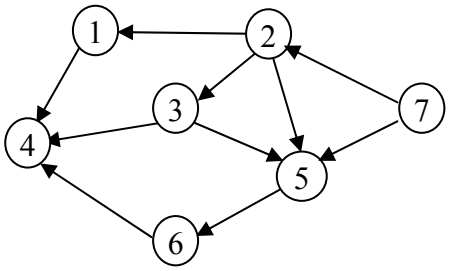
Ejercicios

4.1. Dado un árbol de expansión resultante de un recorrido sobre un grafo no dirigido, ¿qué tipo de arcos (a parte 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 pre-orden de un árbol, un programador decide implementar el equivalente a un recorrido en post-orden (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 **write(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?

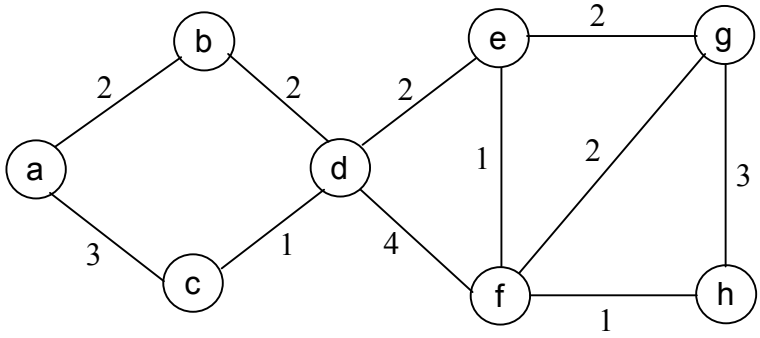
4.3. ¿Cuál es el número máximo de arcos que puede tener un grafo no dirigido sin ciclos? ¿Y cuál será para un grafo dirigido acíclico (GDA)?

4.4. (EX) Mostrar una ordenación topológica para el siguiente grafo dirigido acíclico. ¿La ordenación obtenida es única? En caso negativo mostrar otra ordenación válida.



4.5. 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 coste mínimo utilizando el algoritmo de Prim.
d) El árbol de expansión de coste mínimo utilizando el algoritmo de Kruskal. ¿Son iguales las soluciones obtenidas en ambos algoritmos? En caso contrario, ¿son válidas las distintas soluciones? ¿Por qué?



4.6. 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 4.3.

4.7. 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.

4.8. (EX D04) Escribir una operación 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.

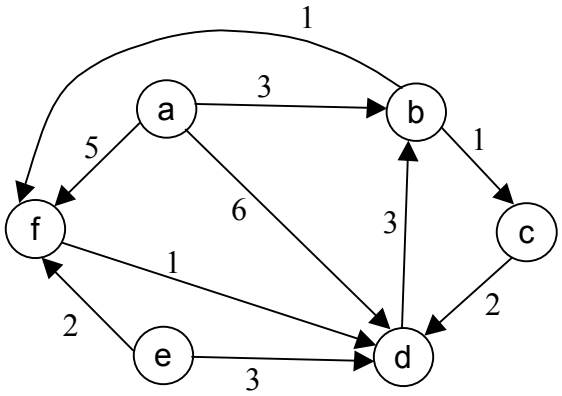
4.9. Diseñar un algoritmo para contar el número de ciclos simples existentes en un grafo no dirigido. ¿Cuál es el orden de complejidad de este algoritmo? ¿Cuántos ciclos pueden haber como máximo en un grafo cualquiera?

4.10. ¿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**: considera la estructura de datos utilizada.

4.11. (EX) Modificar el procedimiento de búsqueda primero en profundidad para que cuente el número de árboles del bosque de expansión 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.

4.12. Demostrar que el algoritmo de Dijkstra no funciona cuando las aristas pueden tener coste 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.

4.13. 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. A partir del resultado, encontrar cuál es el camino más corto desde **a** hasta **d**.



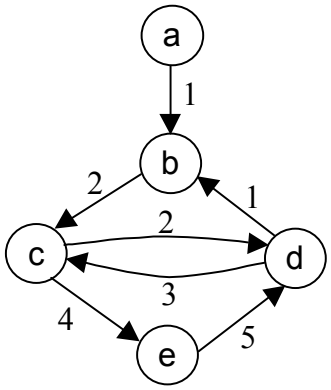
4.14. (TG 5.2) En general, dados dos nodos, **v** y **w**, en un grafo con pesos puede existir más de un camino mínimo entre ambos (aunque necesariamente todos

tendrán el mismo coste). Modificar los algoritmos de Dijkstra y Floyd para contar el número de caminos mínimos distintos existentes entre un nodo **v** y el resto de nodos, o entre todos los nodos, respectivamente.

