4.1 Dado un árbol de expansión, resultante de un recorrido sobre un grafo no dirigido, ¿qué tipo de arcos (aparte de los del árbol) pueden aparecer si el recorrido es una búsqueda en profundidad o una búsqueda en anchura? ¿Qué arcos aparecerán si el recorrido (en profundidad o en anchura) es aplicado sobre un grafo dirigido?

4.2 Puesto que la búsqueda primero en profundidad es equivalente a un recorrido en preorden de un árbol, un programador decide implementar el equivalente a un recorrido en postorden (orden posterior). Para ello, modifica el procedimiento bpp haciendo que la marca de visitado sea establecida al final del mismo. Además, añade también al final una instrucción *escribir(v)* para que se muestre el orden de visita de los nodos. ¿Es correcta esta modificación para realizar un recorrido en orden posterior? En caso negativo, ¿cómo se solucionaría?

4.3 ¿Cuál es el número máximo de arcos que puede tener un grafo no dirigido sin ciclos? ¿Y cuál será para un grafo dirigido acíclico (GDA)?

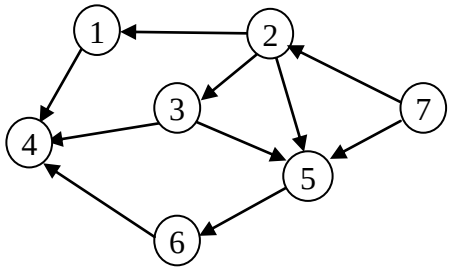
4.4 Implementar la prueba de aciclicidad en un grafo no dirigido. ¿Se puede hacer en un tiempo O(n)? **Sugerencia**: tener en cuenta el resultado del ejercicio anterior.

4.5 Modificar el procedimiento anterior para que, en caso de encontrar que existe un ciclo, devuelva en una lista los elementos que forman el ciclo encontrado. Se supondrá que tenemos un tipo ListaNodos, con operaciones *Anula (Lista)*, *InsertaCabeza (nodo, Lista)* e *InsertaCola (nodo, Lista)*, para añadir un nodo al principio y al final de la lista, respectivamente.

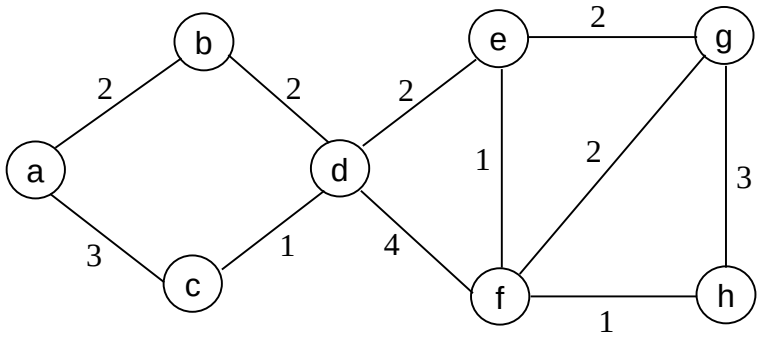
4.6 Una aplicación de planificación de viajes utiliza grafos dirigidos y etiquetados para representar un conjunto grande de ciudades y caminos o rutas aéreas entre ellas. En general, los grafos utilizados serán poco densos (es decir, el número de caminos será mucho menor que (número de ciudades)^2). Necesitamos que la consulta de las carreteras que salen de una ciudad y las que llegan a la misma sea rápida.

a) Proponer una estructura de representación que sea eficiente en cuanto a uso de memoria y a tiempo de ejecución de las dos operaciones de consulta anteriores. Tener en cuenta que necesitamos almacenar mucha información tanto para las ciudades (nombre, posición, país, ...) como para los caminos (tipo, distancia...).
b) ¿Parece adecuada la utilización de grafos dirigidos para este problema? Poner argumentos a favor y en contra.

4.7* Mostrar una ordenación topológica para el siguiente grafo dirigido acíclico. ¿La ordenación obtenida es única? En caso negativo, mostrar otra ordenación válida. (No es necesario explicar la ejecución del algoritmo.)



