u otro tipo de proyecto afecta a la hora de los algoritmos que puede ejecutar el

usuario, ya que si esta seleccionado el proyecto Coloración, aparecerán unos algoritmos de ejecución, distintos a si tenemos seleccionado el proyecto Búsquedas. De manera similar ocurrirá cuando seleccionemos la opción Estadísticas. El segundo cuadro de selección permite al usuario la opción de que en el lienzo se muestren al lado de los vértices, su nombre, su alias, su nombre y alias o que no aparezca ninguna de estas opciones.

Para finalizar tenemos los siguientes botones:

- ▣ **Mostrar ponderación (Distancias):** Permite al usuario poder visualizar el peso de cada arista sobre el lienzo. Si pulsamos dicho botón aparecerán sobre las aristas el peso de cada una de ellas.
- ▣ **Mostrar tooltip:** Permite al usuario visualizar sobre el lienzo, información de los vértices y aristas, según se vaya situando el cursor el ratón sobre ellos.
- ▣ **Matriz de adyacencia:** Mostrará la matriz de adyacencia asociada al grafo activo en la aplicación. Realiza una función similar que si seleccionáramos **Herramientas → Matriz de Adyacencia**.
- ▣ **Matriz de ponderación:** Mostrará la matriz de ponderación asociada al grafo activo en la aplicación. Muestra el peso de las aristas existentes entre vértices. Realiza una función similar que si seleccionáramos **Herramientas → Matriz de Ponderación**.
- △ **Mostrar niveles:** Este icono sólo estará activo al finalizar la ejecución del algoritmo BFS. Mostrará el árbol de niveles generado por el algoritmo BFS sobre el grafo original.
- ▣  **Mostrar bloques:** Estos iconos sólo estarán activos al finalizar la ejecución del algoritmo de bloques (conectividad). Una vez que se pulse el botón de la izquierda (mostrar bloques) se nos mostrará por pantalla el primer bloque existente. Para ver el resto de bloques se deberán pulsar las flechas de dirección situadas a la derecha de este botón.

5.1.3 Información

La zona de información está situada a en la parte izquierda de la pantalla y se compone de tres partes diferenciadas.

- **Información del grafo (Árbol).** Es la parte superior de la zona de información, mostrará información de los nodos y aristas del grafo activo.

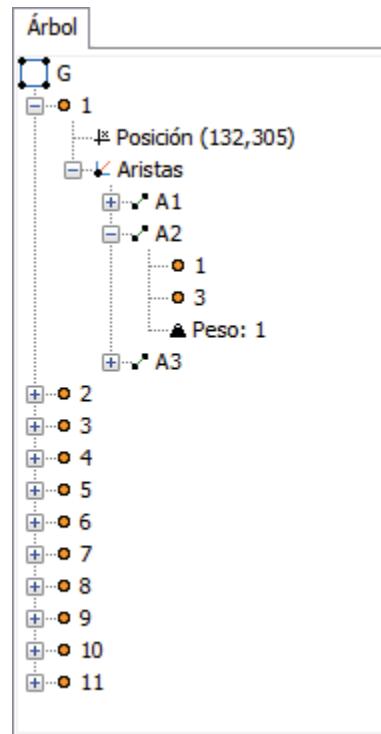


Figura 41. Árbol de un grafo

Podemos ver que nos aparecen listados de arriba hacia abajo los vértices, y si desplegamos un vértice nos aparecerá la posición de dicho vértice y las aristas que llegan y salen de él. Pulsando en una de las aristas podremos ver los dos nodos que unen y el peso de dicha arista.

Durante la ejecución de los algoritmos se mostrará en esta zona una pestaña donde podremos obtener información del significado de los colores usados en vértices y aristas durante el transcurso de

la ejecución del algoritmo. Llamaremos a esta pestaña *Leyenda* y tendrá un aspecto similar a este:

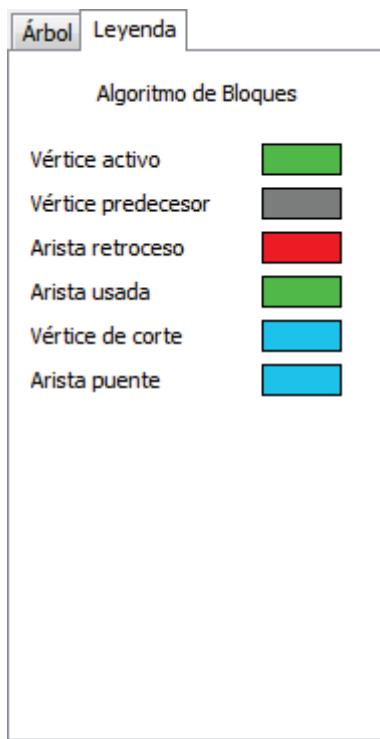


Figura 42. Leyenda del algoritmo de Bloques

Durante la ejecución del algoritmo de Coloración de Brelaz, aparecerá una pestaña junto a la pestaña del Árbol, donde se nos mostrará una tabla con información relativa al algoritmo.

- **Botones de Ejecución.** Aparecen en la parte central de la zona de información y son cuatro botones.

La ejecución automática de un algoritmo puede realizarse de dos maneras diferentes: paso a paso o de manera completa, pudiendo mezclar los dos tipos en la ejecución de un algoritmo

La funcionalidad de los botones es la siguiente:

◀ **Paso hacia atrás.** Si durante la ejecución de un algoritmo paso a paso, queremos volver al paso anterior, se pulsará este botón.

▶ **Iniciar (Play).** Si el usuario desea ejecutar un algoritmo automático hasta que este finalice pulsará el botón de Play.

II Parar (Pause). Una vez que el usuario ha pulsado el botón Play, puede parar la ejecución del algoritmo pulsando el botón de Pause.

» Paso hacia delante. La manera de avanzar en la ejecución de un algoritmo paso a paso es pulsando este botón.

Estos botones inicialmente están deshabilitados y solamente se activarán cuando se seleccione la ejecución automática de un algoritmo.

- **Información del algoritmo en ejecución.** Es la parte inferior de la zona de información, aquí se nos mostrará información sobre la ejecución del algoritmo seleccionado.

Una vez que el usuario haya empezado la ejecución de un algoritmo irá apareciendo como se está realizando el algoritmo, la información que aparezca dependerá del algoritmo seleccionado.

La figura siguiente nos presenta un ejemplo de la información que se obtiene al ejecutar un algoritmo.

```

Conectividad (algoritmo de bloques)
Pila = {1} --> Cima = 1
Candidatos = {2, 3, 4}

Pila = {2, 1} --> Cima = 2
Candidatos = {3, 5}

Pila = {3, 2, 1} --> Cima = 3
Candidatos = {1, 4, 5}

Retroceso = {{(1,3)}}
Candidatos = {4, 5}

Pila = {4, 3, 2, 1} --> Cima = 4
Candidatos = {1, 5, 6}

Retroceso = {{(1,3),(1,4)}}
Candidatos = {5, 6}

Pila = {5, 4, 3, 2, 1} --> Cima = 5
Candidatos = {3, 2, 6, 7}

Retroceso = {{(1,3),(1,4),(3,5)}}
Candidatos = {2, 6, 7}

Retroceso = {{(1,3),(1,4),(3,5),(5,2)}}
Candidatos = {6, 7}

Pila = {6, 5, 4, 3, 2, 1} --> Cima = 6
Candidatos = {4, 7, 10}

```

Figura 43. Información de la ejecución de un algoritmo

5.1.4 Posición

En la parte inferior derecha de la aplicación podremos ver la posición del cursor del ratón sobre el lienzo.



Figura 44. Cuadro de posición del cursor

5.1.5 Lienzo

Es la parte principal de la aplicación, situada en la zona central derecha de la ventana principal.

Sobre el lienzo se mostrará gráficamente el grafo con el que se desea trabajar y sobre se mostrarán las ejecuciones de los algoritmos.

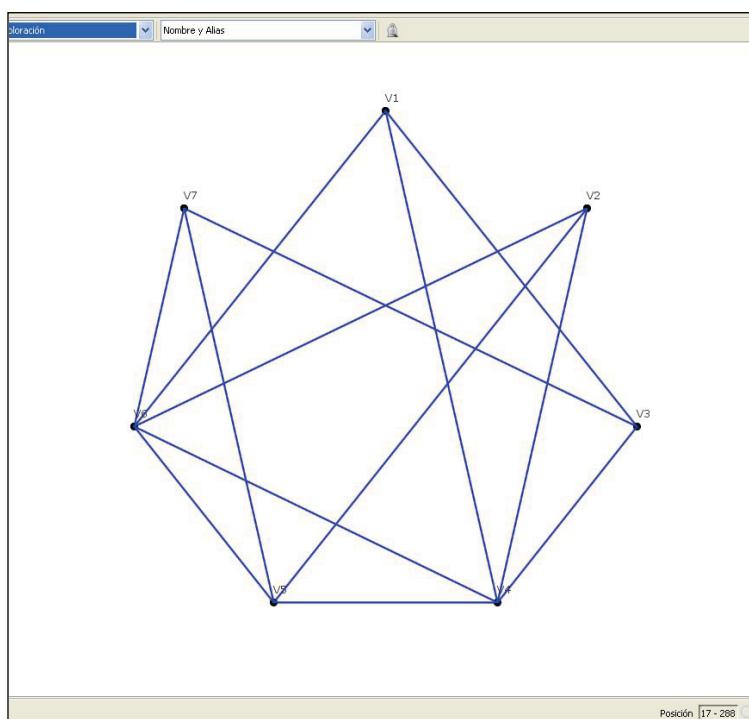


Figura 45. Lienzo de la aplicación mostrando un grafo

Todas y cada una de las acciones que se realicen tanto a través de los menús como de los botones tendrán un efecto sobre el lienzo o sobre el grafo del lienzo.

Adicionalmente se han habilitado un menú contextual cuando se pulsa con el botón derecho sobre el lienzo. Este menú variará dependiendo de si estamos en la ejecución de un algoritmo o editando un grafo.

Si estamos editando un grafo, disponemos de cuatro funcionalidades:

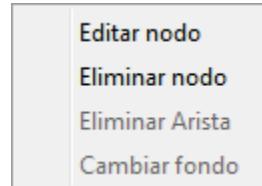


Figura 46. Menú contextual de edición de grafo

- **Editar Nodo:** Esta opción estará únicamente disponible cuando se pulse sobre un vértice. Permite al usuario editar un nodo. Realiza una funcionalidad similar que si seleccionamos **Editar → Editar Nodo**, con la diferencia a que ahora el nodo seleccionado aparece cargado en la ventana de edición del nodo.
- **Eliminar Nodo:** Esta opción estará únicamente disponible cuando se pulse sobre un vértice. Permite al usuario eliminar un nodo. Se eliminarán tanto el nodo seleccionado, como todas las aristas que lleguen o salgan de dicho vértice.
- **Eliminar Arista:** Esta opción estará únicamente disponible cuando se pulse sobre una arista. Permite al usuario eliminar una arista existente entre dos nodos.
- **Cambiar Fondo:** Inserta una imagen de fondo en el lienzo. Realiza la misma funcionalidad que si seleccionáramos **Herramientas → Imagen de fondo**.

Si estamos ejecutando un algoritmo, se nos mostrará un menú similar al de la siguiente figura:

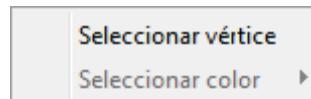


Figura 47. Menú contextual de ejecución de algoritmo

Dependiendo del algoritmo en ejecución variarán las opciones disponibles en este menú. Se explicarán ampliamente en la sección “Ejecución” en la parte correspondiente de cada algoritmo.

El usuario dispone de la posibilidad de visualizar información relativa a los vértices y a las aristas sobre el lienzo. Para ello deberá tener seleccionado el botón *Mostrar Tooltip* . Una vez realizado esto, cuando el usuario pase el cursor del ratón por encima de un vértice o de una arista, se mostrará una ventana similar a la que nos muestra la siguiente figura.

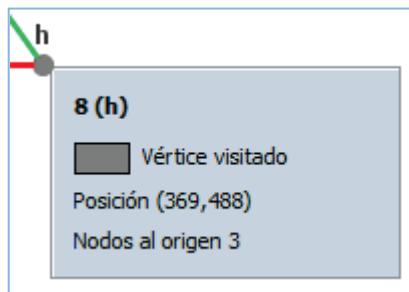


Figura 48. Ventana de información de vértice sobre el lienzo

La información mostrada variará dependiendo de si estamos o no en la ejecución de un algoritmo, y si se está en ejecución, del algoritmo que se esté ejecutando en ese determinado momento.

5.2 Operaciones genéricas

5.2.1 Crear nuevo grafo automático

En el caso de que se seleccione *Nuevo Grafo Automático* en el menú *Archivo* o que se pulse el ícono *Nuevo Grafo Automático*, podremos generar un nuevo grafo de forma automática. En la figura siguiente podemos ver la ventana de creación a través de la cual se puede generar el nuevo grafo.

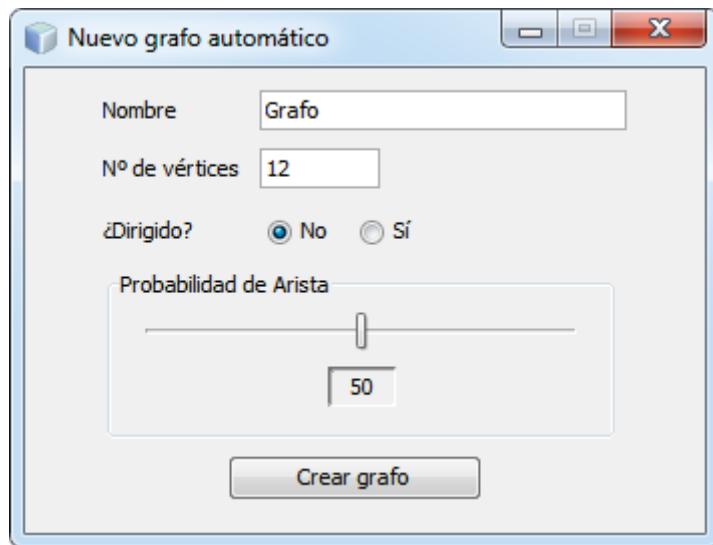


Figura 49. Ventana de creación de un nuevo grafo automático

En ella deberemos introducir el nombre del grafo (de manera opcional), el número de vértices que compondrán el grafo, si es dirigido o no y la probabilidad de que exista una arista entre dos nodos.

El campo *Nº de vértices* solamente acepta caracteres numéricos.

La probabilidad de Arista varía entre 0 y 100. Si el usuario pone como probabilidad 0, solamente aparecerán los nodos (sin ninguna arista) y si el usuario decide que la probabilidad sea 100, resultará un grafo completo, existiendo una arista desde un nodo hacia cada uno de los nodos existentes.

