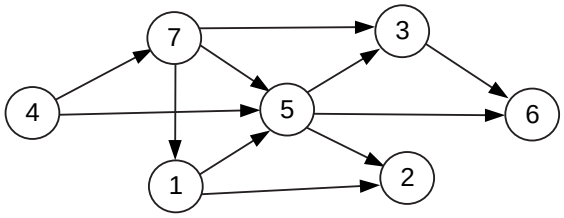

GradoEnt

1	2	3	4	5	6	7
1	2	2	**0**	3	2	1
1	2	2	0	**2**	2	**0**
0	2	**1**	0	**1**	2	0
0	**1**	1	0	**0**	2	0
0	**0**	**0**	0	0	**1**	0
0	0	0	0	0	1	0
0	0	0	0	0	**0**	0
0	0	0	0	0	0	0

orden

1	2	3	4	5	6	7
-	-	-	-	-	-	-
-	-	-	**1**	-	-	-
-	-	-	1	-	-	**2**
3	-	-	1	-	-	2
3	-	-	1	**4**	-	2
3	5	-	1	4	-	2
3	5	6	1	4	-	2
3	5	6	1	4	7	2

C : Cola[entero]

{ 4 }
{ 7 }
{ 1 }
{ 5 }
{ 2 , 3 }
{ 3 }
{ 6 }
{ }

contador

-
1
2
3
4
5
6
7

Figura 5.32: Ordenación topológica en un GDA. Arriba: GDA de ejemplo. Abajo: ejecución del algoritmo sobre el ejemplo.

a bbp(v) tendríamos $orden[v] = n$, para el siguiente $n - 1$, y así sucesivamente. El orden obtenido es un orden topológico porque un nodo sólo acabará su llamada recursiva cuando todos sus adyacentes hayan sido visitados.

5.6.3. Flujo máximo en redes

En teoría de grafos, un grafo dirigido con pesos es también conocido como una **red**. En los problemas de flujo en redes, las aristas representan canales por los que puede circular cierta cosa: datos, agua, coches, corriente eléctrica, etc. Los pesos de las aristas representan la capacidad máxima de un canal: velocidad de una conexión, volumen máximo de agua, cantidad máxima de tráfico, voltaje de una línea eléctrica, etc.; aunque es posible que la cantidad real de flujo sea menor.

El **problema del flujo máximo** consiste en lo siguiente: dado un grafo dirigido con pesos, $G = (V, A, W)$, que representa las capacidades máximas de los canales, un nodo de inicio s y otro de fin t en V, encontrar la cantidad máxima de flujo que puede circular desde s hasta t. En la figura 5.33 se muestra un ejemplo de problema y la solución. El grafo de la izquierda, G, pintado con líneas continuas, representa las capacidades máximas; sería la entrada del problema. El grafo de la derecha, F, representado con líneas discontinuas, indica los flujos reales; es una posible solución para el problema.

La solución del problema debe cumplir las siguientes propiedades:

- La suma de los pesos de las aristas que salen de s debe ser igual a la suma de las aristas que llegan a t. Esta cantidad es el **flujo total** entre s y t.
- Para cualquier nodo distinto de s y de t, la suma de las aristas que llegan al nodo

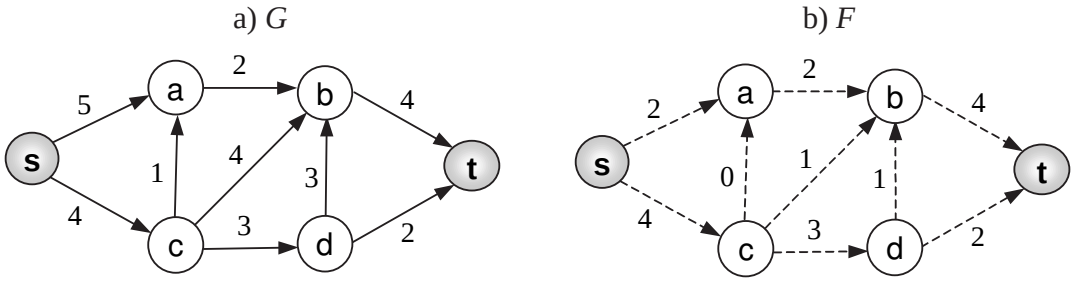

Figura 5.33: Problema de flujo máximo en redes. a) Grafo de capacidades máximas de los canales. b) Solución del problema, grafo de flujos reales.

debe ser igual a la suma de las aristas que salen al mismo.

- Los pesos de las aristas en F no pueden superar los pesos máximos indicados en G. Es decir, si $C_G(a,b)$ es el peso de la arista $<a,b>$ de G y $C_F(a,b)$ es el peso de la misma arista en F, entonces $C_F(a,b) \leq C_G(a,b)$.

Una vez planteado el problema, vamos a estudiar la forma de resolverlo. En primer lugar propondremos un algoritmo intuitivo, y a continuación analizaremos si garantiza la solución óptima o no.