4.15. ¿Cuál es el orden de complejidad del algoritmo para encontrar los componentes fuertemente conexos en un grafo dirigido? Suponer que el grafo tiene **n** nodos y **a** aristas, siendo **a**>**n**.

4.16. Aplicar el algoritmo para obtener los componentes fuertemente conexos para el grafo del ejercicio 4.13. A partir del resultado obtenido, mostrar el grafo reducido correspondiente.

4.17. Mostrar el resultado de la aplicación del algoritmo de Floyd sobre el siguiente grafo dirigido.



4.18. Se define la **excentricidad** de un nodo **v** como la distancia máxima de los caminos mínimos desde todos los nodos hasta **v**. Por otro lado, se define el **centro de un grafo** como el nodo con menor excentricidad. Dar un algoritmo para calcular el centro de un grafo. ¿Cuál es el centro del grafo del ejercicio 4.17?

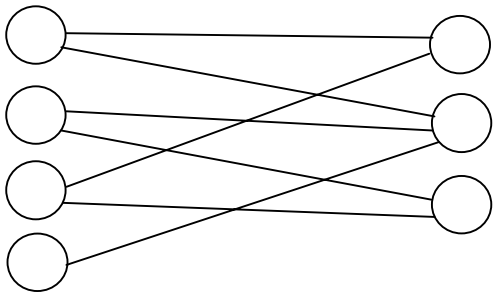
4.19. (EX) Modificar el algoritmo de Dijkstra para que, además de calcular los caminos mínimos entre un nodo y los demás, también calcule en otro array **L** el número de aristas por las que pasa cada uno de los caminos mínimos. Decir únicamente las partes que se deben modificar del algoritmo.

4.20. (EX) 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.

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

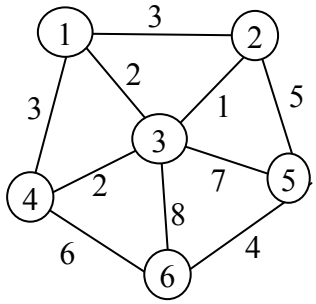
4.22. Un grafo no dirigido **G** = (V, A), se dice que es bipartido si V se puede partir en dos subconjuntos V1 y V2, de manera que para toda arista (v, w) de A, el nodo v pertenece a uno de los conjuntos (V1 ó V2) y w pertenece al otro. Proporcionar un

algoritmo con tiempo O(n+a) para comprobar si un grafo es bipartido o no. **Sugerencia:** Usar una **bpp** para asignar una partición (1 ó 2) a cada nodo visitado.



Ejemplo de grafo bipartido

4.23. (EX) 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?

4.24. (EX) Para desarrollar la especificación formal del TAD grafo dirigido y etiquetado disponemos de los siguientes conjuntos:

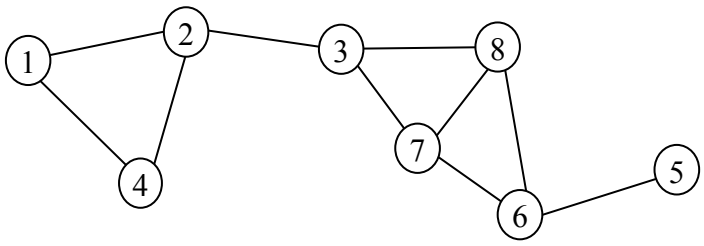
G	Conjunto de grafos dirigidos y etiquetados
M	Conjunto de nodos de los grafos
E	Conjunto de valores de las etiquetas en las aristas
N	Conjunto de naturales
B	Conjunto de booleanos
U	Conjunto de mensajes de error

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

4.25. 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**: haz uso de la ordenación topológica, para seleccionar los nodos entre los candidatos.

4.26. 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?