4.8 Suponer el grafo no dirigido de la siguiente figura. Mostrar:
   a) El bosque de expansión en profundidad, empezando en a y en d.
   b) El bosque de expansión en amplitud, empezando en a y en d.
   c) El árbol de expansión de costo mínimo utilizando el algoritmo de Prim.
   d) El árbol de expansión de costo mínimo utilizando el algoritmo de Kruskal. ¿Son iguales las soluciones obtenidas en ambos algoritmos? En caso contrario, ¿son válidas las distintas soluciones? ¿Por qué?



4.9 Escribir un procedimiento que realice una búsqueda primero en profundidad sobre un grafo dirigido, y que cada vez que encuentre una arista que no es del árbol calcule de qué tipo es (de avance, retroceso o cruce) y lo muestre por pantalla.

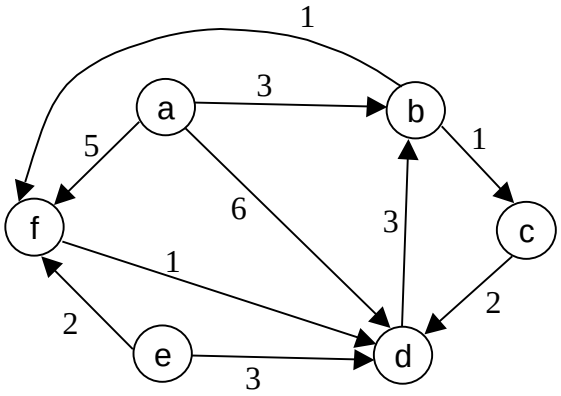
4.10 Diseñar un algoritmo para contar el número de ciclos simples existentes en un grafo no dirigido. ¿Cuál es el orden de complejidad de este algoritmo? ¿Cuántos ciclos pueden haber como máximo en un grafo cualquiera?

4.11 ¿Cómo se podría modificar el procedimiento bpa (utilizado en la búsqueda primero en anchura) para realizar un recorrido en profundidad, realizando un cambio mínimo en el algoritmo? **Sugerencia**: considerar la estructura de datos utilizada.

4.12 Modificar el procedimiento de búsqueda primero en profundidad para que cuente el número de árboles del bosque abarcador en profundidad. Además, se debe almacenar en un array *MA[1..n]* el número de árbol al que pertenece cada nodo. Se supone que el primer árbol generado es el 1, luego el 2 y así sucesivamente.

4.13 Demostrar que el algoritmo de Dijkstra no funciona cuando las aristas pueden tener costo negativo, aun cuando no existan ciclos en el grafo. **Sugerencia**: dar un contraejemplo de un grafo dirigido sin ciclos en el que el algoritmo de Dijkstra no dé el resultado correcto.

4.14 Utilizar el algoritmo de Dijkstra para encontrar los caminos más cortos que van desde el nodo *a* hasta los restantes nodos, en el siguiente grafo dirigido. Mostrar los valores S, D y P para todos los pasos de ejecución del algoritmo.

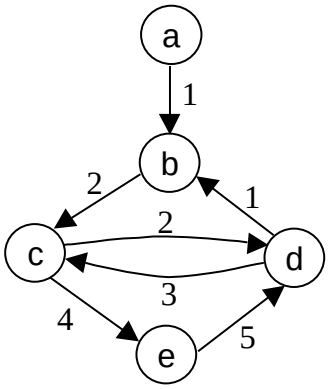


4.15 En general, dados dos nodos v, w en un grafo ponderado, puede existir más de un camino mínimo entre ambos (aunque necesariamente todos tendrán el mismo coste). Explicar cómo se debe modificar el algoritmo de Dijkstra para contar el número de caminos mínimos distintos existentes entre un nodo v y el resto de nodos.

4.16 Determinar cuál es el orden de complejidad del algoritmo para encontrar las componentes fuertemente conexas en un grafo dirigido. Suponer que el grafo tiene n nodos y a aristas, siendo a>n.

4.17 Aplicar el algoritmo para obtener las componentes fuertemente conexas para el grafo del ejercicio 4.14. A partir del resultado obtenido, mostrar el grafo reducido correspondiente.