AIG: Búsquedas

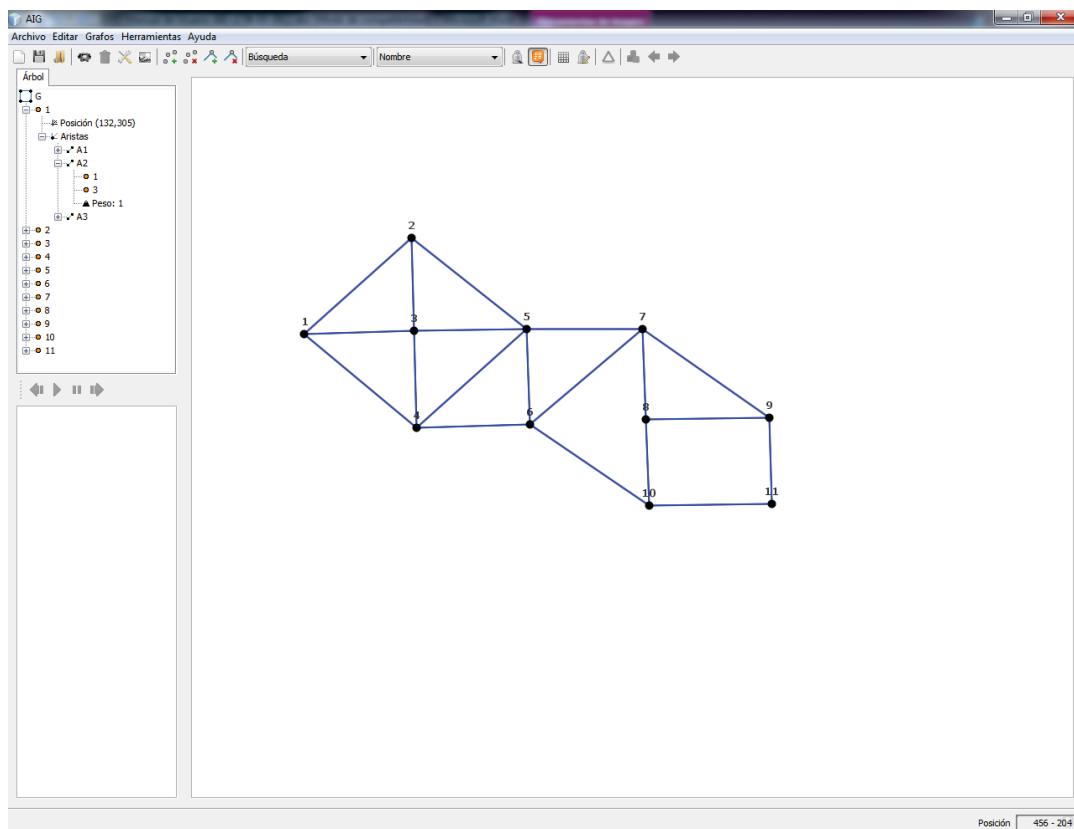


Figura 50. Detalle de la aplicación una vez creado un grafo

Una vez que el usuario pulsa el botón *Crear Grafo*, la ventana de la aplicación tendrá un aspecto similar al que nos muestra la figura anterior.

Podemos comprobar que en el lienzo aparece la representación gráfica del grafo y que en la pestaña *Árbol* de la zona de información aparecerá el grafo en forma de árbol, donde podremos ver sus nodos, posición de estos, y aristas de que se compone.

Si en el momento en el que usuario decide crear un nuevo grafo ya existe un grafo en la aplicación se nos mostrará la siguiente ventana:

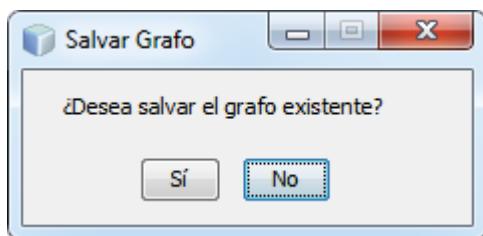


Figura 51. Ventana de confirmación para salvar el grafo

Si el usuario pulsa en el botón *Sí*, se nos mostrará la ventana para salvar el grafo (ver el punto 5.2.3 Guardar Grafo). Si, por el contrario, el usuario pulsa el botón *No* se nos mostrará la ventana para la creación del nuevo grafo automático.

5.2.2 Crear nuevo grafo personalizado

Existe la opción de que el usuario personalice el grafo en mayor profundidad, para ello deberá pulsar en la pestaña *Archivo* y a continuación seleccionar la opción *Nuevo Grafo Personalizado*. Una vez realizado esto, nos aparecerá la siguiente ventana:

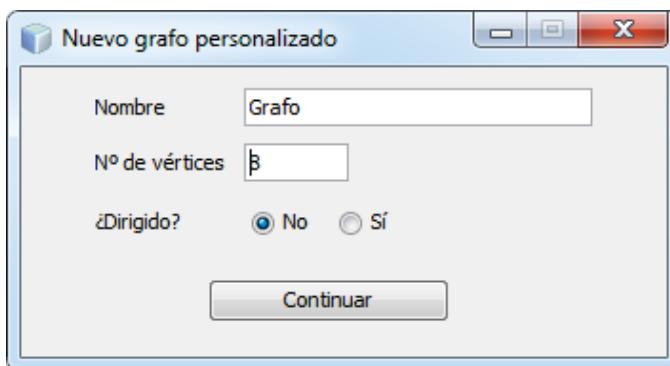


Figura 52. Ventana de creación de un nuevo grafo personalizado

Podemos comprobar que la ventana es similar a la que aparece al crear un grafo de manera automática, con la salvedad que ha desaparecido la probabilidad de que exista una arista entre los nodos.

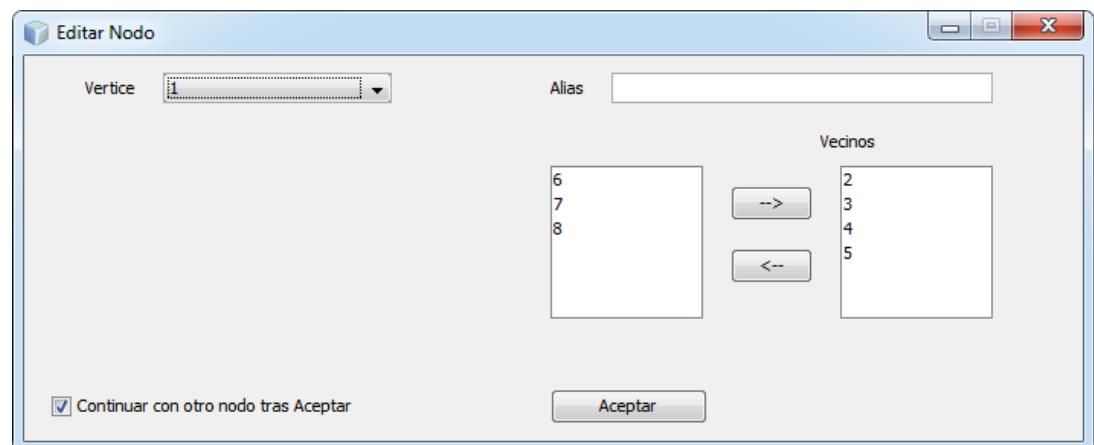


Figura 53. Edición de nodos en la creación de un grafo (no dirigido) personalizado

Si hemos seleccionado que el grafo no será dirigido nos aparecerá una ventana como la de la figura anterior.

En ella podremos ir seleccionando los vértices uno a uno, y podremos ponerle un alias al nodo e indicar cuales son los nodos vecinos, para ello seleccionaremos del cuadro de la izquierda los nodos vecinos para el vértice en edición y pulsaremos el botón con la flecha hacia la derecha (\rightarrow). Si por alguna razón el usuario decide quitar algún nodo vecino, tendrá que seleccionar del cuadro de vecinos dicho nodo y pulsar el botón con la flecha hacia la izquierda (\leftarrow). Se puede ver que cuando se pulsa a uno de estos dos botones, los nodos seleccionados pasan de un cuadro a otro.

Cuando el usuario pulse el botón *Aceptar* podrá comprobar que en el lienzo se han creado las aristas entre los nodos seleccionados como vecinos y el nodo editado.

El usuario podrá ir cambiando de nodo e ir editando cada uno de ellos.

En la parte inferior izquierdo hay un check-box, donde se nos indica que tras aceptar se continuará con la edición de otro nodo. Si el usuario deselecciona esta opción, cuando se pulse el botón *Aceptar*, se cerrará la ventana de edición del nodo.

Si por el contrario el usuario al crear el nodo ha decidido que sea dirigido aparecerá la siguiente ventana:

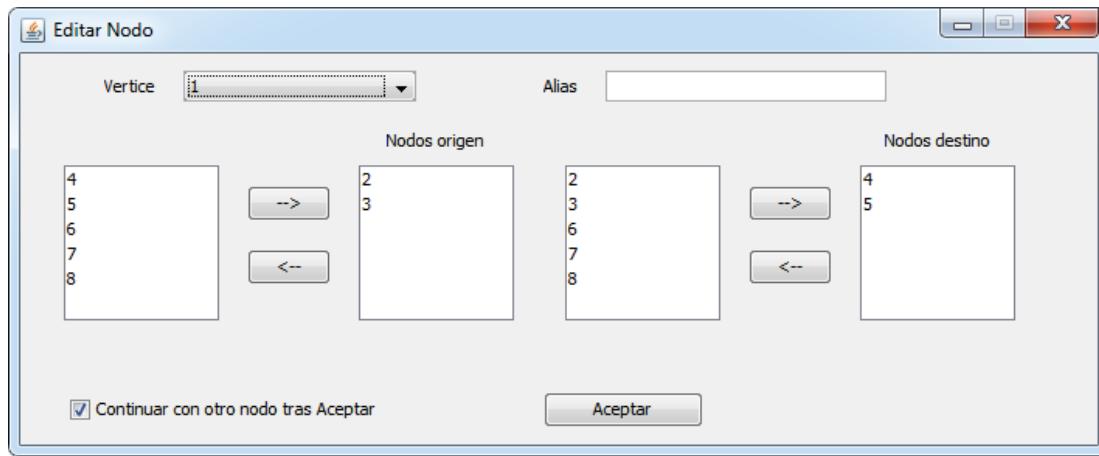


Figura 54. Edición de nodos en la creación de un grafo (dirigido) personalizado

Esta pantalla es similar a la que nos aparece cuando el grafo no es dirigido, con la salvedad, que como ahora el grafo es dirigido, podemos:

- Elegir los vértices que son origen de un arista que llega hasta el vértice seleccionado, que se corresponde con los dos cuadros más a la izquierda de la pantalla (los nodos seleccionados que pasen al cuadro con el texto *Nodo origen*, tendrán una arista cuyo origen será dichos nodos y cuyo destino será el vértice que se está editando).
- Elegir los vértices que son destino de un arista cuyo origen será el vértice seleccionado, que se corresponde con los dos cuadros más a la derecha de la pantalla (los nodos seleccionados que pasen al cuadro con el texto *Nodo destino*, tendrán una arista cuyo destino será dichos nodos y cuyo origen será el vértice que se está editando).

Según vayamos dando al botón *Aceptar*, podremos ir observando los cambios en el lienzo.

Si en el momento en el que usuario decide crear un nuevo grafo ya existe un grafo en la aplicación se nos mostrará una ventana preguntándonos si deseamos salvar el grafo.

Si el usuario pulsa en el botón *Sí*, se nos mostrará la ventana para salvar el grafo (ver el punto 5.2.3 Guardar Grafo). Si, por el contrario, el usuario pulsa el botón *No*, se nos mostrará la ventana para la creación del nuevo grafo personalizado.

5.2.3 Guardar grafo

Una vez que tenemos un grafo activo en la aplicación, se ofrece al usuario la posibilidad de guardar dicho grafo para usarlo en el futuro, solamente tendrá que pulsar en el menú Archivo y seleccionar la opción *Guardar* o pulsar sobre el botón de guardar en la barra de Iconos. El usuario tendrá que introducir el nombre deseado para el grafo. El grafo se salvará en formato XML.

5.2.4 Abrir grafo

En un determinado momento, el usuario podrá decidir cargar en la aplicación un grafo previamente salvado, para ello deberá pulsar en el menú Archivo y seleccionar la opción *Abrir* o pulsar sobre el botón de abrir en la barra de iconos.

Una vez realizado una de estas opciones nos aparecerá la siguiente ventana:

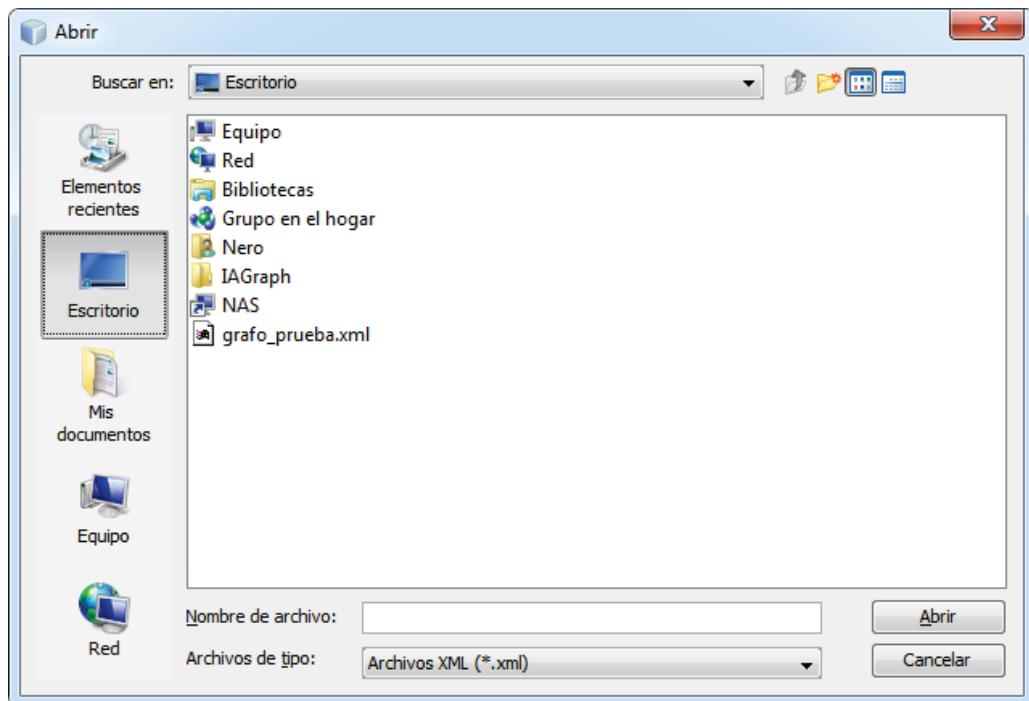


Figura 55. Ventana de selección del archivo (grafo) a cargar en la aplicación

Fíjese que es una ventana similar a la que nos aparece en la opción *Guardar*. El usuario deberá ir navegando por su árbol de directorios hasta llegar a la carpeta donde se aloja el grafo que se desea cargar en la aplicación, le seleccionará y pulsará sobre el botón Abrir.