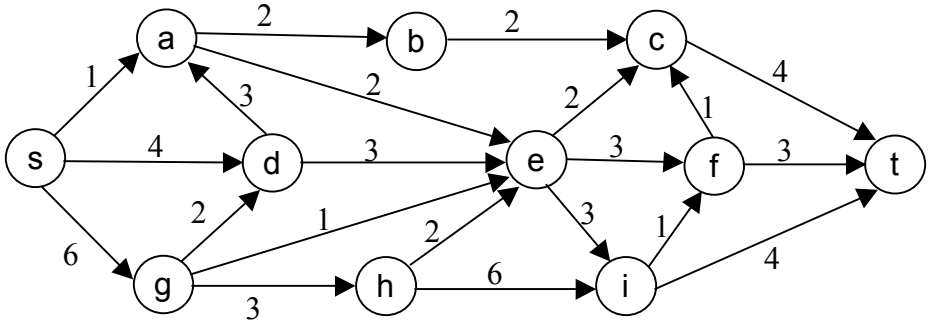
4.27. (EX) Mostrar los puntos de articulación del siguiente grafo. ¿Cuáles son los componentes biconexos?



4.28. 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.

4.29. (EX) 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.

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

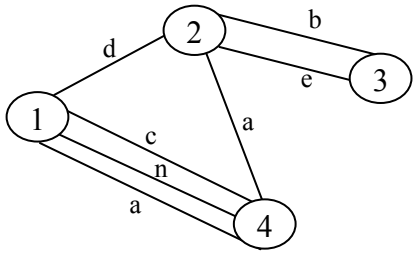


4.31. Un problema de planificación de tareas es representado utilizando GDA. Los nodos contienen tareas a realizar y las aristas indican la precedencia de tareas (si tenemos **<v, w>** entonces **w** no puede empezar hasta que haya acabado **v**). Estas aristas son etiquetadas con un costo, que indica el tiempo necesario para acabar la tarea que es cabeza de la arista (en este caso **v**). Existen dos nodos destacados, *Inicio* y *Fin*, 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.

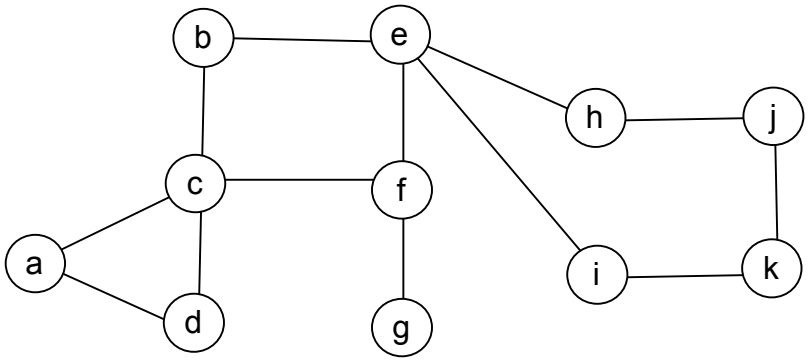
b) Propón un algoritmo para calcular la longitud del camino más largo entre los nodos *Inicio* y *Fin*, teniendo en cuenta que trabajamos con GDA. ¿Cuál es el orden de complejidad del algoritmo?

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



Ejemplo de multigrafo etiquetado

4.33. 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**.



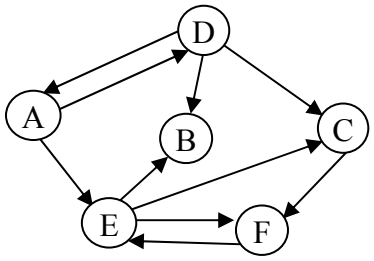
4.34. Supongamos un grafo no dirigido y conexo con **n** nodos. ¿Cuál es el máximo valor posible de conectividad para este grafo? ¿A qué tipo de grafo corresponde? ¿Cuál es el número mínimo de aristas necesario para que el grafo sea biconexo, es decir, que no tenga puntos de articulación?

4.35. 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?

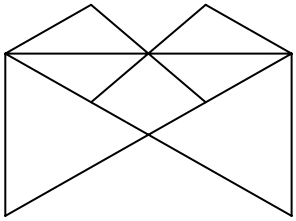
4.36. En el algoritmo para calcular los puntos de articulación de un grafo no dirigido, puesto que la condición del punto 4 no se puede cumplir para los nodos hoja del árbol de expansión en profundidad, se puede concluir que las hojas nunca serán puntos de articulación. Demostrar que esta conclusión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

4.37. (TG sección 5.6.3) Diseñar un algoritmo óptimo para calcular el flujo máximo que puede circular entre dos nodos, en un grafo dirigido con pesos. Aplicarlo sobre el grafo del ejercicio 4.30, obteniendo el grafo G_F de flujos resultantes.

4.38. (EX) Encuentra los componentes fuertemente conexos del siguiente grafo dirigido. Con el resultado obtenido mostrar el grafo reducido correspondiente.