4.18 Mostrar el resultado de la aplicación del algoritmo de Floyd sobre el siguiente grafo dirigido. Con el resultado del algoritmo, calcular cuál es el nodo más central del grafo.

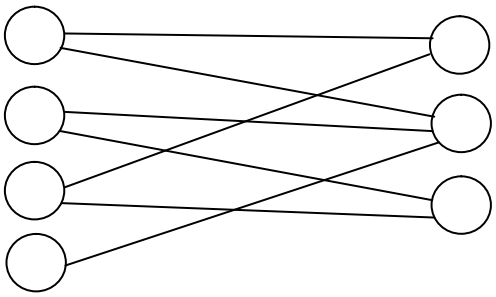


4.19 * Modificar el algoritmo de Dijkstra para que, además de calcular los caminos mínimos entre un nodo y los demás, también calcule en otro array *L* el número de aristas por las que pasa cada uno de los caminos mínimos. Decir únicamente las partes que se deben modificar del algoritmo.

4.21* Modificar el algoritmo de Dijkstra para que, además de calcular los caminos mínimos entre un nodo y los demás, también calcule otro array *L* de booleanos, indicando para cada nodo si su camino mínimo desde el origen es único o no. Decir únicamente las partes que se deben modificar del algoritmo.

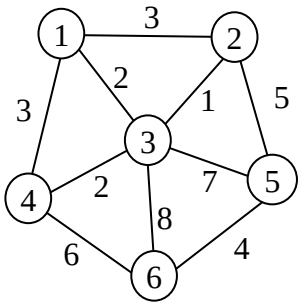
4.22 Demostrar que el grafo reducido de un grafo dirigido G cualquiera (el que representa cada componente conexa de G con un nodo) es siempre un GDA.

4.23 Un grafo no dirigido G = (V, A), se dice que es bipartido si V se puede particionar en dos subconjuntos V_1 y V_2, de manera que para toda arista (v, w) de A, el nodo v pertenezca a uno de los conjuntos (V_1 o V_2) y w pertenezca a la otra partición. Proporcionar un algoritmo con tiempo O(n+a) para comprobar si un grafo es bipartido o no. **Sugerencia:** usar una bpp para asignar una partición (1 o 2) a cada nodo visitado.



Ejemplo de grafo bipartido

4.24 * Aplicar el algoritmo de Kruskal sobre el siguiente grafo, mostrando el orden en que son añadidas las aristas a la solución.



Si aplicáramos el algoritmo de Prim, ¿podemos asegurar que se obtendría siempre la misma solución? ¿Podemos asegurar que el coste de la solución sería el mismo? ¿Por qué?

4.25 * Para desarrollar la especificación formal del TAD grafo dirigido y etiquetado disponemos de los siguientes conjuntos:

G	Conjunto de grafos dirigidos y etiquetados
M	Conjunto de nodos de los grafos
E	Conjunto de valores de las etiquetas en las aristas
N	Conjunto de naturales
B	Conjunto de booleanos
U	Conjunto de mensajes de error

Escribir la parte de sintaxis correspondiente a las operaciones que son los constructores del tipo. Poner también la sintaxis de alguna operación de modificación y otra de consulta sobre el grafo.

4.26 Suponer que estamos trabajando con GDA. ¿Es posible mejorar la eficiencia obtenida con el algoritmo de Dijkstra para este tipo de grafos? ¿Cómo sería la modificación de este algoritmo para conseguir la mejora? **Idea**: hacer uso de la ordenación topológica, para seleccionar los nodos entre los candidatos.