Una vez realizado esto el grafo se cargará en la aplicación y se podrá ver su representación gráfica sobre el lienzo, y en el árbol de la zona de información, podremos ver sus características.

Si seleccionamos un archivo que no contiene guardado un grafo, se producirá un error durante el proceso de carga del archivo y se mostrará por pantalla un mensaje.

5.2.5 Cerrar

El usuario dispone de la posibilidad de cerrar el grafo con el que está trabajando dejando la aplicación de la misma manera que cuando se inicializa.

Para ello deberá pulsar en el menú Archivo y seleccionar la opción *Cerrar*, una vez hecho esto la aplicación nos indicará si deseamos salvar el grafo en ejecución. Después de que el usuario decida si salva o no el grafo activo, la aplicación volverá a su situación inicial, sin ningún grafo activo en la aplicación, mostrando tanto el lienzo como el árbol de la zona de información vacíos.

5.2.6 Limpiar algoritmos

La funcionalidad principal de la aplicación es la de ejecutar una serie de algoritmos sobre un grafo que esté cargado en la aplicación. Durante la ejecución de estos algoritmos, sobre el grafo mostrado en el lienzo se irán haciendo diversas modificaciones, se colorearán tanto nodos como aristas dependiendo del algoritmo seleccionado.

Una vez terminada la ejecución de un algoritmo, o durante la ejecución de uno, el usuario puede decidir que quiere ejecutar otro algoritmo, o simplemente volver al estado inicial del grafo (sin las modificaciones comentadas anteriormente).

Para ello el usuario pulsará en el menú *Archivo* y seleccionará la opción *Limpiar algoritmos*, una vez hecho esto, el grafo volverá a la situación inicial antes de la ejecución del algoritmo. Además se limpiará la ventana de información del algoritmo de todo texto mostrado durante la ejecución del algoritmo.

5.2.7 Exportar grafo

La aplicación ofrece la posibilidad de generar documentación en formato XLS del grafo activo. Para ello el usuario pulsará en Archivo y dentro de la opción *Exportar a* elegirá el formato Excel.

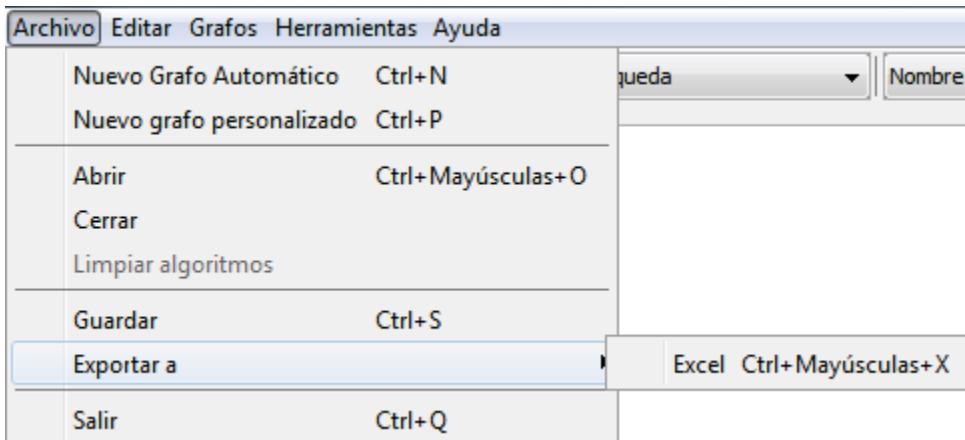


Figura 56. Menú Archivo → Exportar a

Una vez seleccionada una de estas dos opciones se nos abrirá una ventana similar a la de la opción *Guardar*, donde tendremos que elegir la ubicación y el nombre del archivo que se va a generar (en formato XLS).

El archivo XLS generado se compone de cuatro pestañas:

- **Información:** Contiene información general del grafo: nombre, si es dirigido, número de vértices y de aristas. Además incluye una representación gráfica del mismo.
- **Vértices:** Nos muestra la lista de vértices del grafo. Para cada vértice podremos ver su nombre, posición y las aristas que llegan/salen del nodo y el vértice de origen/destino de dicha arista.
- **Aristas:** Nos muestra la lista de aristas del grafo. Pudiendo ver para cada arista, su nombre, peso y los vértices de origen y destino.
- **Matriz de adyacencia:** Representa la matriz de adyacencia del grafo.

5.2.8 Salir

Para salir de la aplicación el usuario deberá pulsar en el menú *Archivo* y seleccionar la opción *Salir*.

5.2.9 Editar nodo

Se ofrece al usuario la posibilidad de editar un nodo. Para ello deberá pulsar en el menú *Editar* y seleccionar *Editar Nodo*, o pulsar con el botón derecho del ratón sobre el nodo que se desea editar y en el menú que se abrirá seleccionar *Editar Nodo*.

Una vez que el usuario ha realizado esta operación se nos mostrará la ventana de edición de nodo. Esta ventana ya se ha visto en la sección *Crear Nuevo Grafo Personalizado*. La ventana de edición de nodo será diferente si el grafo es o no dirigido.

Grafo No Dirigido: Se nos mostrará la siguiente pantalla:

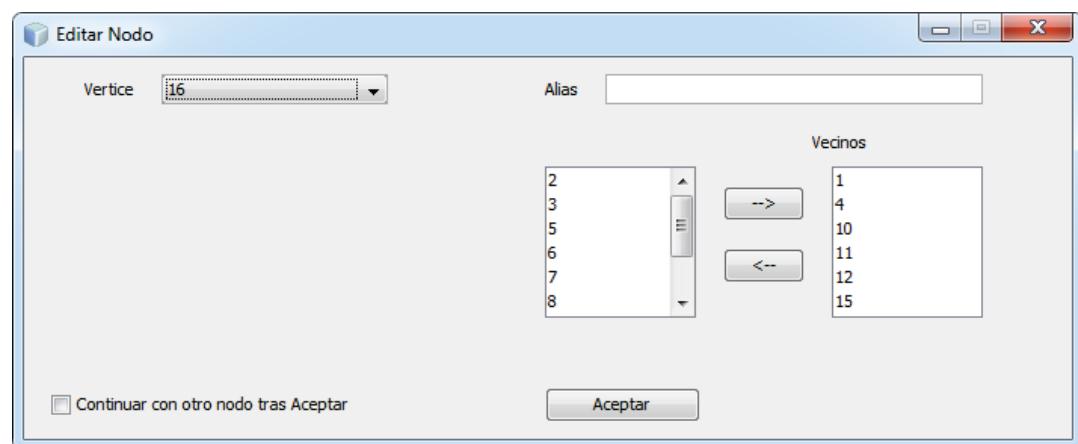


Figura 57. Ventana de edición de un grafo no dirigido

En el cuadro de selección *Vértice* podremos seleccionar el vértice que deseamos editar. Una vez seleccionado se cargarán los datos referidos a dicho vértice: su alias (si tuviese). Los vértices vecinos al nodo en edición (vértices unidos a través de una arista) aparecerán en el cuadro de la derecha bajo la etiqueta *Vecinos*. El resto de nodos del grafo, que no son vecinos del nodo en edición aparecerán en el cuadro de la izquierda. Se dispone de la posibilidad de añadir un vecino seleccionando un vértice del

cuadro de la izquierda y pulsando el botón \rightarrow . El añadir un vecino implica la creación de una arista entre estos dos vértices.

Asimismo disponemos de la posibilidad de eliminar a un vecino. Se seleccionará el vecino del cuadro de la derecha y se pulsará al botón \leftarrow . El eliminar un vecino implica la desaparición de la arista que unía a estos dos vértices.

Para confirmar los cambios se pulsará al botón *Aceptar*.

Grafo Dirigido: Se nos mostrará la siguiente pantalla:

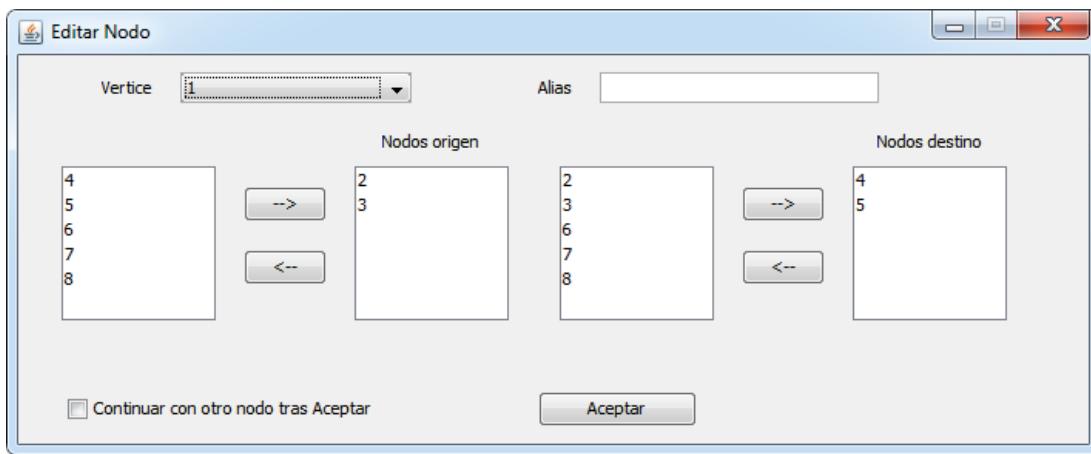


Figura 58. Ventana de edición de un grafo dirigido

En el cuadro de selección Vértice podremos seleccionar el vértice que deseamos editar. Una vez seleccionado se cargarán los datos referidos a dicho vértice: su alias (si tuviese).

Además nos mostrará los vértices que son origen de una arista cuyo destino es el vértice en edición (serán los vértices que aparecen en el cuadro *Nodos origen*) y los vértices que son destino de una arista cuyo origen es el vértice en edición (serán los vértices que aparecen en el cuadro *Nodos destino*).

Al igual que con los grafos no dirigidos se dispone de la posibilidad de añadir/suprimir aristas seleccionando un vértice y añadiéndolo o quitándolo del cuadro correspondiente.

Para confirmar los cambios se pulsará al botón *Aceptar*.

5.2.10 Insertar vértices

Para añadir un vértice, el usuario tendrá que pulsar sobre el botón *Insertar Nodo* de la barra de iconos (botones).

Una vez pulsado este botón, se podrá comprobar que el botón queda seleccionado y el usuario únicamente tendrá que pulsar con el botón izquierdo del ratón en la zona del lienzo donde desea situar el vértice.

5.2.11 Insertar arista

Para añadir una arista entre dos vértices del grafo, el usuario dispone de dos posibilidades distintas.

- **Edición del nodo:** Como ya se ha explicado en la sección *Editar Nodo*.
- **Manualmente sobre el lienzo:** Para ello, el usuario tendrá que pulsar sobre el botón *Insertar Arista* de la barra de iconos (botones).

Para crear la arista el usuario únicamente tendrá que pulsar con el botón izquierdo del ratón sobre el vértice que será origen de la arista, y sin soltar dicho botón, mover el ratón hasta colocar el puntero sobre el vértice que será destino de la arista, una vez encima del vértice, bastará con soltar el botón del ratón para generar la arista. Durante el movimiento del ratón se hace visible la arista que se desplazará por el lienzo siguiendo al puntero del ratón, hasta que llegue al vértice deseado.

5.2.12 Eliminar nodo

Para eliminar un vértice del grafo, el usuario dispone de dos maneras distintas para realizarlo. Hay que recordar que cuando se elimine un vértice del grafo, a su vez se eliminarán todas las aristas que inciden en él.

- Sobre el lienzo, con el botón derecho del ratón: El usuario pulsará con el botón derecho del ratón sobre el nodo que se desea eliminar, y pulsará en la opción *Eliminar Vértice* del menú que aparecerá junto al puntero del ratón. Una vez seleccionada esta opción el vértice y todas las aristas que lleguen hasta él desaparecerán.

- Sobre el lienzo, con el botón izquierdo del ratón: Para ello, el usuario tendrá que pulsar sobre el botón *Eliminar Vértices* de la barra de iconos (botones).

Una vez pulsado este botón, se podrá comprobar que el botón queda seleccionado. Para eliminar el vértice, el usuario únicamente tendrá que pulsar con el botón izquierdo del ratón sobre el vértice que desea borrar, una vez realizado esto, se podrá comprobar que el vértice y las aristas que incidían sobre él han sido borrados.

5.2.13 Suprimir arista

Para eliminar una arista del grafo, el usuario dispone de dos maneras distintas para realizarlo.

- Sobre el lienzo, con el botón derecho del ratón: El usuario pulsará con el botón derecho del ratón sobre arista que se desea eliminar, y pulsará en la opción *Eliminar Arista* del menú que aparecerá junto al puntero del ratón.
- Sobre el lienzo, con el botón izquierdo del ratón: Para ello, el usuario tendrá que pulsar sobre el botón *Eliminar Arista* de la barra de iconos (botones).

Una vez pulsado este botón, se podrá comprobar que el botón queda seleccionado. Para eliminar la arista, el usuario únicamente tendrá que pulsar con el botón izquierdo del ratón sobre la arista que desea borrar, una vez realizado esto, se podrá comprobar que la arista ha sido borrada.

5.2.14 Cambiar las opciones

Esta ventana nos permite modificar en cualquier momento la configuración de la aplicación, modificándose tanto para el grafo abierto, como para los que se abran a continuación.

Para acceder a la ventana de opciones el usuario deberá pulsar en el menú Herramientas y seleccionar *Opciones*. O pulsar sobre el botón *Opciones* de la barra de iconos (botones).

