# Métodos Algorítmicos en Resolución de Problemas I

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas

Hoja de ejercicios 4 Curso 2022-2023

---

## Ejercicios de grafos

**Ejercicio 1** Se llama *grado de salida* de un vértice $v$ de un grafo dirigido al número de aristas salientes de $v$, y *grado de entrada*, al de aristas entrantes. ¿Cuál es el coste de un algoritmo que calcula el grado de salida de cada vértice, si el grafo viene representado mediante listas de adyacencia? ¿Y el de uno que calcula el grado de entrada?

**Ejercicio 2** Se llama *traspuesto* de un grafo $(V, A)$ al grafo $(V, A^T)$ donde $A^T = \{(v, u) \mid (u, v) \in A\}$. Diseñar algoritmos para calcular el traspuesto de un grafo, y dar su coste, en los casos en que el grafo venga representado por listas de adyacencia, y por matriz de adyacencia.

**Ejercicio 3** Se llama *cuadrado* de un grafo $(V, A)$ al grafo $(V, A^2)$ donde $A^2 = \{(u, w) \mid (u, v) \in A \wedge (v, w) \in A\}$. Diseñar algoritmos para calcular el cuadrado de un grafo, y dar su coste, en los casos en que el grafo venga representado por listas de adyacencia, y por matriz de adyacencia.

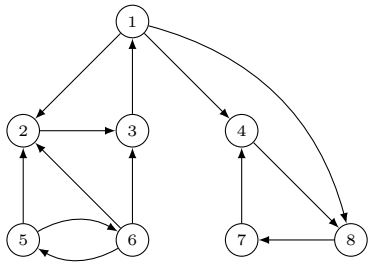
**Ejercicio 4** En un curso con $n$ estudiantes se conoce la relación $A = \{(x, y) \mid x \text{ ama a } y\}$, siendo $a = \mid A \mid$. Diseñar un algoritmo de coste $O(n + a)$ que etiquete a todos los estudiantes, bien como *amantes*, si solo aparecen en parte la izquierda de las tuplas de $A$, bien como *amados*, si solo aparecen en la derecha, o como *solitarios* si no aparecen, o bien que indique que tal asignación no es posible.

**Ejercicio 5** Supongamos, que en la representación mediante listas de adyacencia y para cada vértice $v$, sustituimos su lista de adyacencia por una tabla *hash* que represente el conjunto de sus vértices adyacentes. ¿Cuál sería el coste de la operación $existeArista(G, v, w)$? ¿Cuál sería el coste de $adyacentes(G, v)$? ¿Cuál sería el coste en espacio de esta representación?

**Ejercicio 6** Un vértice $p$ de un grafo dirigido se llama *sumidero* si para todo otro vértice $v$ existe la arista $(v, p)$ pero no la arista $(p, v)$. Escribir un algoritmo que detecte la presencia de un sumidero en un grafo $G$ en tiempo $O(n)$, donde $n$ es el número de vértices. El algoritmo debe aceptar un grafo representado por su matriz de adyacencia.

**Ejercicio 7** Decir cuál sería el coste en tiempo del algoritmo BFS, si el grafo estuviera representado mediante una matriz de adyacencia.

**Ejercicio 8** Mostrar cómo progresa el recorrido en anchura en el siguiente grafo, suponiendo que los vértices adyacentes se visitan atendiendo a su orden numérico y que los vértices iniciales que sean necesarios también se eligen en función de su orden numérico.




**Ejercicio 9** Mostrar cómo progresa el recorrido en profundidad del grafo del ejercicio 8, suponiendo que los vértices adyacentes se visitan atendiendo a su orden numérico y que los vértices iniciales que sean necesarios también se eligen en función de su orden numérico. Indicar los tiempos de descubrimiento y finalización de cada vértice. A partir de esta numeración, ¿es posible calcular una ordenación topológica de sus vértices?

**Ejercicio 10** Escribir un algoritmo iterativo que realice un recorrido en profundidad en un grafo.

**Ejercicio 11** Utilizar un recorrido en profundidad para encontrar las componentes conexas de un grafo no dirigido. De forma precisa, modificar el procedimiento *DFS* para que asigne a cada vértice $v$ un valor $cc[v]$ que represente la componente conexa a la que pertenece.