4.27 * En la especificación formal del TAD *GrafoNoDirigido*, por el método algebraico, tenemos definidos los siguientes constructores del tipo:

GrafoVacío: $\rightarrow$ **G** *// Crea un grafo vacío*
InsertaArista: Gx**N**$_1$x**N**$_2$ $\rightarrow$ **G** *// Añade la arista (N_1, N_2) al grafo*

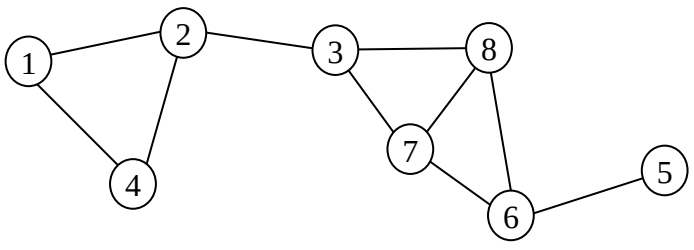
Mostrar las fórmulas que deberían aparecer en la parte semántica para la siguiente operación de consulta, que comprueba si la arista (N_1, N_2) pertenece al grafo o no:

ExisteArista: Gx**N**$_1$x**N**$_2$ $\rightarrow$ **B**

Se suponen los conjuntos:
G: Conjunto de grafos no dirigidos
N: Conjunto de nodos ($N=N_1=N_2$)
B: Conjunto de valores booleanos= {true, false}

4.28 Diseñar un algoritmo para encontrar el ciclo simple más largo en un grafo dirigido, pasando por un vértice dado **v**. ¿Cuál es la complejidad de este algoritmo?

4.29 * Mostrar los puntos de articulación del siguiente grafo. ¿Cuáles son las componentes biconexas?



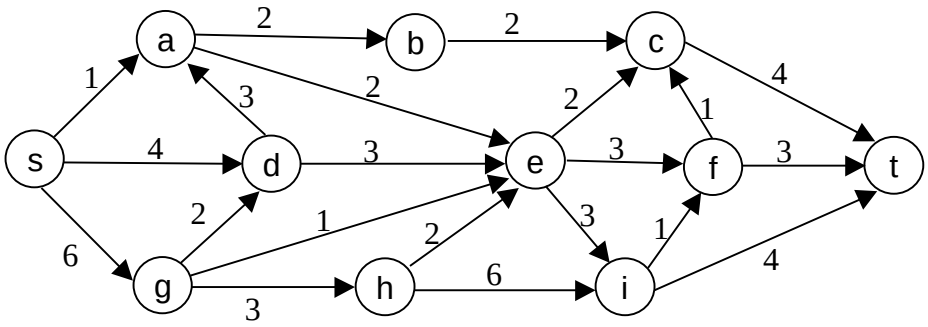
4.30 Una raíz de un GDA es un vértice *r* tal que todos los nodos del grafo pueden alcanzarse desde *r* (es decir existen caminos entre *r* y el resto de nodos). Escribir un procedimiento para determinar si un GDA posee una raíz.

4.31 * Para los siguientes tipos de grafos, decir si son biconexos o no, y cuál es su conectividad:

- Grafo con estructura de anillo.
- Grafo con estructura de árbol.
- Grafo completo, con n nodos.

4.32 Construir un algoritmo para evaluar expresiones aritméticas representadas mediante GDA. Suponer que tenemos almacenada una etiqueta para cada nodo ETIQ[1..n] con los valores (+, -, *, A, B, ..., Z). Los valores para las variables son obtenidos llamando a un función *ValorVariable (carácter): Real*. **Sugerencia**: almacenar el valor de cada nodo en un array VALOR[1..n].

4.33 Encontrar una ordenación topológica para el siguiente grafo. Mostrar los pasos de ejecución del algoritmo. ¿Es única esta ordenación o existen otras ordenaciones válidas?

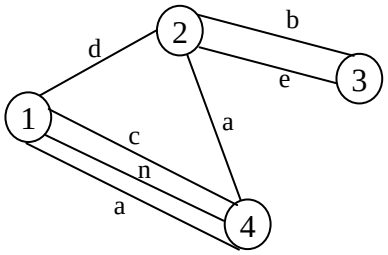


4.34 Un problema de planificación de tareas es representado utilizando GDA. Los nodos contienen tareas a realizar y las aristas indican la precedencia de tareas (si tenemos <v, w>, entonces w no puede empezar hasta que haya acabado v). Estas aristas son etiquetadas con un costo, que indica el tiempo necesario para acabar la tarea que es cabeza de la arista (en este caso v). Existen dos nodos destacados, *Inicio* y *Final*, para indicar el principio y el final del plan. Puesto que todas las tareas se deben realizar necesariamente para acabar el plan, el problema fundamental consiste en calcular la longitud del camino más largo entre *Inicio* y *Final*, que será el tiempo mínimo en ejecutar el plan.