Una vez realizado esto se nos abrirá la ventana de opciones:

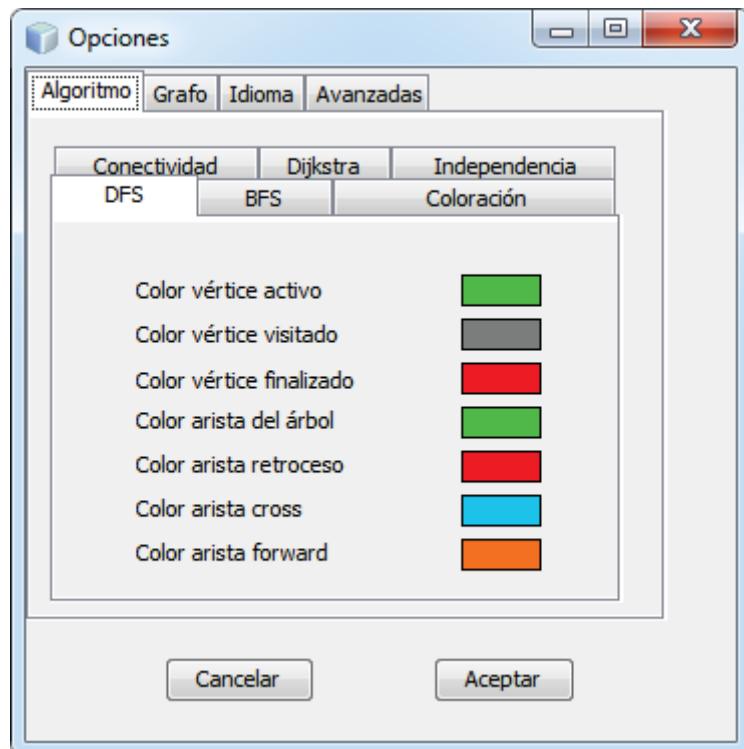


Figura 59. Ventana de opciones

Podremos ver en la mayoría de las pestañas de las opciones algo similar a esto:



Figura 60. Detalle de un elemento de la ventana de opciones

En este ejemplo podemos comprobar que el color para el vértice activo es el verde. Si el usuario decide cambiar dicho color solamente tendrá que hacer un clic con el botón izquierdo del ratón sobre el cuadro con color. Una vez hecho esto nos aparecerá una pantalla similar a la siguiente:

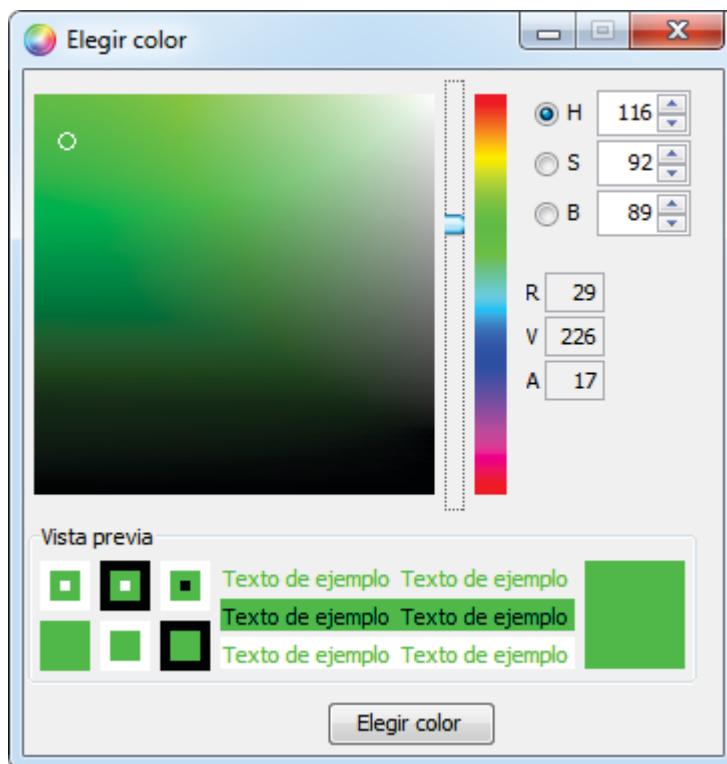


Figura 61. Ventana de selección de color

En ella el usuario podrá desplazar la barra vertical hasta colocarla en una tonalidad de color deseada y a continuación pulsar sobre el color exacto en el cuadro de color de la izquierda, una vez realizado esto, el usuario pulsará sobre el botón elegir color y el color seleccionado aparecerá seleccionado en las opciones.

El usuario podrá comprobar que según se va moviendo la barra de selección de color, los valores de la izquierda se van modificando. En estos cuadros de valor, el usuario podrá introducir manualmente el valor del color deseado (si lo conociese).

Pulsando sobre los botones de selección H, S, B se modificará el formato del cuadro de color de la izquierda, visualizándose de manera distinta.

A continuación iremos explicando cada una de estas partes que componen la ventana de opciones:

- **Algoritmo:** Configura cada una de las opciones requeridas en cada algoritmo, podemos ver que hay seis subpestañas:

- **DFS:** Nos permite configurar el color del vértice activo, visitado y candidato así como el color de la arista, durante la ejecución del algoritmo de búsqueda DFS
- **BFS:** Igual que el algoritmo de búsqueda DFS.
- **Coloración:** Nos permite seleccionar hasta 20 colores diferentes para elegir durante los algoritmos de coloración.
- **Conectividad:** Nos permite seleccionar los colores de vértice de corte y predecesor, de arista puente y de retroceso, así como el color de los vértices y aristas visitados durante la ejecución del algoritmo de conectividad (búsqueda de bloques, vértices de corte y puentes).
- **Dijkstra:** Nos permite configurar el color de los vértices visitados, candidatos y activo, así como el color de las aristas, de la distancia y del peso.
- **Independencia:** Podremos configurar el color del vértice activo, de los vértices vecinos y de los vértices que no estarán disponibles durante la ejecución del algoritmo de Independencia.
- **Grafo:** Configura la visualización del grafo antes de la ejecución de algún algoritmo. Podremos seleccionar el color del vértice, de la arista y del vértice seleccionado, así como el grosor tanto del vértice como de la arista.
- **Idioma:** La aplicación AIG esta internacionalizada. Esto quiere decir que seleccionando cualquiera de los idiomas disponibles esta ventana, todos los textos que aparezcan en la aplicación aparecerán en dicho idioma. Actualmente los idiomas disponibles son español e inglés.
- **Avanzadas:** Nos permite seleccionar el número de pasos hacia atrás que se pueden dar durante la ejecución de cualquier algoritmo.

5.2.15 Imagen de fondo

A la hora de trabajar con el grafo, el usuario puede insertar una imagen de fondo que le permita colocar los vértices de un nodo sobre unos puntos específicos, por ejemplo, el usuario podría cargar como imagen de fondo un mapa, colocar los vértices sobre ciudades y añadir aristas entre las ciudades.

Para poder cargar una imagen de fondo, el usuario solamente tendrá que pulsar sobre el menú *Herramientas* y seleccionar *Imagen de Fondo* o pulsar sobre el botón *Imagen de Fondo* de la barra de iconos (botones).

En ese momento aparecerá una ventana similar a la de cargar grafo, donde el usuario tendrá que seleccionar el archivo deseado. Una vez seleccionado y pulsado el botón de *Abrir*, la imagen seleccionada aparecerá como fondo del lienzo. Si el archivo seleccionado no es un archivo de imagen, no se mostrará ninguna imagen de fondo.

5.2.16 Estadísticas

El usuario en un determinado momento puede desear obtener una información de cómo será la ejecución de varios algoritmos sobre un mismo grafo. Para poder obtener esta información, deberá pulsar en el menú *Herramientas* y a continuación seleccionar *Estadísticas*.

Se nos mostrará una ventana con los algoritmos existentes para realizar las estadísticas. Esta ventana variará dependiendo del proyecto seleccionado, Coloración o Búsqueda, en el cuadro de *Selección de Proyecto* de la ventana principal de la aplicación.

Las ventanas con los algoritmos que se mostrarán son las siguientes:

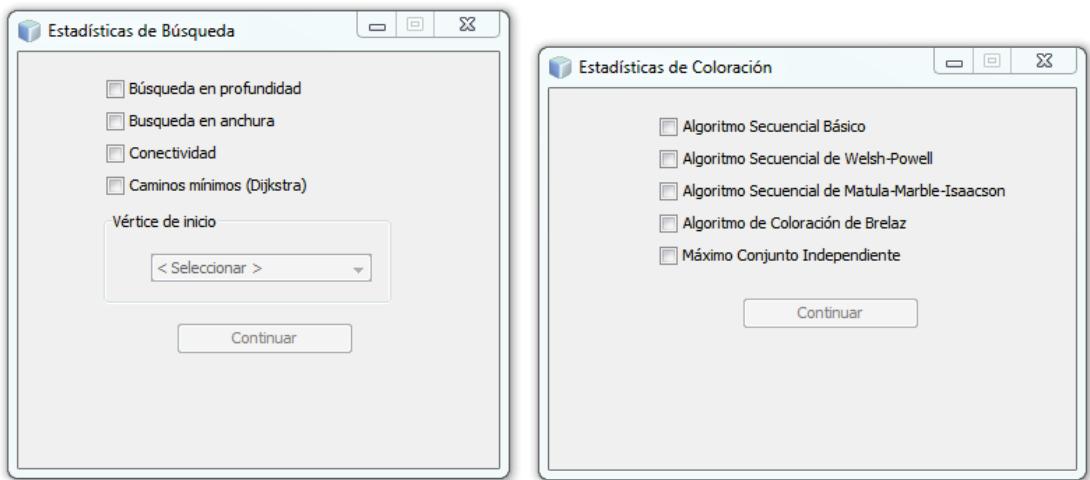


Figura 62. Ventanas de selección de algoritmos para la generación de estadísticas

En estas ventanas podemos observar que podemos seleccionar cualquiera de los algoritmos de cada proyecto para obtener sus estadísticas respecto a un grafo.

El archivo de estadísticas se generará en formato Excel. Una vez seleccionados los algoritmos y el formato de salida el usuario pulsara sobre el botón *Continuar*, abriéndose una ventana donde el usuario deberá indicar el nombre y la ubicación donde se alojará el archivo.

El archivo generado será el mismo que se genera cuando se exporta un grafo, al que se añadirán partes correspondientes a los algoritmos seleccionados.

Para cada uno de los algoritmos se mostrará información relativa a dicho algoritmo, número de pasos, tiempo de ejecución, estado final del grafo, etc.

5.2.17 Matriz de adyacencia

El usuario dispone de la posibilidad de obtener la matriz de adyacencia del grafo. Para ello deberá pulsar en el menú Herramientas y a continuación en Matriz de Adyacencias o bien pulsar sobre el botón *Matriz de Adyacencia* de la barra de iconos (botones). Una vez hecho esto se mostrará una ventana con la matriz de adyacencias del grafo activo.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	1	0	0	1	0	1	0	0	1	1	1	1
2	0	0	1	1	1	1	1	0	0	0	1	1	1	1
3	1	1	0	0	0	0	0	0	0	1	1	0	0	0
4	0	1	0	0	0	1	1	1	1	1	0	1	0	0
5	0	1	0	0	0	0	1	0	0	1	0	0	0	0
6	1	1	0	1	0	0	0	0	0	0	0	0	0	1
7	0	1	0	1	1	0	0	0	1	1	0	0	0	0
8	1	0	0	1	0	0	0	0	1	1	1	0	0	0
9	0	0	0	1	0	0	1	1	0	1	0	0	0	0
10	0	0	1	1	1	0	1	1	1	0	1	0	0	0
11	1	1	1	0	0	0	0	1	0	1	0	0	0	0
12	1	1	0	1	0	0	0	0	0	0	0	0	0	0
13	1	1	0	0	0	1	0	0	0	0	0	0	0	0
14	0	0	1	1	1	0	1	1	0	1	0	1	1	1
15	0	1	1	0	1	1	1	1	1	0	1	0	1	1
16	1	0	0	1	0	0	0	0	0	1	1	1	0	0

Figura 63. Ventana de la matriz de adyacencia

La matriz de adyacencia muestra la conectividad entre los vértices del grafo, apareciendo un 1 si existe una arista entre los vértices y un 0 si no existe dicha arista.

5.2.18 Matriz de ponderación

El usuario dispone de la posibilidad de obtener la matriz de ponderación del grafo. Para ello deberá pulsar en el menú Herramientas y a continuación en Matriz de Ponderación o bien pulsar sobre el botón *Matriz de Ponderación* de la barra de iconos (botones). Una vez hecho esto se mostrará una ventana con la matriz de ponderación del grafo activo.

La matriz de ponderación muestra el peso de las aristas existentes entre dos vértices del grafo. Los valores existentes se pueden modificar, para ello, el usuario deberá ir a la celda correspondiente, realizar un doble clic sobre ella y a continuación escribir el valor deseado. Una vez realizados todos los

cambios, el usuario pulsará el botón *Aceptar* para salvar dichas modificaciones.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1			56			7		22			11	41	27			86
2			86	66	48	67	13				76	42	96			40
3	56	86							51	27			61	48		
4		66				19	82	38	9	79		42		4		43
5		48					68			58				33	9	
6	7	67		19									80		85	
7		13		82	68				30	95				73	38	
8	22			38					41	13	52			86	13	
9			9			30	41			41					39	
10			51	79	58		95	13	41		35			92		54
11	11	76	27				52		35					56	5	
12	41	42		42										43		40
13	27	96			80									91	5	
14			61	4	33		73	86		92		43	91			
15		40	48		9	85	38	13	39		56		5			13
16	86			43						54	5	40				13

Figura 64. Ventana de la matriz de ponderación

5.2.19 Grafos tipo

Hay determinados grafos que son utilizados frecuentemente en Teoría de Grafos por tener propiedades características o por ser base de otros grafos. Debido a su alto uso se le ofrece al usuario la posibilidad de utilizarlos sin tener que crearlos manualmente. Para ello el usuario deberá pulsar en el menú *Grafos* y seleccionar en grafo deseado. A continuación listaremos los grafos que el usuario podrá elegir:

- Nulo
- Completo (K_n)
- Bipartito completo ($K_{r,s}$)
- Tripartito completo ($K_{r,s,t}$)

- Ciclo
- Grafo de Heawood
- Estrella
- Rueda
- Grafos de Rejilla: Rectangular y Triangular
- Cubo
- Grafos Platónicos: Tetraedro, Hexaedro, Octaedro, Icosaedro y Dodecaedro
- Grafo de Herschel
- Grafos Harary
- Enlazados: Enlazado $L_{n,r}$ y Enlazado $L_{n,r,s}$
- Grafo de Petersen
- Grafo de Grötzsch
- Grafo Tutte

5.3 Ejecución

Como ya hemos comentado antes, la funcionalidad principal de la aplicación es la de ejecutar ciertos algoritmos de Búsqueda o de Coloración sobre un grafo activo en la aplicación.

Para poder realizar esto, una vez que ya se tiene cargado un grafo sobre la aplicación, el usuario pulsará sobre el ícono de *Ejecución* en la barra de iconos o en el Menú *Herramientas* y en ejecución.