**Ejercicio 12** Escribir un algoritmo que se base en el recorrido en profundidad para determinar si un grafo no dirigido es acíclico y conexo.

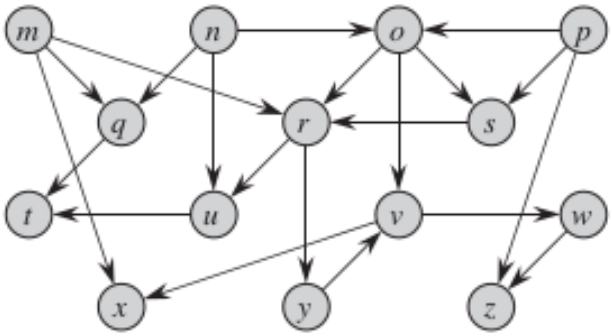
**Ejercicio 13** Dado un grafo no dirigido $G$ representado de forma abstracta por medio de una función *adyacentes* que a cada vértice $i$ le asocia la lista de sus vértices adyacentes,

1. Escribir un algoritmo que calcule el número de componentes conexas de $G$ y cuántas de ellas no contienen ciclos.
2. Indicar cómo encontrar de manera eficiente un subconjunto de arcos del grafo lo más pequeño posible que mantenga la relación de alcanzabilidad original, es decir si en el grafo dado el vértice $j$ es alcanzable desde el $i$, entonces en el subgrafo generado, $j$ también lo será.

**Ejercicio 14** Un grafo dirigido es una *arborescencia* si existe un vértice, llamado raíz, desde el que se puede alcanzar cualquier otro vértice a través de un camino único. Escribir un algoritmo (basándose en el recorrido en profundidad) para determinar si un grafo dirigido es una arborescencia y, en caso de serlo, determinar su raíz.

**Ejercicio 15** Se dice que un grafo no dirigido $G = (V, E)$ es $k$-coloreable si todos los vértices de $G$ pueden ser coloreados usando $k$ colores diferentes de manera que no haya dos vértices adyacentes que tengan el mismo color. Diseñar un algoritmo cuyo coste en tiempo esté en $O(|V| + |E|)$ que coloree un grafo con dos colores o determine que el grafo no es 2-coloreable.

**Ejercicio 16** Mostrar el resultado y los pasos intermedios del algoritmo de ordenación topológica para el grafo de la figura. Cada parte del algoritmo que necesite recorrer un conjunto de vértices lo hará siempre en orden alfabético.



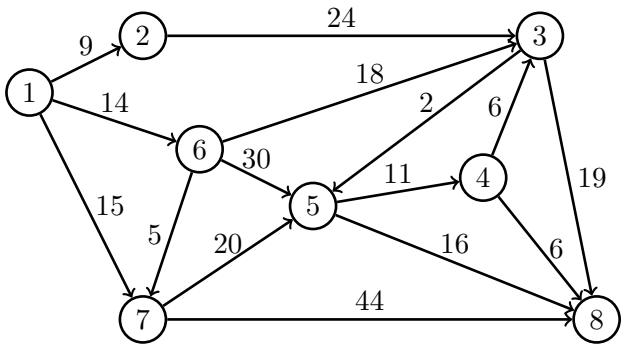

**Ejercicio 17** Dado un grafo dirigido y valorado $G = (V, A)$, la capacidad de flujo de un camino es el mínimo de los costes de sus aristas.

Dados dos vértices $s$ y $t$, y un valor $C$, implementar un algoritmo que encuentre un camino desde $s$ hasta $t$ con una capacidad de flujo mayor o igual que $C$ o que diga que tal camino no existe. La complejidad en tiempo de la solución propuesta debe estar en $O(|V| + |A|)$.

**Ejercicio 18** Demostrar que los caminos mínimos construidos por el algoritmo de Dijkstra para un grafo no dirigido y conexo forman un *árbol de recubrimiento* del grafo. ¿Se trata de un árbol de recubrimiento mínimo? Justificar la respuesta.