a) Comprobar que una simple modificación del algoritmo de Dijkstra (buscando máximos en lugar de mínimos) no resuelve el problema de calcular el camino más largo.
b) Proponer un algoritmo para calcular la longitud del camino más largo entre los nodos *Inicio* y *Final*, teniendo en cuenta que trabajamos con GDA. ¿Cuál es el orden de complejidad del algoritmo?

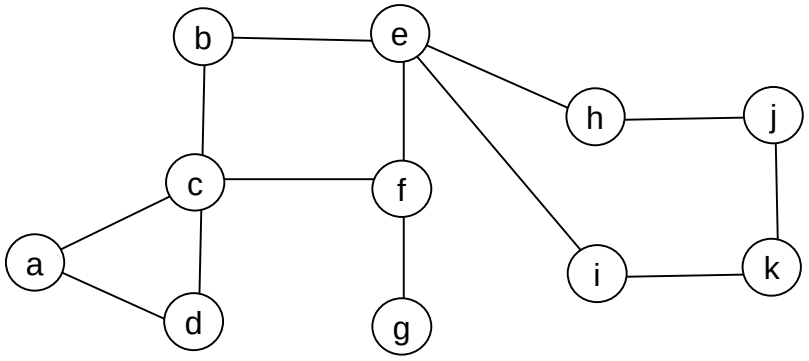
4.35 La asignación de números en orden topológico (num_top) puede ser realizada con un recorrido en profundidad. Si G es un GDA, entonces el orden de terminación de las llamadas recursivas de bpp (orden posterior) es un orden topológico inverso. La numeración se puede realizar asignando N al primer nodo en salir de bpp, N-1 para el siguiente y así sucesivamente. Es decir, llevaríamos un contador que empieza en N y va decreciendo hasta 1. Demostrar que el orden obtenido de esta forma es un orden topológico.

4.36 * En una aplicación que usa grafos etiquetados, queremos permitir que pueda existir más de una arista entre dos nodos (v, w). Tendríamos lo que se denomina multigrafo. ¿Cuál de las estructuras de representación de grafos se adapta más fácilmente a esta modificación? Justificar la respuesta brevemente.



Ejemplo de multigrafo etiquetado

4.37 Aplicar el algoritmo para calcular los puntos de articulación sobre el grafo de la siguiente figura. Suponer que la búsqueda primero en profundidad empieza desde el nodo **a**.



4.38 Supongamos un grafo no dirigido y conexo con N nodos. ¿Cuál es el máximo valor posible de conectividad para este grafo? ¿A qué tipo de grafo corresponde? ¿Cuál es el número mínimo de aristas necesario para que el grafo sea biconexo, es decir, que no tenga puntos de articulación?

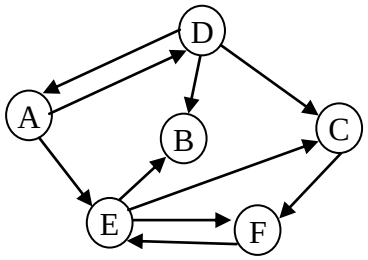
4.39 Implementar el algoritmo para la búsqueda de puntos de articulación en un grafo no dirigido, modificando las partes adecuadas del procedimiento bpp de la búsqueda primero en profundidad. ¿Cuál es el orden de complejidad del algoritmo?

4.40 En el algoritmo para calcular los puntos de articulación de un grafo no dirigido, puesto que la condición del punto 4 no se puede cumplir para los nodos hoja del árbol de expansión en profundidad, se puede concluir que las hojas nunca serán puntos de articulación. Demostrar que esta conclusión es cierta, sin usar las propiedades del algoritmo comentado.