Una vez hecho esto aparecerá una ventana donde el usuario podrá seleccionar el algoritmo a ejecutar. Esta pantalla dependerá del tipo de Proyecto que se haya tenga seleccionado en este momento. A continuación mostraremos las pantallas de selección de algoritmos:

Selección de algoritmos de Coloración:

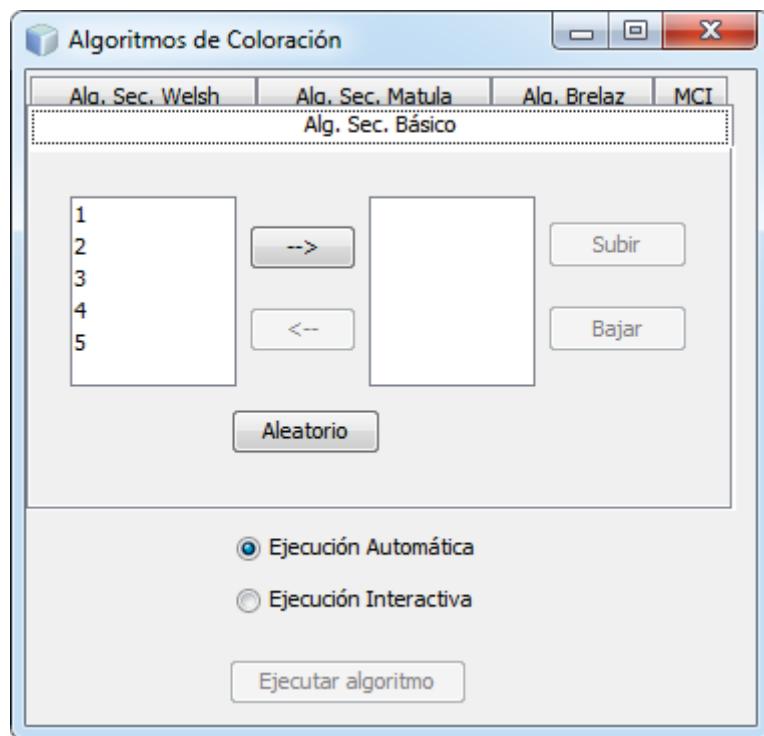


Figura 65. Ventana de selección de algoritmos de coloración

El usuario podrá elegir ejecutar uno de los cinco algoritmos posibles, estos son: Algoritmo Secuencial Básico (Aleatorio), Algoritmo Secuencial de

Welsh, Algoritmo Secuencial de Matula, Algoritmo de Brelaz y el Algoritmo de Independencia (MCI).

Para poder elegir uno de estos algoritmos el usuario deberá pulsar sobre la pestaña correspondiente y a continuación, seleccionar si se desea que la ejecución sea Automática o Interactiva.

En el caso del algoritmo secuencial Básico, se ofrece al usuario la posibilidad de elegir el orden en el que se van a tratar los nodos. Si el usuario no desea ningún orden en especial, pulsará el botón *Aleatorio* para que los nodos se coloquen en un orden determinado por la aplicación. Si el usuario selecciona que la ejecución sea automática hasta que todos los vértices del grafo no estén colocados en un orden no se podrá proceder a la ejecución de este algoritmo. Sin embargo si la ejecución es interactiva no hace falta seleccionar ningún orden determinado, ya que este orden lo irá determinando el usuario durante la ejecución.

Si el usuario ha seleccionado el algoritmo de Brelaz, una vez que haya pulsado el botón de *Ejecutar Algoritmo* en la ventana de información, podremos ver que aparece una nueva pestaña con información relativa al algoritmo, útil para poder comprender la ejecución de este algoritmo:

Vértice	$g(v)$	$gs(v)$	Orden	Color
V1	2	0		
V2	3	0		
V3	2	0		
V4	3	0		
V5	3	0		
V6	2	0		
V7	3	0		

Figura 66. Detalle de la tabla del algoritmo de Brelaz

Esta tabla nos mostrará información relativa a cada vértice, durante la ejecución de cada paso del algoritmo de Brelaz, en ella se nos muestra además del vértice, el grado del vértice ($g(v)$), el grado de saturación del vértice ($gs(v)$), el orden en que ha sido coloreado y el color que se le ha asignado.

Selección de algoritmos de Búsqueda:

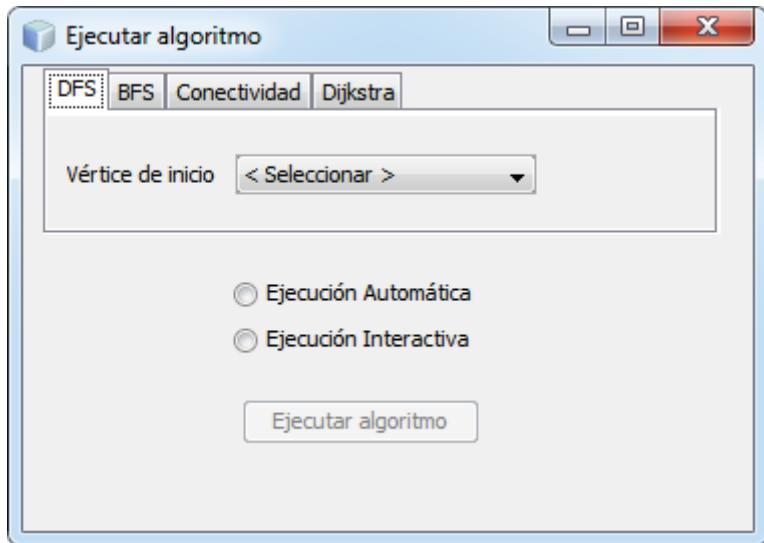


Figura 67. Ventana de selección de algoritmos de búsqueda

El usuario podrá elegir ejecutar uno de los cuatro algoritmos posibles, estos son: Algoritmo DFS, Algoritmo BFS, Conectividad (algoritmo de Bloques) y Algoritmo de Dijkstra.

Para poder elegir uno de estos algoritmos el usuario deberá pulsar sobre la pestaña correspondiente y a continuación, seleccionar si se desea que la ejecución sea Automática o Interactiva.

Dependiendo del algoritmo seleccionado el usuario deberá indicar una información suplementaria, por ejemplo, para los algoritmos DFS y BFS, el usuario deberá indicar cuál es el vértice e inicio.

En el algoritmo de Dijkstra a parte de tener que seleccionar el vértice de inicio del algoritmo, nos encontramos con dos botones adicionales:

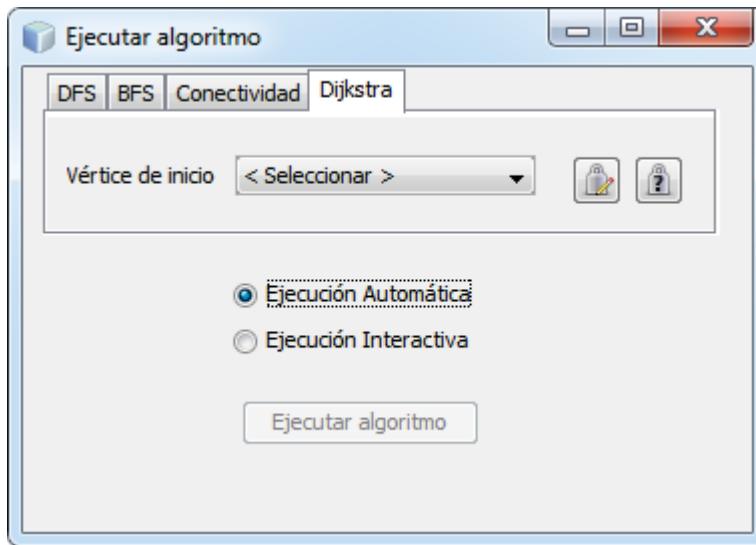


Figura 68. Pestaña del algoritmo de Dijkstra en la ventana de selección de algoritmos

Estos dos botones son para *Editar la ponderación* y *Generar una Ponderación Aleatoria*.

- ☞ El usuario podrá introducir manualmente el peso de las aristas existentes para este algoritmo.
- ☞ El usuario obtendrá una ponderación aleatoria del peso de las aristas existentes. Dicho peso podrá ser modificado por el usuario.

Una vez que el usuario ha seleccionado el algoritmo y la información necesaria y ha decidido la forma de ejecución, pulsará sobre el botón *Ejecutar Algoritmo* para comenzar la ejecución del mismo.

5.3.1 Ejecución automática

Una vez que el usuario ha seleccionado el algoritmo deseado, ha suministrado la información previa necesaria, y ha seleccionado ejecución automática, la pantalla de ejecución desaparecerá y el usuario podrá comprobar que se han activado los botones de ejecución del algoritmo situados en la parte central de la zona de información de la aplicación.

La funcionalidad de los botones es la siguiente:

- ◀ **Paso hacia atrás.** Si durante la ejecución de un algoritmo paso a paso, queremos volver al paso anterior, se pulsará este botón.
- ▶ **Iniciar (Play).** Si el usuario desea ejecutar un algoritmo automático hasta que este finalice pulsará el botón de *Play*.
- ⏸ **Parar (Pause).** Una vez que el usuario ha pulsado el botón *Play*, puede parar la ejecución del algoritmo pulsando el botón de *Pause*.
- ▶ **Paso hacia delante.** La manera de avanzar en la ejecución de un algoritmo paso a paso es pulsando este botón.

Durante la ejecución del algoritmo el usuario podrá ir alternando la manera de visualizar la ejecución del algoritmo automático.

El usuario puede pulsar el botón *Play* y esperar a que finalice el algoritmo. También puede ir paso a paso, pulsando el botón *Paso hacia delante* para ver cómo va evolucionando el grafo.

El usuario dispone de la posibilidad de parar el algoritmo si ha pulsado previamente el botón *Play* y salvar la configuración del grafo para continuar posteriormente.

El usuario podrá comprobar que la ejecución del algoritmo ha finalizado observando que tanto el botón *Play* como el botón de *Paso hacia delante* están deshabilitados, además en el cuadro de información del algoritmo mostrará un texto indicando que dicho algoritmo ha llegado a su fin.

Existe la posibilidad de volver hacia atrás en cualquier momento, para que el usuario compruebe los pasos que ha ido realizando el algoritmo, por si tiene algún tipo de duda o para revisar los pasos realizados.

5.3.2 Ejecución interactiva

Una vez que el usuario ha seleccionado el algoritmo deseado y ha seleccionado ejecución interactiva, la pantalla de selección de algoritmo desaparecerá.

A partir de este momento, la ejecución interactiva comenzará. Para ello el usuario deberá pulsar con el botón derecho sobre el vértice o arista deseada (dependiendo del algoritmo) y una vez hecho esto aparecerá un menú,

donde se mostrarán activas las diversas posibilidades que puede seleccionar el usuario, dependiendo del algoritmo seleccionado.

A continuación veremos cómo realizar la ejecución interactiva para los algoritmos implementados.

5.3.2.1 Coloración: Algoritmo Secuencial Básico

En este algoritmo, el usuario tendrá que ir estableciendo los colores de los vértices, sin seguir ningún orden, únicamente tendrá que cumplir que los vértices vecinos no comparten color.

Para ello el usuario pulsará con el botón derecho sobre el vértice al que desea establecer el color, mostrándose el siguiente menú:

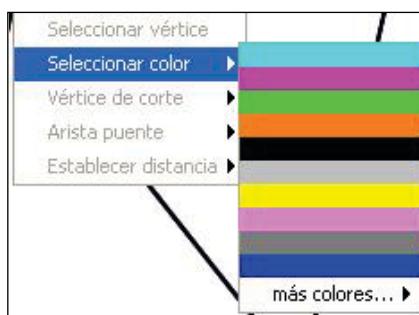


Figura 69. Menú contextual de selección de color (algoritmos de coloración)

Podemos comprobar que la única opción que puede elegir el usuario es *Seleccionar color*, una vez que el usuario coloca el puntero sobre dicha opción se nos mostrarán los diversos colores que puede elegir. Una vez que el usuario ha decidido el color deseado, pulsará sobre él y podrá comprobar que el color del nodo, se ha modificado por el color seleccionado.

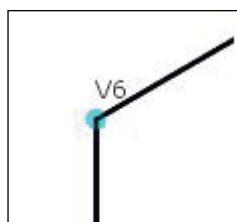


Figura 70. Detalle de un vértice coloreado

El usuario realizará esta operación, tantas veces como vértices existan, hasta que se haya coloreado todo el grafo.

Si en un determinado momento, el usuario selecciona un color que es el mismo que ya tiene un vecino del vértice seleccionado se nos mostrará la siguiente ventana de error:

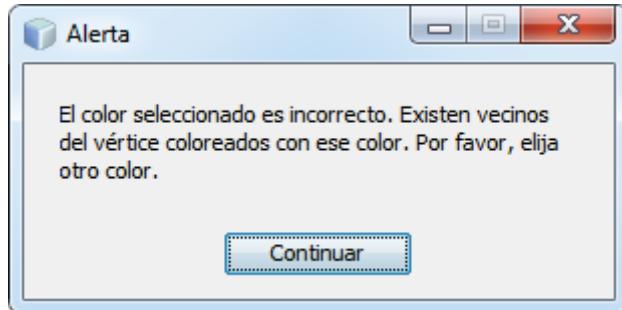


Figura 71. Ventana de alerta: color incorrecto

También puede darse el caso, que el usuario no seleccione el color óptimo para el nodo (siguiendo estrictamente el algoritmo ese color no sería el correcto), haciendo que no se utilicen los mínimos colores posibles para colorear el grafo. Cuando se de esta situación, se nos mostrará la siguiente pantalla:

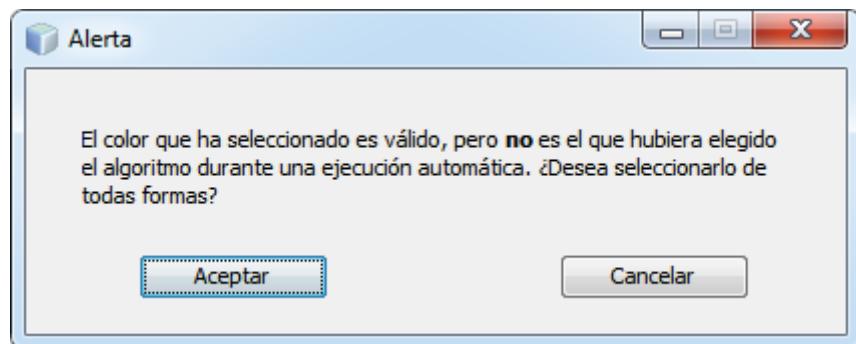


Figura 72. Ventana de alerta: color no adecuado

En este momento el usuario tiene la posibilidad de continuar la ejecución del algoritmo, utilizando el color seleccionado (aunque no realizará una coloración optima), pulsando sobre el botón *Aceptar*, o seleccionar otro color, para encontrar el color optimo, con lo cual, deberá pulsar sobre el botón *Cancelar*.