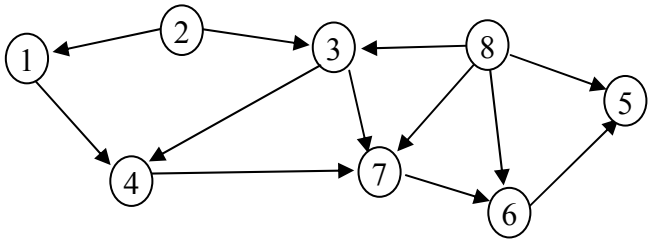
4.39. (EX) Dado el siguiente dibujo de líneas, ¿es posible dibujarlo con un bolígrafo, pasando una y sólo una vez por cada línea y sin levantar el bolígrafo del papel (pudiendo empezar y acabar en sitios distintos)? En caso afirmativo, dibujarlo en el orden adecuado. En caso negativo, demostrar por qué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



4.40. (EX) Muestra 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?

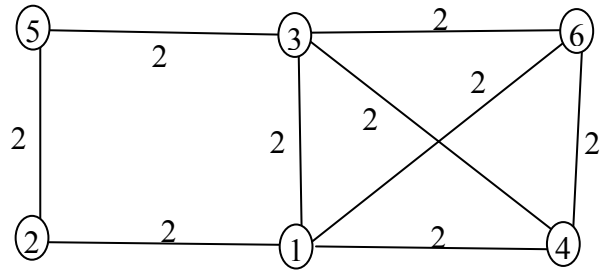
4.41. (EX) Para resolver el problema de los caminos mínimos entre cualquier par de nodos se puede usar el algoritmo de Floyd o el algoritmo de Dijkstra repetido **n** veces (una vez por cada nodo origen). ¿Cuándo será más rápida una u otra opción, en cuanto a órdenes de complejidad (usando las implementaciones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

4.42. (EX M01) Dado el siguiente GDA, calcula los componentes fuertemente conexos 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?

4.43. (EX D01) Elige un algoritmo para calcular el árbol de expansión de coste mínimo. Muestra 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?

4.44. (EX D01) Para representar las aristas de un grafo dirigido y etiquetado usamos una estructura de dispersión abierta con B cubetas. Dada una arista <v, w> con etiqueta E(v, w), la función de dispersión es h(v, w). Compara, 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. Supón las demás constantes que consideres necesarias.

4.45. (EX M02) 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

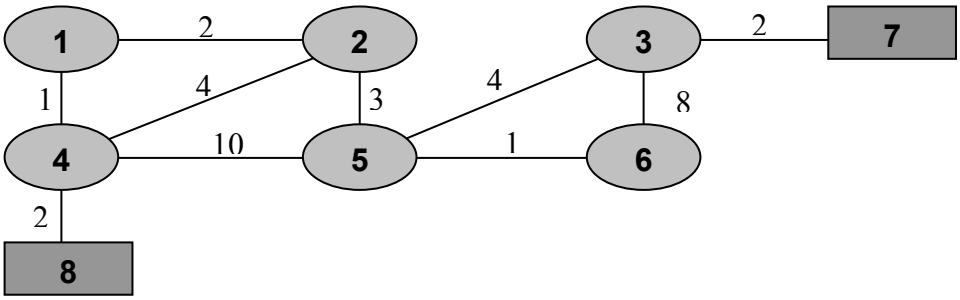
4.46. (TG 5.5, EX D02) La experimentación con ratones es básica para crear nuevos fármacos, que resultan en importantes mejoras en la calidad de vida de los roedores. En un experimento con fármacos cerebrales, se colocan ratones dentro de un laberinto formado por **n** celdas, **m** salidas del laberinto y varios pasadizos entre las celdas (o las salidas). Las celdas son numeradas de **1** a **n** y las salidas de **n+1** a **n+m**. En cada pasadizo los ratones tardan un tiempo dado, que llamaremos **P[i, j]**: tiempo de ir de la celda **i** a la celda (o salida) **j**. Se supone que el tiempo que tardan en las celdas es despreciable.

El experimento consiste en colocar inicialmente un ratón en cada celda, y contar el número de ratones que son capaces de salir del laberinto en un tiempo dado $\mathbf{t_{max}}$. Suponiendo que los ratones eligen siempre el camino más corto hacia alguna de las salidas, las preguntas a resolver son: i) ¿Cuántos ratones habrán salido en el instante dado $\mathbf{t_{max}}$? ii) ¿Cuánto tiempo se necesita para conseguir que todos los ratones hayan salido?