**Ejercicio 19** Dado el siguiente grafo dirigido y valorado, ejecutar sobre él el algoritmo de Dijkstra a partir del vértice 1, e indicar en una tabla de tantas líneas como se necesiten: (a) la iteración en la que se incorpora cada vértice al conjunto de vértices seleccionados; (b) la distancia mínima

calculada para cada vértice; (c) el camino mínimo calculado para el mismo; y (d) las distancias mínimas provisionales a los vértices que aún no han sido seleccionados.




**Ejercicio 20** Modificar el algoritmo de Dijkstra para que además indique para cada vértice *cuántos* caminos mínimos hay a él desde el vértice origen. ¿Cambia el coste del algoritmo?

**Ejercicio 21** Dado un grafo dirigido y valorado, junto con un vértice origen $v_0$, escribir un algoritmo que modifique lo menos posible el de Dijkstra de forma que devuelva los caminos de mínimo coste de $v_0$ a cualquier otro vértice, pero garantice además que, en caso de haber más de un camino de mínimo coste, devuelva el que tenga menor número de aristas.

**Ejercicio 22** Durante la Segunda Guerra Mundial y con el fin de poder transportar pertrechos con comodidad, los japoneses construyeron puentes de dirección única que conectaban determinados pares de islas. De cada uno conocían su *limitación de peso*. Establecida su base de operaciones en la isla de Midway, se preguntaron cuál sería el *peso máximo* de los tanques que podrían trasladar desde Midway hasta cada isla destino, y cuál sería *la ruta de puentes* que deberían utilizar en cada caso.

1. Proporcionar un algoritmo que resuelva el problema de forma óptima.
2. La aviación americana destruyó el puente entre las islas $A$ y $B$ y los japoneses debían mandar una partida de peso $P$ a la isla $C$. El Coronel, tras consultar brevemente la salida del algoritmo, tranquilizó a sus subordinados indicándoles que el ataque no afectaría a ese envío. ¿Cómo llegó el Coronel a esa conclusión?
3. Tras consultar también muy brevemente la citada salida, el General les indicó que en realidad la destrucción de ese puente no alteraría sus planes de suministros a *ninguna* de las islas del archipielago. ¿Cómo llegó el General a esa conclusión?

**Ejercicio 23** Dado un grafo $G$ no dirigido y conexo, se llama *punto de articulación* de $G$ a un vértice tal, que si se suprime el mismo y sus aristas asociadas, el grafo deja de ser conexo. Estudiar la relación entre dichos vértices y la información que proporciona el algoritmo DFS, y escribir un algoritmo basado en DFS que determine si un vértice $v$ dado es o no un punto de articulación. Decir cuál es su coste en tiempo.

**Ejercicio 24** En *Silvania* disponen de una red de caminos bidireccionales que conectan directa o indirectamente todas las ciudades que la conforman. Desgraciadamente, el ejercito de *Transilvania* es mucho más potente y podrá hacer caer una a una las ciudades de Silvania, destruyendo como consecuencia los caminos que unen en cada caso la ciudad conquistada con la parte aun no conquistada. Los habitantes de Silvania pueden disponer sus defensas de manera que la conquista tenga que realizarse en el orden que ellos decidan. Demuestra que podrán escoger un orden tal, que durante todo el proceso se mantengan totalmente conectadas (por caminos aún no destruídos) las ciudades que sigan formando parte de la Silvania *libre*. Diseña un algoritmo eficiente que fije el orden en que los silvanos irán dejando caer sus ciudades.

**Ejercicio 25** Disponemos de una estructura de datos $H$ con los horarios de todos los vuelos interiores disponibles en España, un determinado día. Para simplificar, supondremos que las horas de salida y los tiempos de vuelo vienen dados en minutos (entre 0 y 1440) y que ningún vuelo llega más tarde de las 12 de la noche.

Se pide implementar un algoritmo que modifique lo menos posible el de Dijkstra, que recibiendo $H$, el aeropuerto de origen, y un valor $t_0$ que representa la hora de llegada a dicho aeropuerto, devuelva la **hora mínima** posible de llegada a cada una de las restantes ciudades y la ruta de transbordos seguida. Para simplificar, supondremos que un enlace es posible si llegamos a un aeropuerto no más tarde de la hora de salida del vuelo siguiente. Suponer la existencia de las operaciones consultoras razonables sobre $H$ que necesitéis, que por tanto no será necesario implementar.