Existe la restricción de no poder modificar el color elegido a un vértice, con lo que si seleccionamos un color a un vértice al que ya le hayamos asignado uno, se nos mostrará la siguiente ventana:

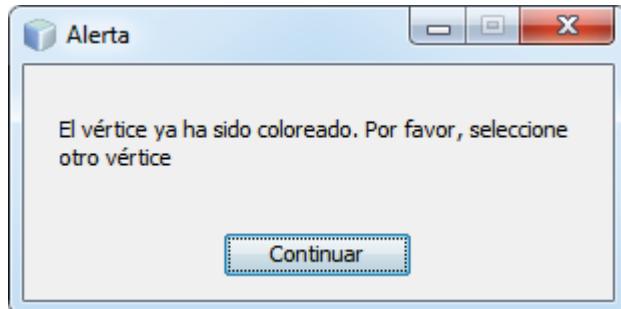


Figura 73. Ventana de alerta: vértice ya coloreado

Según vayamos asignando colores a los vértices, en el cuadro de información del algoritmo, se irá mostrando información relativa a la ejecución. Mostraremos un pequeño ejemplo de ello:

```
>> Le asignamos el color: 2 al no tener vecinos con ese color asignado

Los grupos de vértices con colores quedan de la siguiente manera (color --> vértices)
Color 0 --> V6, V2
Color 1 --> V1, V5
Color 2 --> V4

Nodo Activo 3
>> El vértice activo tiene vecinos
>> Le asignamos el color: 1 al no tener vecinos con ese color asignado

Los grupos de vértices con colores quedan de la siguiente manera (color --> vértices)
Color 0 --> V6, V2
Color 1 --> V1, V5, V3
Color 2 --> V4

>> No quedan vértices por colorear

FIN
```

Figura 74. Cuadro de información del algoritmo de coloración secuencial básico

5.3.2.2 Coloración: Algoritmo Secuencial de Welsh-Powell

En este algoritmo, es similar al Algoritmo Secuencial Básico, con la única salvedad que el orden de coloración de los vértices. En este caso, el usuario tendrá que ir estableciendo los colores de los vértices, siguiendo un orden, de mayor a menor grado, es decir, en función del número de vértices adyacentes, además tendrá que cumplir que los vértices vecinos no comparten color.

La dinámica de ejecución es exactamente igual que el Algoritmo Secuencial Básico. Se mostrarán las mismas pantallas de alerta, en las mismas situaciones (cuando seleccionemos un color que ya ha sido asignado a un vértice vecino, cuando el nodo ya haya sido coloreado y cuando se seleccione un color que no es el óptimo). Además de estas pantallas de alerta tendremos dos pantallas adicionales:

Si el usuario selecciona un color para un vértice cuyo grado es menor que el grado de un vértice todavía no coloreado nos mostrará la siguiente ventana:

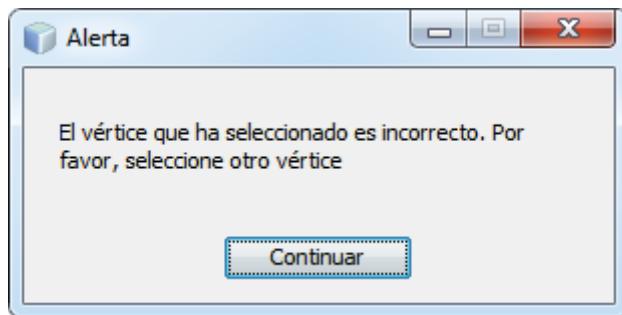


Figura 75. Ventana de alerta: vértice incorrecto

Como hemos dicho antes, el algoritmo de Welsh-Powell, colorea los vértices en función del grado de los mismos, ordenando los vértices de mayor a menor grado. Cuando dos vértices tienen el mismo grado, el algoritmo los ordena en función del orden de aparición del vértice en el grafo. Pero se ofrece al usuario la posibilidad de no seguir estrictamente esta norma, con lo que se mostrará el siguiente mensaje:

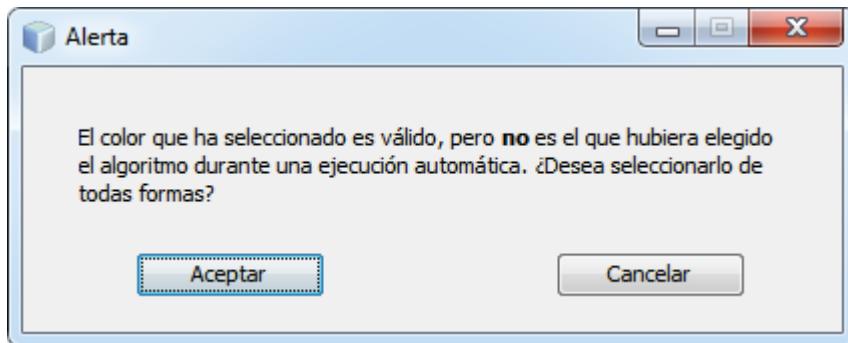


Figura 76. Ventana de alerta: vértice no adecuado

Se indica que el vértice seleccionado no sigue el orden que hubiera seguido el algoritmo, pero como tiene el mismo grado que el vértice que debería haberse elegido, se le ofrece la posibilidad de poder continuar con la ejecución, coloreando dicho nodo, pero se le indica que no es la solución que hubiese elegido el algoritmo si se hubiese realizado una ejecución automática.

5.3.2.3 Coloración: Algoritmo Secuencial de Matula-Marble-Isaacson

En este algoritmo, es similar al Algoritmo Secuencial Básico, con la única salvedad que el orden de coloración de los vértices. En este caso, el orden de coloración de los nodos es inverso del orden de selección. El orden de selección se realizará primero estableciendo el vértice de menor grado. El siguiente vértice será el vértice de menor grado del grafo resultante de quitar al vértice seleccionado anteriormente al grafo original. Este procedimiento se irá haciendo hasta que se hayan seleccionado todos los vértices.

Una vez obtenido el orden de selección de los nodos, comenzaremos a colorear el último vértice seleccionado, a continuación el siguiente, y así hasta que lleguemos a colorear al primer vértice que se seleccionó.

La dinámica de ejecución es exactamente igual que el Algoritmo Secuencial Básico. Se mostrarán las mismas pantallas de alerta, en las mismas situaciones (cuando seleccionemos un color que ya ha sido asignado a un vértice vecino, cuando el nodo ya haya sido coloreado y cuando se seleccione un color que no es el óptimo).

Si el usuario selecciona un color para un vértice, que no cumple con los requisitos del algoritmo, se nos muestra la siguiente ventana

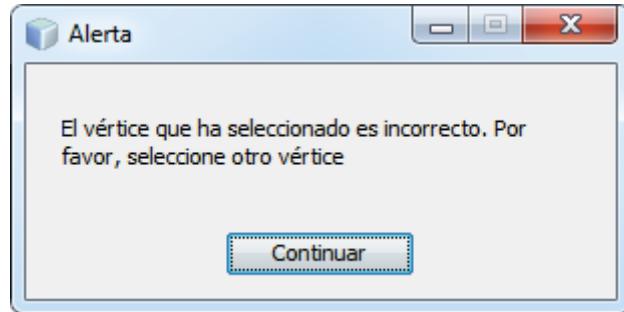


Figura 77. Ventana de alerta: vértice incorrecto

5.3.2.4 Coloración: Algoritmo de Coloración de Brelaz

En este algoritmo, el orden de coloración de los vértices depende del grado de los vértices ($g(V)$), y del grado de saturación o color de los vértices ($gs(V)$), y es determinado en tiempo de ejecución. Por lo que es bastante importante recurrir a la pestaña de Brelaz que aparecerá en la zona de información de la aplicación.

Árbol Brelaz				
Vértice	$g(v)$	$gs(v)$	Orden	Color
V1	2	0		
V2	3	0		
V3	2	0		
V4	3	0		
V5	3	0		
V6	2	0		
V7	3	0		

Figura 78. Detalle de la tabla del algoritmo de Brelaz

El grado de un vértice es el número de adyacentes del mismo, y el grado de saturación es el número de colores no repetidos usados en los adyacentes o vecinos.

La selección de nodos a colorear vendrá determinada primeramente por el grado de saturación, seleccionándose el vértice con mayor grado de

saturación, y en caso de que existan varios con el mismo valor, se elegirá de entre estos, el vértice que tenga mayor grado. Si aun así existiese más de un vértice a seleccionar, se elegiría dependiendo del nombre, con lo que se v1 y v4 tienen el mismo grado de saturación y el mismo grado, se coloreará primeramente a v1 antes que a v4.

Una vez que se ha coloreado un vértice, el grado de saturación de cada vértice no coloreado volverá a recalcularse. A parte de esto, se debe seguir con el criterio de no colorear un vértice con un color con el que ya se haya coloreado a un vértice vecino (adyacente).

Se mostrarán ventanas de alerta cuando:

- Se seleccione un color ya usado por un vértice vecino
- Cuando se seleccione un color para un vértice ya coloreado
- Cuando se seleccione un vértice que no es el correcto a colorear.
- Cuando se seleccione un color que no coincide con el color que se deberá seleccionar.

Debemos fijarnos que para la ejecución de este algoritmo no se permite seleccionar un color no óptimo, ni seleccionar un vértice que no sea el correcto, aunque coincidan en grado de saturación y grado del vértice.

5.3.2.5 Coloración: Algoritmo de Independencia MCI

En este algoritmo lo que se busca son conjuntos de independencia, y una vez que ya hayan sido establecidos todos los conjuntos independientes, la aplicación asignará un color a los vértices, coloreando los vértices de cada conjunto independiente con el mismo color.



Figura 79. Menú contextual en la ejecución del algoritmo MCI

El usuario deberá ir seleccionando los vértices con menor grado, para ir añadiéndolos al conjunto independiente, tal y como se muestra en la figura anterior, mediante el menú contextual.

Una vez que el usuario ha seleccionado el vértice correcto, podemos comprobar que el color tanto del vértice seleccionado, como sus vecinos ha cambiado.

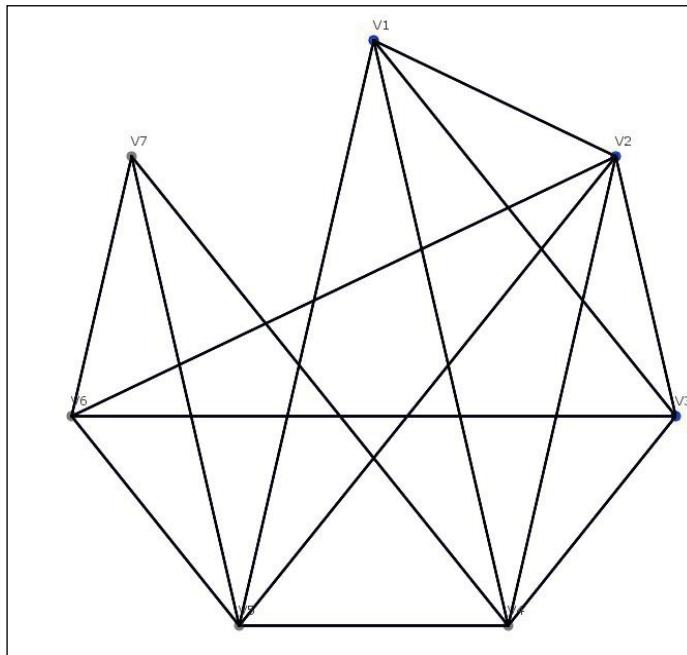


Figura 80. Detalle del lienzo en el algoritmo MCI, mostrando vértices no disponibles

Este cambio de color, nos indica que dichos vértices ya no pueden usarse para obtener el conjunto independiente, el vértice seleccionado, porque ya pertenece a él, y sus vecinos por que no pueden pertenecer al conjunto independiente ya que existe un vértice adyacente a ellos que ya pertenece a dicho conjunto independiente.

Con los vértices disponibles se sigue el mismo criterio de selección, dependiendo del grado de cada vértice, con la salvedad que para calcular el grado, no hay que tener en cuenta a los vértices no disponibles, con lo que para ver que vértice seleccionar, al grado real del vértice habrá que restarle el número de vértices vecinos que no pueden ser seleccionados para el conjunto independiente, porque ya hayan sido seleccionados o por que algún vecino pertenece al conjunto independiente.

Una vez que ya no quedan más vértices disponibles para formar parte del conjunto independiente se recalcula el grafo, mostrando los vértices que no forman parte de ningún conjunto independiente calculado previamente, y las aristas existentes entre estos vértices:

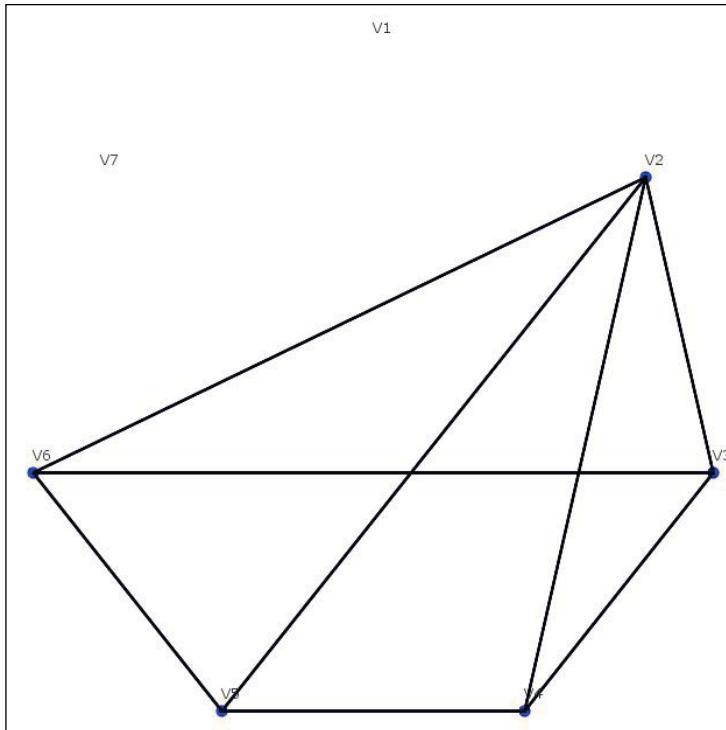


Figura 81. Detalle del lienzo en el algoritmo MCI, después de recalcular el grafo

Sobre el grafo resultante se volverá a calcular los vértices que forman parte del nuevo conjunto independiente.

Esto procedimiento se realizará hasta que no quede ningún vértice sin pertenecer a un conjunto independiente. Una vez que esto ocurra la aplicación, volverá a mostrar el grafo inicial, coloreando los vértices según su conjunto independiente. Podemos verlo en la figura siguiente.