Se pide:

a) Dar un algoritmo para resolver los dos problemas anteriores. Se supone que el instante inicial es **t**= 1. Hacer una estimación aproximada del orden de complejidad del algoritmo. No se pueden dar por supuestos los algoritmos vistos en clase, aunque sí las implementaciones de los tipos de datos lista y grafo.

b) Aplicar el algoritmo al siguiente ejemplo, donde **n**= 6, **m**= 2 y $\mathbf{t_{max}}$= 5. Las celdas han sido representadas con elipses, las salidas con rectángulos y los pasadizos con aristas etiquetadas. Además, en este caso se supone que **P[i, j]** = **P[j, i]**.

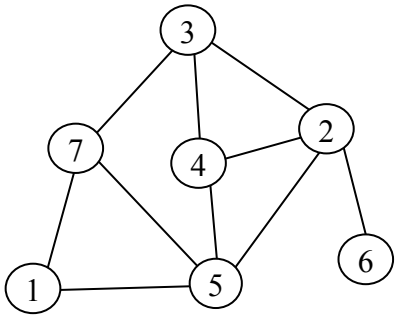


4.47. (TG 5.4, EX M03) En cierto país petrolífero se supone la existencia de armas de destrucción masiva. La ONU ha decidido planificar una serie de inspecciones exhaustivas, para analizar todos los posibles almacenes de armas. Tenemos un mapa **T** de localizaciones, con **n** sitios sospechosos y los caminos existentes entre ellos. En el mapa, **T**[**i**,**j**]=**T**[**j**,**i**] valdrá 1 si existe un camino entre **i** y **j**, y 0 en caso contrario. Existen varios equipos de inspectores, que parten inicialmente del mismo sitio. Cada equipo visita un sitio durante un día y se desplaza por la tarde (es decir, en todos los caminos tardamos un tiempo unitario).

Hay que tener en cuenta que las armas se pueden mover de sitio, para evitar ser descubiertas. Las armas se mueven por los mismos caminos que los inspectores, aunque lo hacen por la noche (es decir, no se pueden cruzar por el camino). Para garantizar que el plan de inspecciones sea efectivo debemos garantizar lo siguiente: 1) todos los sitios son inspeccionados, 2) hay que evitar que se puedan mover armas a un sitio que ya ha sido inspeccionado; para ello, debemos garantizar que cualquier posible camino entre un sitio inspeccionado y uno no inspeccionado pase por un sitio que esté siendo inspeccionado.

Diseñar un algoritmo para resolver alguno de los siguientes problemas (elegir uno). Mostrar la ejecución sobre el mapa de la figura de abajo.

a) Suponiendo que todos los equipos de inspectores parte del sitio **I**, encontrar cuánto tiempo tarda el plan de inspecciones (como mínimo), qué sitios se deben visitar cada día, cuántos equipos de inspectores necesitamos en total y por dónde se debe mover cada equipo. Ejecutar con **I** = 1.
b) Suponiendo que queremos minimizar el tiempo total del plan de inspecciones, encontrar desde qué sitio deben partir las inspecciones, qué sitios se deben visitar cada día y cuántos equipos de inspectores necesitamos.



4.48. (EX) Para una boda tenemos una lista de **n** personas que pueden ser invitadas o no. Pero no todas pueden serlo, ya que existen enemistades conocidas entre ellas, que debemos evitar. Las enemistades conocidas vienen dadas como pares ($\mathbf{a_1}$, $\mathbf{b_1}$), ($\mathbf{a_2}$, $\mathbf{b_2}$),…, ($\mathbf{a_k}$, $\mathbf{b_k}$), indicando cada par **i-**ésimo que $\mathbf{a_i}$ y $\mathbf{b_i}$ son enemigos conocidos. Aplicando transitividad, los enemigos de una persona **i** son todos los enemigos conocidos de **i** y los enemigos de sus amigos. Decimos que los amigos de una persona son todos los enemigos de sus enemigos.

Una persona asistirá a la boda si es invitada y son invitados todos sus amigos y no es invitado ninguno de sus enemigos. El objetivo del problema es decidir la lista de invitados de manera que se maximice el número de asistentes a la boda.