**Ejercicio 26** Dado un grafo $G = (V, A)$ **no** dirigido, valorado y **conexo**, el siguiente algoritmo devuelve un *árbol de recubrimiento mínimo* de $G$:

$T = \{\}$
**para cada** $v_i \in V$ **hacer**
  $V_i = \{v_i\}$
  $E_i = \{(v_i, v) \in A\}$
**mientras** tengamos mas de un conjunto $V_i$ **hacer**
  elegir $V_i$, uno cualquiera de los conjuntos de la coleccion
  $(u, v) =$ arista de minimo coste de $E_i$, con $u \in V_i$ y $v \in V_j$
  **si** $i \neq j$ **entonces**
    $T = T \cup \{(u, v)\}$
    $V_i = V_i \cup V_j$
    $E_i = E_i \cup E_j$
    Se elimina $V_j$ de la coleccion de conjuntos
**retorna** $T$

Elige las estructuras de datos que consideres más apropiadas para representar los conjuntos de vértices $V_i$ y de aristas $E_i$, y cualquier otra estructura auxiliar que necesites para completar tu implementación, indicando las operaciones que usarás de cada una y su coste. En base a lo anterior, calcula el coste total del algoritmo y compáralo con los costes de los algoritmos de Kruskal y Prim, que resuelven este mismo problema.

**Ejercicio 27** Dado un grafo $G = (V, A)$ no dirigido, valorado y conexo, se tienen las siguientes versiones abstractas de los algoritmos de Kruskal y Prim, que devuelven un *árbol de recubrimiento mínimo*:

**funcion** Kruskal (V,A) **retorna** T
  $T = \{\}$; $n = |V|$; L = aristas_por_costes_crecientes(A);
  **mientras** $|T| < n - 1$ **hacer**
    $a$ = L.min_coste(); L.borra_min();
    **si** no_hay_ciclos($T \cup \{a\}$) **entonces**
      $T = T \cup \{a\}$
**retorna** $T$

**funcion** Prim (V,A) **retorna** T
  $T = \{\}$; $n = |V|$; $V_T = \{1\}$; salen=$\{(1, i) \in A \mid 2 \leq i \leq n\}$;
  **repetir** $n - 1$ **veces**
    $a$ = salen.min_coste(); j = vertice de $a$ que no esta en $V_T$;
    salen = (salen$- \{(i, j) \in$ salen$\mid i \in V_T\}) \cup \{(j, k) \in A \mid k \notin V_T\}$
    $T = T \cup \{a\}$; $V_T = V_T \cup \{j\}$;
**retorna** $T$

Indicar cuál sería el comportamiento de cada uno de los algoritmos anteriores, en el caso de que el grafo suministrado no fuera conexo. Si necesitas precisar cómo está implementado el grafo, o cómo se implementan las funciones que aparecen en el seudocódigo, indica las hipótesis utilizadas. A continuación, indica las modificaciones necesarias en cada uno de ellos, para que los algoritmos devuelvan un resultado razonable, incluso cuando el grafo dado no sea conexo. ¿Cuál sería dicho resultado en cada caso?

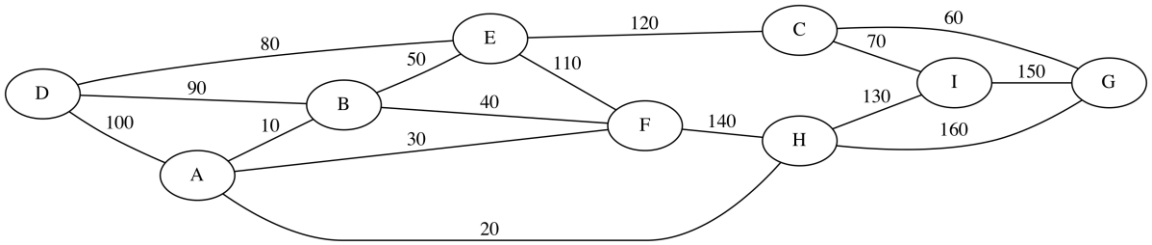
**Ejercicio 28** Decimos que un grafo dirigido es *redondo*, cuando contiene un ciclo en el que aparecen todos sus vértices, y ningún otro ciclo de longitud menor.