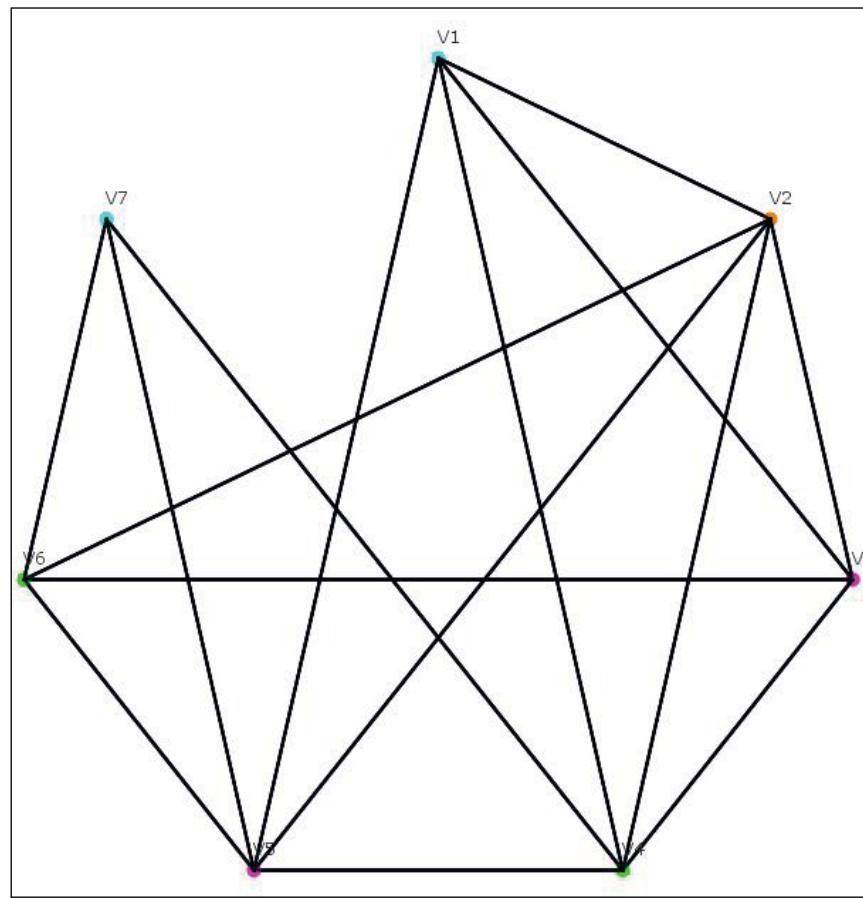


Figura 82. Detalle del lienzo en el algoritmo MCI, con los vértices coloreados

Debemos fijarnos que existen cuatro colores distintos, lo que quiere indicar que existen cuatro conjuntos independientes diferentes.

5.3.2.6 Búsqueda: Algoritmo DFS

En este algoritmo, el usuario tendrá que ir estableciendo los vértices que se van recorriendo, partiendo del vértice inicial seleccionado:

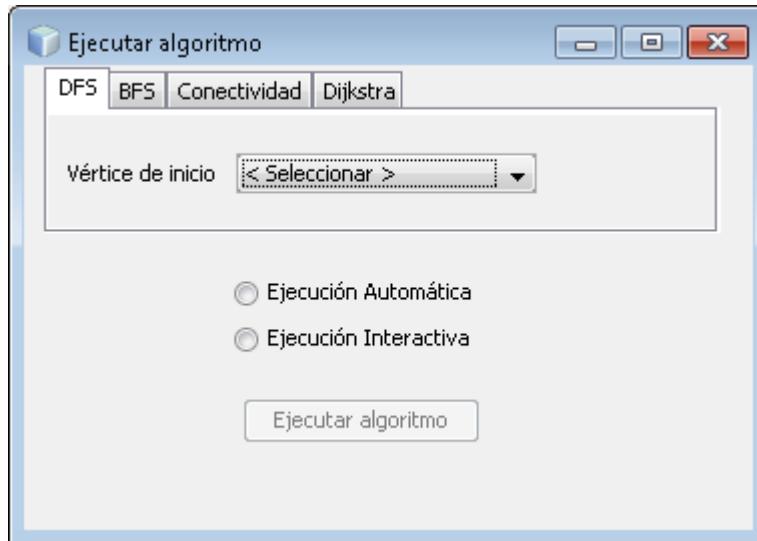


Figura 83. Ventana de algoritmo de búsqueda DFS. Selección de vértice de inicio

Para ello el usuario pulsará con el botón derecho sobre el vértice que desea seleccionar a continuación, mostrándose el siguiente menú:

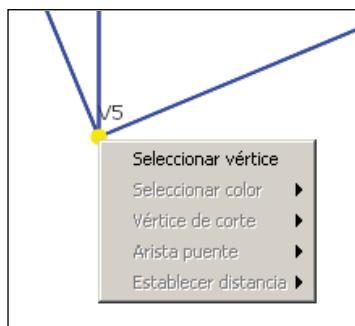


Figura 84. Menú contextual en la ejecución del algoritmo DFS

Podemos comprobar que la única opción que puede elegir el usuario es *Seleccionar vértice*. El usuario realizará esta operación para recorrer las aristas de cada vértice, tal y como indica el algoritmo, definiendo la búsqueda en profundidad sobre el grafo. Cuando un vértice es correctamente elegido, pasa a ser vértice actual (el vértice actual anterior

pasa a ser visitado) y se reconfiguran los nuevos vértices candidato (identificados por un círculo blanco en el centro).

A medida que se vayan recorriendo las aristas, éstas definirán su naturaleza:

- **Arista del árbol:** si el vecino al que alcanzan no ha sido aún visitado.
- **Arista de retroceso:** si el vecino al que alcanzan ha sido visitado.

En el caso de que el grafo sea dirigido, aparecen dos nuevos tipos de arista:

- **Arista forward:** si el vecino ha sido finalizado, en un orden de búsqueda posterior al del vértice activo.
- **Arista cross:** si el vecino ha sido finalizado, en un orden de búsqueda anterior al del vértice activo.

El usuario deberá seleccionar uno de los vértices candidato. Si selecciona un vértice diferente, se mostrará el correspondiente mensaje de error:

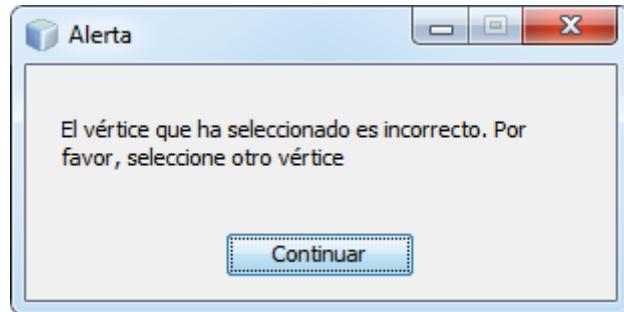


Figura 85. Ventana de alerta: vértice incorrecto

Dentro de los vértices candidato, el algoritmo siempre elegirá aquél que esté situado en la cima de la pila de ejecución. Si el usuario selecciona otro diferente de entre los candidatos, el sistema mostrará un mensaje advirtiendo de que no es el que hubiera elegido el algoritmo, a pesar de ser una elección igualmente válida:

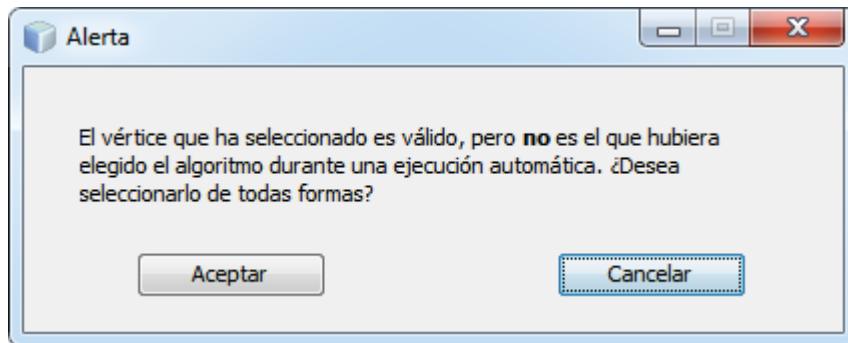


Figura 86. Ventana de alerta: vértice no adecuado

En este momento el usuario tiene la posibilidad de continuar la ejecución del algoritmo utilizando el vértice seleccionado, pulsando sobre el botón *Aceptar*, o seleccionar otro vértice, pulsando sobre el botón *Cancelar*.

Si en un momento dado no hay candidatos (porque todos los vecinos del vértice actual han sido visitados), pero aún quedan vértices por visitar, significará que el grafo no es conexo, por lo que el usuario deberá seleccionar uno de los restantes vértices (el algoritmo elegiría el de menos índice, aunque cualquiera de éstos sería válido). El algoritmo finalizará cuando todos los vértices hayan sido visitados y finalizados.

Según vayamos seleccionando vértices, en el cuadro de información del algoritmo, se irá mostrando información relativa a la ejecución.

5.3.2.7 Búsqueda: Algoritmo BFS

En este algoritmo, el usuario tendrá que ir estableciendo los vértices que se van recorriendo, partiendo del vértice inicial seleccionado:

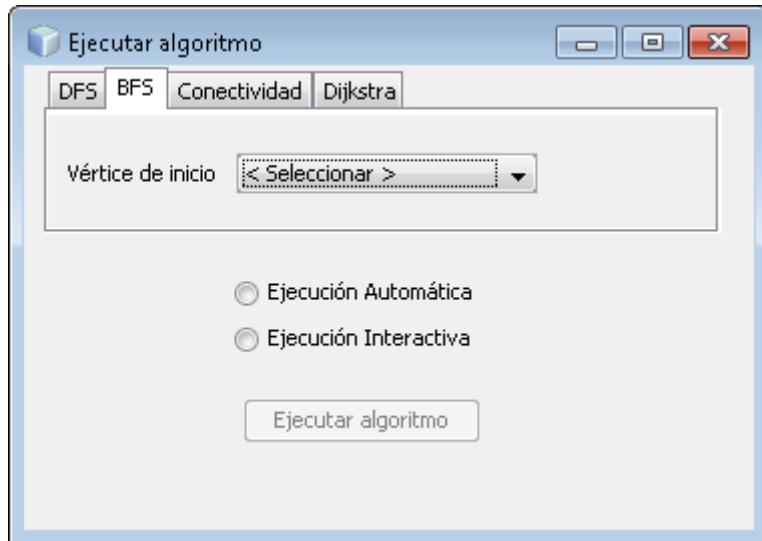


Figura 87. Ventana de algoritmo de búsqueda BFS. Selección de vértice de inicio

Para ello el usuario pulsará con el botón derecho sobre el vértice que desea seleccionar a continuación, mostrándose el siguiente menú:

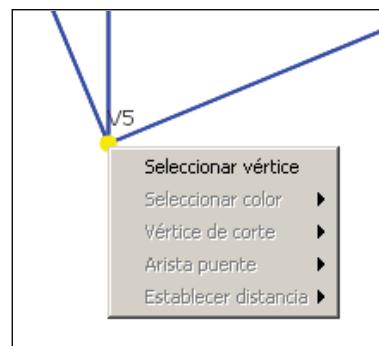


Figura 88. Menú contextual en la ejecución del algoritmo BFS

Podemos comprobar que la única opción que puede elegir el usuario es *Seleccionar vértice*. El usuario realizará esta operación, tantas veces como vértices existan, definiendo la búsqueda en anchura sobre el grafo. Cuando un vértice es correctamente elegido, pasa a ser vértice visitado (ya no será candidato).

A la hora de elegir vértice a seleccionar, en el grafo se mostrarán de un color diferente, aquéllos que son candidatos a ser seleccionados: no han sido seleccionados previamente, y son vecinos del vértice actual. El usuario deberá seleccionar uno de los vértices candidato. Si selecciona un vértice diferente, se mostrará el correspondiente mensaje de error:

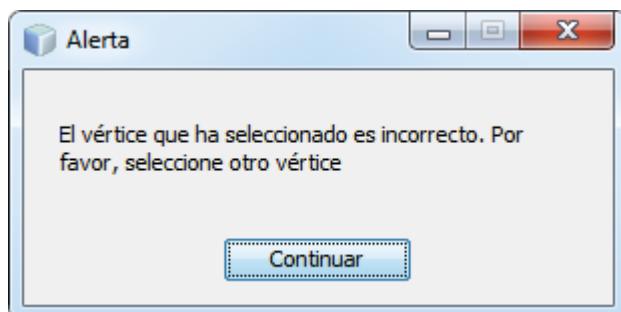


Figura 89. Ventana de alerta: vértice incorrecto

Dentro de los vértices candidato, el algoritmo siempre elegirá aquél de menor índice. Si el usuario selecciona otro diferente de entre los candidatos, el sistema mostrará un mensaje advirtiendo de que no es el que hubiera elegido el algoritmo, a pesar de ser una elección igualmente válida:

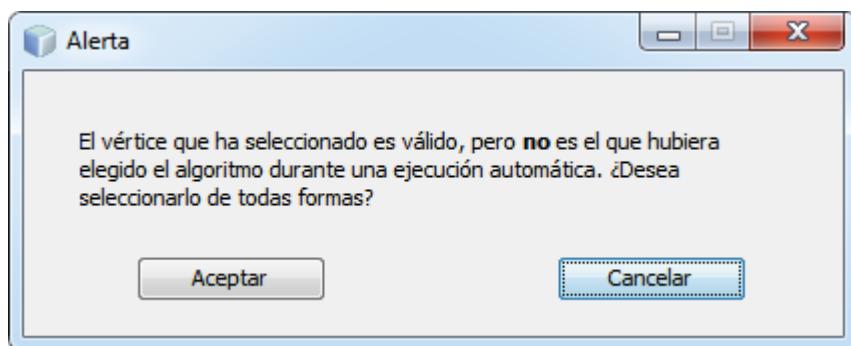


Figura 90. Ventana de alerta: vértice no adecuado

En este momento el usuario tiene la posibilidad de continuar la ejecución del algoritmo utilizando el vértice seleccionado, pulsando sobre el botón *Aceptar*, o seleccionar otro vértice, pulsando sobre el botón *Cancelar*.

Si todos los vecinos del vértice actual han sido visitados, el usuario deberá seleccionar el vértice actual tal y como lo haría el algoritmo: ir hacia atrás sacando elementos de la cola hasta encontrar un vértice con vecinos por

visitar. Si la cola se queda vacía y no hay candidatos, el algoritmo llega a su fin (aunque queden vértices por visitar).

Según vayamos seleccionando vértices, en el cuadro de información del algoritmo, se irá mostrando información relativa a la ejecución.

Una vez haya finalizado la ejecución del algoritmo, el usuario tiene la opción de visualizar el árbol de búsqueda resultante, utilizando el ícono a tal efecto.

5.3.2.8 Búsqueda: Conectividad

La aplicación nos permite realizar una ejecución interactiva para determinar los bloques en los que se puede descomponer un grafo. Recordemos que un bloque es un grafo no trivial, conexo y sin vértices de corte, por lo que el algoritmo generará bloques tras la detección de los vértices de corte.

El algoritmo de bloques utiliza el doble etiquetado (orden de búsqueda, función L) para determinar los puntos de corte (y, a partir de éstos, los bloques en que se descompone) y los puentes. El usuario deberá ir pulsando con el botón derecho sobre los vértices del grafo candidatos siguiendo el algoritmo, y éste irá recalculando los valores del doble etiquetado.