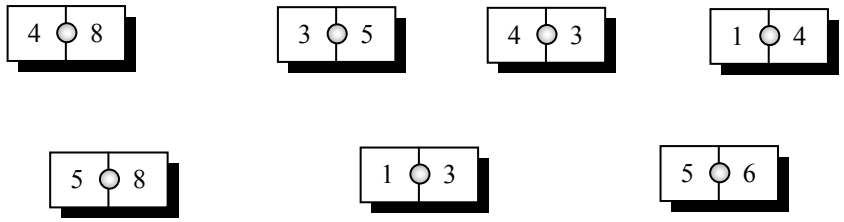
Diseñar un algoritmo para resolver el problema, justificando razonadamente su funcionamiento. Acotar el orden de complejidad del algoritmo diseñado. Mostrar la ejecución sobre el siguiente ejemplo.

Posibles invitados: **n** = 8

Enemistades conocidas: (1,2), (5,7), (6,8), (1,4), (2,7)

Nota: Aplicando las definiciones es posible, en algún caso, que dos personas sean a la vez amigos y enemigos. En ese caso ninguno de ellos puede ser invitado.

4.49. (EX) Consideremos un juego de dominó generalizado, donde cada ficha tiene un par de números **(a, b)**, que pueden ir de 0 a 99. Podemos formar cadenas de fichas, poniendo una al lado de otra. Como ya debes saber, en el dominó podemos poner una ficha al lado de otra si tienen el mismo número en el lado en el que se juntan. El objetivo del problema es, dado un conjunto de **n** fichas de dominó con valores **($\mathbf{a_i}$, $\mathbf{b_i}$)** para **i**= 1, ..., **n**, encontrar la cadena de fichas más larga que se puede formar. Diseñar un algoritmo para resolver el problema. Indicar si el algoritmo garantiza la solución óptima o no.



4.50. (EX) Cagitán City, años 20. Las mafias controlan varios **sitios** y **calles** de la ciudad. El alcalde, que debe desplazarse diariamente de su residencia al ayuntamiento, está seriamente amenazado. Debes encontrar la ruta más segura.

La ciudad está descrita como un conjunto de **n** sitios y varias calles que unen esos sitios. La matriz **C**: array [1..**n**, 1..**n**] de booleano, indica en cada **C[a, b]** si entre los sitios **a** y **b** existe una calle o no (es una matriz simétrica). En los arrays **M**: array [1..**n**] de booleano y **N**: array [1..**n**, 1..**n**] de booleano, se indican los sitios y calles controlados por la mafia, respectivamente (**true**: sitio o calle controlado; **false**: sitio o calle libre). Suponemos que la residencia del alcalde es el sitio **1**, y el ayuntamiento el sitio **n**. Debes encontrar un camino entre **1** y **n**, que pase por el menor número de calles y sitios controlados por la mafia.

Escribir un algoritmo para resolver el problema, explicando su funcionamiento de forma razonada.

4.51. (EX) Seguimos en el Cagitán. Pero ahora el mafioso eres tú. Decides cambiar de estrategia. En lugar de controlar varios sitios y calles (ahora están todos libres), tramas una emboscada contra el alcalde, colocándote en un sitio estratégico. Debes decidir en qué sitio colocarte.

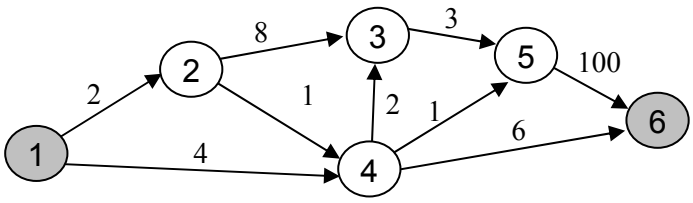
El sitio elegido debe cumplir dos propiedades. Primera: no puede ser en el ayuntamiento ni en la residencia del alcalde, altamente vigilados por la policía. Segunda: debes garantizar que el alcalde pasará por ese sitio siempre que se desplace de su residencia al ayuntamiento, sea cual sea el camino que elija.

Escribir un algoritmo, en pseudocódigo, para resolver el problema anterior eficientemente. Si no existe ningún sitio que cumpla las propiedades requeridas, el algoritmo debe indicarlo. Si existen varios, debe encontrar todos ellos.

4.52. (EX J04) Una empresa de reparto planifica las rutas de su flota de camiones entre distintas ciudades. Tenemos un grafo que representa el mapa de carreteras, siendo **C**[*v*,*w*] el tiempo que se tarda entre los puntos *v* y *w*. En general, nos interesará pasar por los caminos mínimos. Pero para prevenir incidentes, como atascos o cortes de carreteras, decidimos quedarnos con los dos caminos menos costosos (en lugar de sólo con el mínimo).