1. Dado un grafo dirigido y un vértice $v$ cualquiera del mismo, aplicar una variante del algoritmo de Dijkstra, o de la exploración en anchura, para calcular la longitud mínima de un ciclo que contenga al vértice $v$. El algoritmo devolverá un valor imposible de no existir ninguno.
2. Dar un algoritmo que decida si un grafo dirigido es redondo y calcular su complejidad.

**Ejercicio 29** Se conoce la estructura de una compleja red de datos que conecta un conjunto de nodos de comunicación $\{1, \ldots, n\}$. Los enlaces entre los nodos son unidireccionales, y cada enlace $a \to b$ tiene asociado un grado $g_{a,b} \geq 0$ de calidad. Una comunicación entre dos nodos distantes $v_1,\ v_2$ se realiza eligiendo una secuencia de enlaces que conectan $v_1$ y $v_2$, posiblemente utilizando nodos intermedios. Quien determina la calidad de la comunicación es el enlace de menor calidad. Se pide implementar un algoritmo que modifique lo menos posible el de Dijkstra, que calcule los mejores caminos para comunicar un nodo fijo $v_0$ con todos los demás nodos de la red.

Se pide también un razonamiento detallado y riguroso de por qué los caminos calculados son óptimos.

**Ejercicio 30** Considera el siguiente grafo no dirigido y valorado:




1. Da el orden en el que las aristas se añaden al ARM (árbol de recubrimiento de coste mínimo) utilizando el algoritmo de Kruskal.
2. Da el orden en el que las aristas se añaden al ARM utilizando el algoritmo de Prim, considerando $A$ como vértice inicial.
3. Supón que se añade la arista $H$—$C$ al grafo. ¿Qué valoraciones de $H$—$C$ harían que esta nueva arista perteneciera al ARM?

**Ejercicio 31** Dado un grafo no dirigido, conexo, valorado y en el que no hay dos aristas que tengan el mismo coste, razonar con el mayor rigor posible que los algoritmos de Kruskal y Prim necesariamente calcularán el mismo árbol de recubrimiento de coste mínimo.

**Ejercicio 32** Nos dan un grafo $G = \langle V, A \rangle$ no dirigido, valorado y conexo y un ARM (árbol de recubrimiento de coste mínimo) para él. También nos dan una arista nueva $(u, v) \notin A$, con un valor asociado $c$. Se pide un algoritmo de coste en $O(|V|)$ que modifique el ARM de manera que el resultado sea un ARM para $G' = \langle V, A \cup \{(u, v)\} \rangle$. Razona la corrección y el coste.

**Ejercicio 33** Nos dan dos vasijas de capacidades $m$ y $n$ litros, respectivamente, y una cantidad a conseguir $c < n$. Las únicas operaciones permitidas sobre las vasijas son:

1. Llenar completamente una vasija, si está previamente vacía.
2. Vaciar completamente una vasija.
3. Trasvasar el contenido de una vasija a la otra hasta que una de las dos quede completamente llena o completamente vacía.

Con $m = 4,\ n = 3,\ c = 2$, detallar un grafo en el que los estados de las vasijas sean los vértices y las transiciones legales entre estados sean las aristas. Implementar un algoritmo inspirado en el recorrido en anchura de un grafo dirigido que, dados $m, n$ y $c$, encuentre la secuencia más corta de transiciones que conduzca desde el estado inicial, en el que ambas vasijas están vacías, hasta un estado final, si este existe, en el que la cantidad $c$ está en una de las vasijas y la otra vasija está vacía.

**Ejercicio 34** Consideremos un grafo $G$ no dirigido, conexo y valorado. ¿Cuáles de los siguientes métodos sirven para calcular un *árbol de recubrimiento de coste máximo* de tal grafo?

1. Aplicar una versión modificada del algoritmo de Kruskal en la que las aristas se eligen de mayor a menor coste.
2. Aplicar una versión modificada del algoritmo de Prim en la que la cola de prioridad usada es de máximos y en la que se cambian las comparaciones $\leq$ por comparaciones $\geq$.
3. Transformar el grafo $G$ en $\overline{G}$ a base de invertir el signo de todos los valores y aplicar a continuación al algoritmo de Kruskal o de Prim estándar a $\overline{G}$ para calcular su árbol de recubrimiento de coste mínimo.