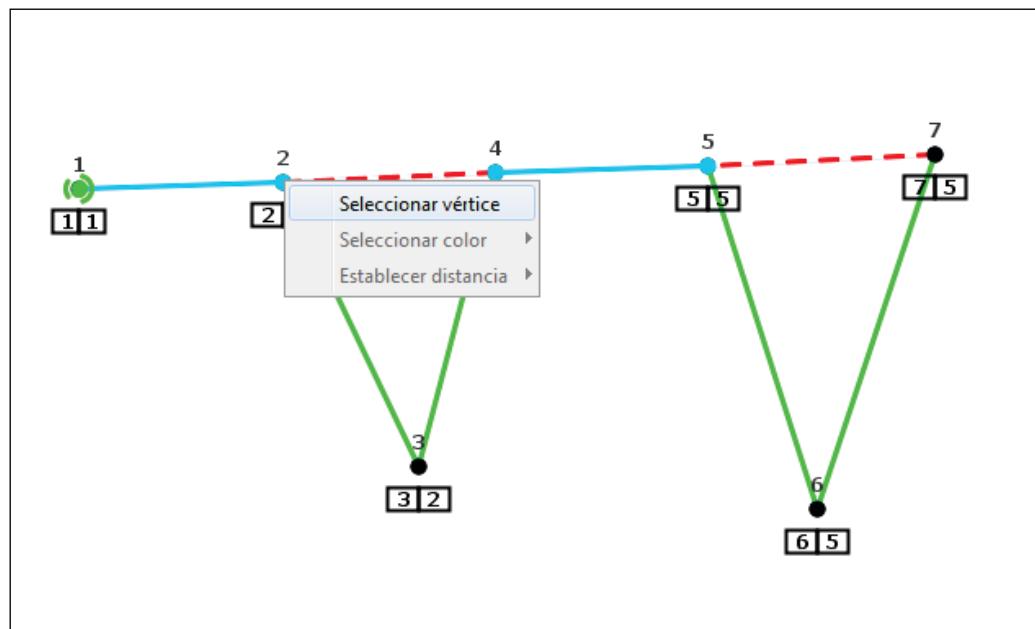


Figura 91. Menú contextual en la ejecución del algoritmo de Bloques

Si el usuario se ha equivocado al seleccionar un vértice (por no ser candidato), se nos mostrará la siguiente ventana:

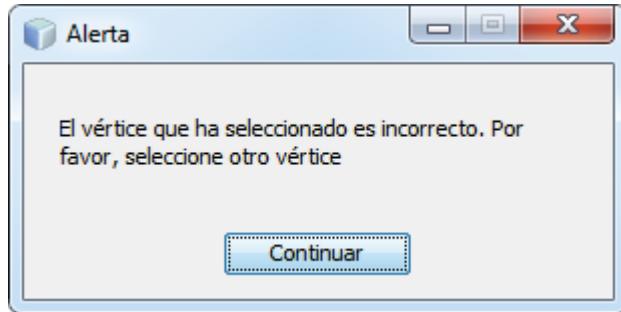


Figura 92. Ventana de alerta: vértice no determinado correctamente

Según vayamos seleccionando vértices, en el cuadro de información del algoritmo, se irá mostrando información relativa a la ejecución.

Cuando termina la ejecución del algoritmo, podemos ver los bloques en que se ha descompuesto el grafo, utilizando para ello los botones que se habilitan en la barra de iconos .

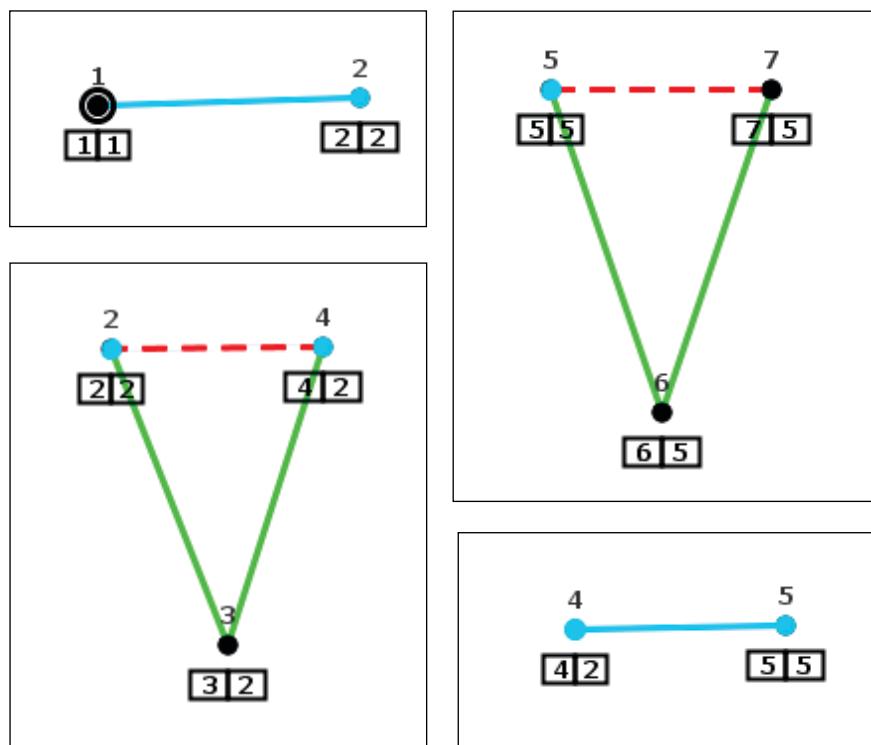


Figura 93. Bloques en que queda descompuesto el grafo inicial tras la ejecución

5.3.2.9 Búsqueda: Algoritmo de Dijkstra

En este algoritmo, el usuario tendrá que ir estableciendo los vértices que se van recorriendo, partiendo del vértice inicial seleccionado:

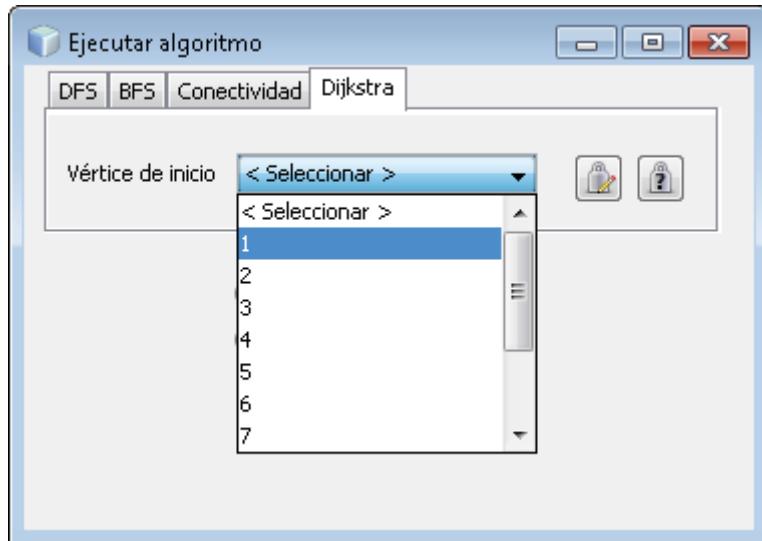


Figura 94. Ventana de algoritmo de Dijkstra. Selección de vértice de inicio

Además de seleccionar el vértice de inicio, el usuario tiene la posibilidad de visualizar la matriz de ponderación (pesos o distancias de las aristas del grafo) con el fin de editarla mediante el icono e incluso asignar valores aleatorios a las aristas (inicialmente establecidos en 1), mediante el icono .

A continuación el usuario pulsará con el botón derecho sobre el vértice que desea seleccionar pudiéndosele mostrar dos tipos de menú:

Seleccionar el siguiente vértice activo:

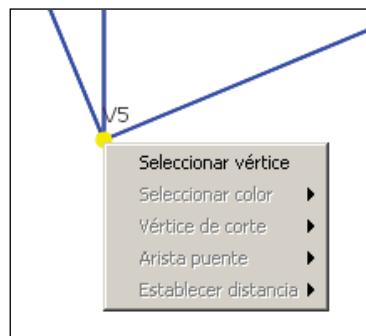


Figura 95. Menú contextual de Dijkstra. Seleccionar vértice

Si el vértice activo no tiene más candidatos, deberá seleccionar el siguiente vértice activo. El actual vértice activo parará a ser un vértice visitado (con su color correspondiente en el lienzo). Si, por el contrario, todavía existen más candidatos, la aplicación mostrará un error.

A la hora de elegir el siguiente vértice activo, se deberá seleccionar aquél cuya distancia al origen sea mínima (las distancias mínimas se visualizan en color rojo en la imagen siguiente):

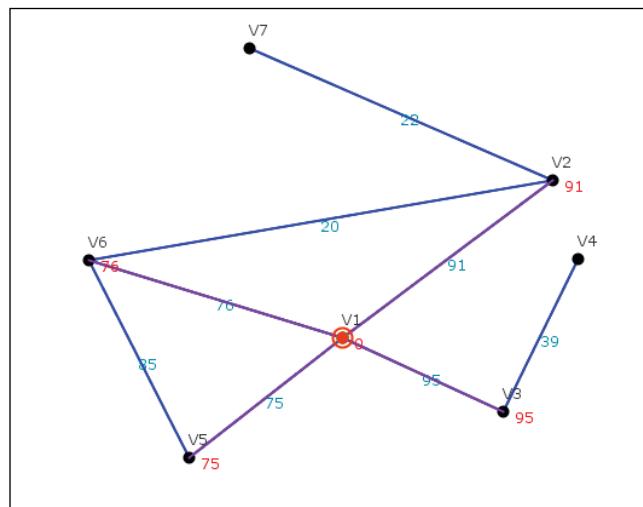


Figura 96. Detalle del lienzo en el algoritmo de Dijkstra

Si no fuera así, el sistema mostrará igualmente un error:

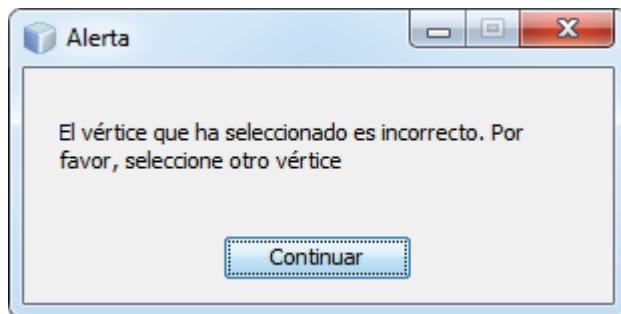


Figura 97. Ventana de alerta: vértice seleccionado incorrecto

Seleccionar la distancia mínima al vértice seleccionado:

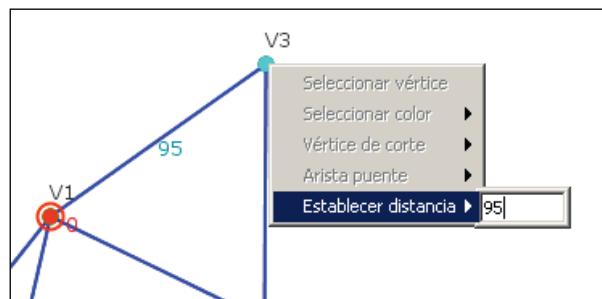


Figura 98. Menú contextual de Dijkstra. Establecer distancia

Si hace clic sobre un vértice candidato, podrá determinar la distancia. Para ello, deberá escribirla en el recuadro y pulsar la tecla <Intro>. Si introdujo la distancia correcta, el algoritmo continuará avanzando, y dicho vértice ya no será candidato para el vértice actual. Se establecerá la distancia mínima para dicho vértice. En caso contrario, la aplicación mostrará un error:

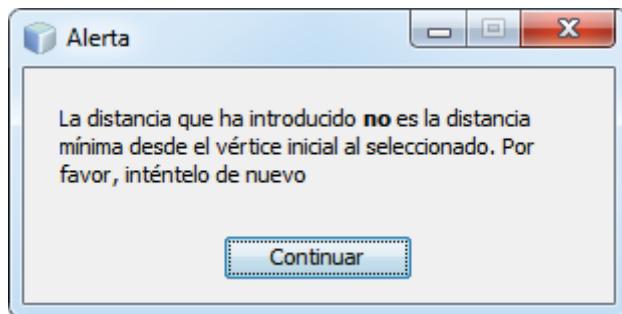


Figura 99. Ventana de alerta: distancia incorrecta

El usuario realizará esta operación, determinando la distancia mínima al origen, para cada vecino (y a su vez para cada vértice, en el orden correcto), definiendo la búsqueda de caminos mínimos de Dijkstra.

Según vayamos seleccionando vértices, en el cuadro de información del algoritmo, se irá mostrando información relativa a la ejecución.

6 CONCLUSIONES

Se ha desarrollado una aplicación que permite trabajar de forma sencilla con grafos. Se han realizado pruebas de rendimiento, y en equipos de gama media o media/baja, el sistema es capaz de *mover* con solvencia grafos de un tamaño importante (+500 nodos).

Este proyecto se ha llevado a cabo de forma conjunta con otro compañero. Nuestra intención desde un primer momento (hace ya 3 años) fue la de crear una herramienta completamente nueva, moderna y eficaz, que fuera el germen de algo más grande. Casi todo el desarrollo es aprovechable y se pueden agregar infinidad de algoritmos sin necesidad de cambiar la estructura de la aplicación significativamente.

Es nuestro deseo que este trabajo pueda ser continuado por otros alumnos y que la herramienta siga creciendo. Para ello hemos puesto mucha dedicación en el análisis y diseño y en que el código esté lo más estructurado posible, con el fin de que sea lo más sencillo posible de comprender.

Por otra parte, uno de los grandes empeños de nuestro tutor fue el de que fuera una herramienta con un trasfondo docente, y creo que lo hemos conseguido. Todos los algoritmos implementados contemplan un modo de ejecución interactivo, donde el usuario deberá ir *acertando* el siguiente paso a dar, para lo cual deberá conocer el funcionamiento del algoritmo. Además, la herramienta va explicando paso a paso el desarrollo del algoritmo.

Quiero destacar la complejidad que ha supuesto el llevar a buen puerto el desarrollo, compaginándolo en el caso de ambos con nuestra vida laboral, con lo que ello supone. Ha habido momentos agradables y momentos difíciles, en los que ha sido clave el apoyo que nos hemos prestado mutuamente para seguir adelante y no decaer en nuestro empeño. Es una lección que me llevo para mí: si perseveras no hay límites, sólo los que uno se impone.

7 BIBLIOGRAFÍA

Libros:

- G. Hernández. *Grafos: Teoría y Algoritmos*. Servicio de Publicaciones, Facultad de Informática, UPM, 2003.
- D. Jungnickel. *Graphs, Networks and Algorithms*. Springer, 2008.

Webs:

- <http://www.wikipedia.org>
- <http://docs.oracle.com/javase/6/docs/api/>
- <http://www.dma.fi.upm.es/java/matematicadiscreta/busqueda/>
- <http://www.personal.kent.edu/~rmuhamma/>
- <http://gaussianos.com/los-puentes-de-konigsberg-el-comienzo-de-la-teoria-de-grafos/>
- http://enciclopedia.us.es/index.php/Teor%C3%ADA_de_grafos
- <http://codebreakerscorp.wordpress.com>