Diseñar un algoritmo que encuentre los dos caminos de menor coste entre dos nodos dados del grafo. Aplicarlo al grafo de abajo para encontrar los dos caminos mínimos entre 1 y 6.

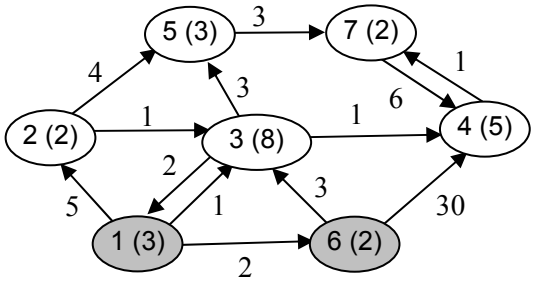
Nota: obviamente, los dos caminos deben ser distintos, aunque pueden compartir aristas. Por otro lado, ambos pueden tener el mismo coste si la solución óptima no es única.



4.53. (EX S04) En el algoritmo para encontrar el flujo máximo en un grafo dirigido con pesos, se buscan sucesivos caminos desde el nodo origen **s** al nodo final **t**, conocidos como *caminos crecientes*. Suponer que el grafo del problema, **G**, y el grafo de flujos resultantes, **F**, se almacenan en sendas matrices de adyacencia, **A** y **B**, de [1..n, 1..n] con los pesos de las aristas correspondientes. Escribir un algoritmo para encontrar un camino creciente en el grafo, que calcule el flujo que puede pasar por ese camino y que actualice las matrices **A** y **B** de forma adecuada.

4.54. (EX M05) Los marcianos han decidido invadir el planeta, otra vez. Su plan es el siguiente. Inicialmente invadirán un conjunto de ciudades seleccionado. A partir de ahí, moverán sus naves por todos los caminos posibles, invadiendo las ciudades que encuentren. Es decir, si en cierto instante han invadido una ciudad **X**, a continuación enviarán naves para invadir (de forma simultánea) todas las ciudades adyacentes a **X**, que no estén ya invadidas o siendo invadidas. Diseñar un algoritmo para resolver las dos siguientes cuestiones: cuánto tiempo tardarán en invadir todo el planeta, y por qué caminos se moverán (suponer que a cada ciudad sólo se llega desde un sitio). Aplicar el algoritmo al ejemplo de abajo.

Datos del problema. Existen **n** ciudades, numeradas desde 1 hasta **n**. La matriz **H**, de tamaño **n**x**n**, indica en cada posición **H[X,Y]** el tiempo que se tarda desde la ciudad **X** a la **Y**. La matriz no es simétrica y tendrá un valor infinito si no existe conexión de **X** a **Y**. Además, una vez han llegado a una ciudad, los marcianos tardan cierto tiempo en invadirla, según la resistencia que encuentren. El array **F**, de tamaño **n**, indica en cada **F[X]** el tiempo de invadir la ciudad **X**. Las ciudades invadidas inicialmente están dadas en el array **I**, de booleanos, siendo **I[X]** = *true* si y sólo si la ciudad **X** es un punto de inicio de la invasión.



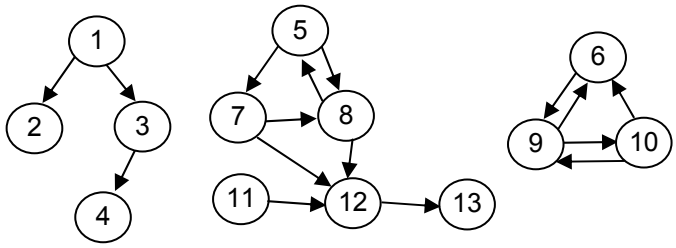
Los tiempos de invadir cada ciudad, **F**, se indican entre paréntesis. Las ciudades iniciales de invasión son las que aparecen señaladas en gris.

4.55. (EX J05) El Metro de Mula ya está en funcionamiento. Su uso supondrá una reducción en los tiempos de desplazamiento entre los distintos puntos del pueblo, más en las distancias largas que en las cortas. La compañía "Metro de Mula" quiere medir la ganancia media de tiempos. La ganancia para dos puntos **a** y **b** se define como la diferencia entre el tiempo mínimo entre **a** y **b** sin usar el metro, y el tiempo mínimo entre **a** y **b** pudiendo usar el metro. Notar que en este segundo mínimo puede que lo conveniente sea no usar el metro, o usarlo una o varias veces. Hay que tener en cuenta que cada vez que entramos o salimos del metro tardamos un tiempo adicional **K**. Diseñar un algoritmo en pseudocódigo para resolver el problema, obteniendo la ganancia media. Se pueden usar llamadas a algoritmos estudiados en clase; no es necesario programarlos pero sí aclarar sus datos de entrada y salida.