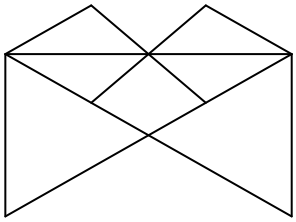
4.41 ¿Cuál es el costo, en el peor caso, del algoritmo para calcular el flujo máximo en un grafo dirigido? Suponer que el grafo tiene *n* nodos, *a* aristas y estamos usando la primera versión de las vistas en clase (que no permite deshacer caminos).

4.42 Aplicar el algoritmo de flujo máximo sobre el grafo del ejercicio 127, obteniendo el grafo G_F de flujos resultantes. Aplicar el algoritmo que permite deshacer caminos.

4.43 * Encontrar las componentes fuertemente conexas del siguiente grafo dirigido. Con el resultado obtenido, mostrar el grafo reducido correspondiente.



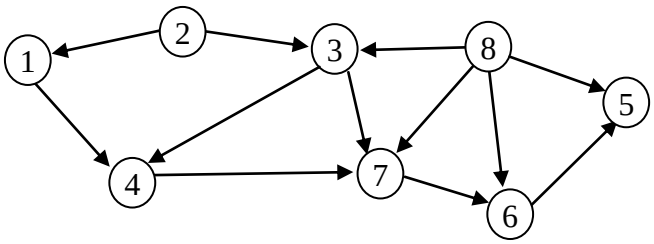
4.44 * Dado el siguiente dibujo de líneas, ¿es posible dibujarlo con un bolígrafo, pasando una y sólo una vez por cada línea y sin levantar el bolígrafo del papel (pudiendo empezar y acabar en sitios distintos)? En caso afirmativo, dibujarlo en el orden adecuado. En caso negativo, demostrar por qué no es posible.



4.45 * Mostrar un ejemplo de grafo no dirigido con conectividad 3 y que tenga cinco nodos o más. ¿Cuántos nodos hay que eliminar para desconectar el grafo?

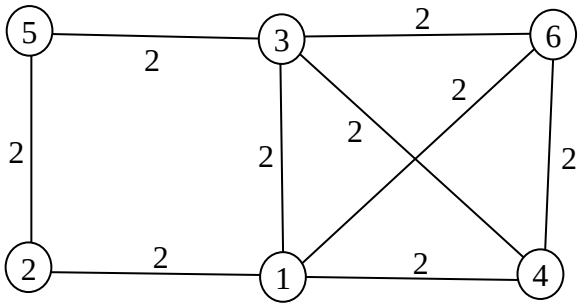
4.46 * Para resolver el problema de los caminos mínimos entre cualquier par de nodos se puede usar el algoritmo de Floyd o el algoritmo de Dijkstra repetido *n* veces (una vez por cada nodo origen). ¿Cuándo será más rápida una u otra opción, en cuanto a órdenes de complejidad (usando las implementaciones vistas en clase)?

4.47 * Dado el siguiente GDA, calcular las componentes fuertemente conexas y el grafo reducido correspondiente.



A la vista del resultado, ¿podemos establecer alguna relación general entre un GDA y su grafo reducido? ¿Cuál? ¿Por qué?

4.48 * Elegir un algoritmo para calcular el árbol de expansión de coste mínimo. Mostrar los pasos de ejecución (de forma breve) y el resultado obtenido para el siguiente grafo.



En general, en el caso donde todas las aristas tienen el mismo coste *K*, ¿qué conclusiones podemos extraer sobre el árbol de expansión de coste mínimo?

4.49 * Para representar las aristas de un grafo dirigido y etiquetado usamos una estructura de dispersión abierta con M cubetas. Dada una arista <v, w> con etiqueta E(v, w), la función de dispersión es h(v, w). Comparar, mediante una tabla, la anterior estructura con las matrices y listas de adyacencia, para la memoria ocupada y el tiempo de las operaciones de búsqueda de una arista y de contar el número de aristas. Los grafos tienen *n* vértices y **a** aristas. Suponer las demás constantes que se consideren necesarias.

4.50 * Demostrar que la siguiente proposición es cierta o dar un contraejemplo en caso de ser falsa.

PROPOSICIÓN: Dado un grafo no dirigido G cualquiera:
La conectividad de G es *k* $\Leftrightarrow$ todos los vértices de G tienen grado *k* o mayor