Datos del problema. El mapa de la ciudad tiene **n** puntos. Los tiempos de los caminos directos vía superficie (es decir, sin usar el metro) están dados en la matriz de costes **T**. Los tiempos entre los mismos puntos usando el metro están dados en la matriz **M**. Ambas matrices son simétricas y ambas contendrán $+\infty$ si no existe camino o línea de metro entre dos puntos. Recordar también el tiempo adicional **K** cada vez que se entra o sale del metro.

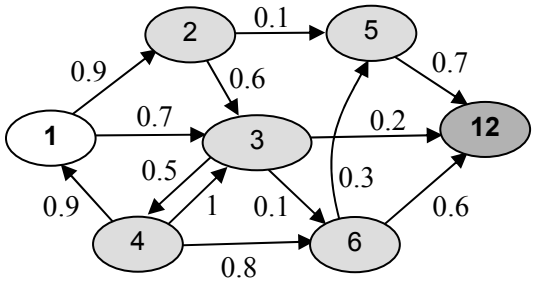
4.56. (EX S05) Tenemos un conjunto de páginas web que queremos poner en una página de enlaces. Pero nos damos cuenta de que algunas de estas páginas ya enlazan a otras del conjunto, de forma que podemos eliminar las ya enlazadas. Por ejemplo, supongamos que la página 1 enlaza la 2 y la 3; y que la 3 enlaza a la 4. Si seleccionamos la 1, no necesitamos poner 2, 3 ni 4. El objetivo es seleccionar el número mínimo de páginas tal que todas las páginas del conjunto están enlazadas directa o indirectamente. Escribir un algoritmo para resolver el problema. Se pueden suponer los algoritmos vistos en clase, siempre que quede claro su uso (cuáles son los parámetros de entrada, el resultado y el efecto del algoritmo).

Datos del problema. El conjunto inicial tiene **n** páginas, enumeradas como {1, 2, ..., n}. La matriz de booleanos **M**, indica en cada posición **M[a, b]** si la página **a** enlaza a la **b**. La matriz **M** no es necesariamente simétrica.
Ejemplo, con **n** = 13. Las páginas se representan con círculos y los enlaces mediante flechas. Una posible solución óptima (no la única) sería seleccionar: 1, 6, 7, 11.



4.57. (EX F06) ¡El mundial se acerca y la selección española te necesita! Tenemos un esquema del equipo con 12 posiciones, una por cada jugador (el portero es el número 1) más una para la portería contraria (que es el número 12). Para cada par de jugadores (**a**, **b**), tenemos la probabilidad, **P[a, b]** ∈ (0, 1), de que un pase desde **a** hasta **b** salga bien, es decir, que no lo intercepte el equipo contrario. La matriz no es simétrica, y **P[a, 12]** indica la probabilidad de que **a** marque un gol al chutar. A partir de esas probabilidades básicas, se puede calcular la probabilidad de una secuencia de pases, mediante el producto. La probabilidad de que la secuencia de pases **a** → **b** → **c** salga bien será: **P[a, b]*P[b, c]**, y así sucesivamente. Una estrategia de juego es una secuencia de pases que empieza en nuestro portero y acaba en gol en la portería contraria.

Escribir un algoritmo eficiente que encuentre la estrategia de juego óptima, es decir, la secuencia de pases entre 1 y 12 que maximice la probabilidad de salir bien. Aplicar el algoritmo al ejemplo de abajo, indicando la estrategia óptima y la probabilidad asociada.



Esquema del equipo con probabilidades **P[a, b]** de que los pases salgan bien. Si no aparece una flecha es que la probabilidad es 0

4.58. (EX) Un desfile se realiza por las calles de un pueblo. Los puntos de inicio y finalización del desfile son fijos. Queremos que el recorrido pase por el mayor número de calles posible, pero sin repetir ninguna calle ni intersección. Supongamos que hay *n* intersecciones entre calles y que disponemos de una matriz *C* de booleanos, de tamaño *n* x *n*, en la que *C*[*i,j*] es cierto si existe una calle de *i* a *j*, y falso en caso contrario. Seleccionar alguno de los algoritmos de recorrido en grafos estudiados en clase y modificarlo para resolver el problema, devolviendo el camino para el desfile.