Un posible algoritmo para calcular el flujo máximo

La idea de los flujos, que van desde s hasta t, es muy próxima a la de un camino por el que circula cierto fluido. Cada unidad de flujo que llega hasta un nodo, debe salir por alguna de sus aristas. Por lo tanto, un posible algoritmo podría basarse en encontrar caminos $(s, v_1, v_2, ..., v_k, t)$ en G. Por ese camino podemos mandar cierta cantidad de flujo. ¿Cuánto? Pues todo lo que quepa. Por ejemplo, si en el grafo G de la figura 5.33 tomamos el camino (s, a, b, t), vemos que las aristas por las que pasa tienen pesos: 5, 2, 4. El máximo flujo que podemos mandar por ese camino está limitado por el mínimo de las capacidades por las que pasa el camino; en este caso 2. De esta forma, el algoritmo iría encontrando caminos en G, añadiendo los flujos correspondientes al grafo F y quitándolos de G. Y así seguiría hasta que no queden más caminos para enviar flujo.

La estructura del algoritmo que lleva a cabo esta idea sería la siguiente:

1. Sea $G = (V, A, C_G)$ el grafo de capacidades máximas. Inicializar el grafo de flujos reales, F, con los mismos nodos y aristas de G, pero con pesos 0. Es decir, $C_F(v,w) = 0$; $\forall <v,w> \in A$. Este grafo guardará el resultado del algoritmo.

2. Buscar un camino en G, desde s hasta t, pasando por aristas cuyo peso sea mayor que 0. Este camino es denominado **camino creciente**. Supongamos que el camino es $(s, v_1, v_2, ..., v_k, t)$. Tomamos $m = min\{C_G(s, v_1), C_G(v_1, v_2), ..., C_G(v_k, t)\}$. Es decir, por este camino pueden fluir hasta m unidades de flujo, como máximo.

3. Para cada arista $<v,w>$ del camino anterior, añadir m al coste de la arista correspondiente en F y quitarlo en G. Es decir, $C_F(v,w) = C_F(v,w) + m$; $C_G(v,w) =$

$C_G(v, w) - m$; para todo <v,w> del camino del paso 2.

4. Volver al paso 2 mientras siga existiendo algún camino creciente entre s y t en G.

Todavía nos queda por determinar la forma de encontrar el camino creciente del paso 2. Una vez más, la búsqueda primero en profundidad puede sernos de utilidad. Para encontrar un camino creciente, podríamos iniciar una búsqueda en profundidad en G a partir del nodo s. Cuando la búsqueda llegue a t ya tenemos un camino de s a t[12]. Además, el procedimiento bpp debería ser modificado para tener en cuenta sólo las aristas con peso mayor que cero.

Por otro lado, está claro que entre s y t puede haber más de un camino creciente. Esta primera versión del algoritmo indica que se encuentre un camino cualquiera. El algoritmo será óptimo si, independientemente de los caminos elegidos en el paso 2, siempre encuentra el flujo máximo. Vamos a ver que no siempre ocurre así.

Ejemplo 5.6 Vamos a aplicar la primera versión del algoritmo del flujo máximo sobre el grafo G de la figura 5.33a). En la figura 5.34 se muestra una ejecución posible del algoritmo, donde no se alcanza la solución óptima.

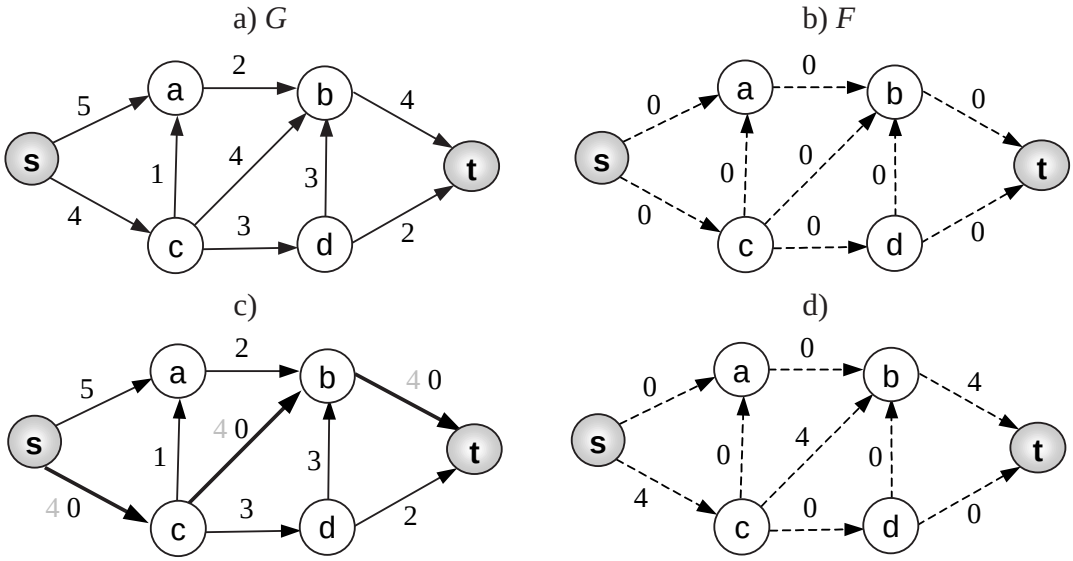

Figura 5.34: Primera versión del algoritmo de flujo máximo en redes. a) Grafo del problema, G. b) Grafo de flujos F inicial. c) Grafo G después de quitar el camino (s, c, b, t). d) Grafo de flujos después de añadir el mismo camino.

En la primera ejecución del paso 2, se encuentra el camino (s, c, b, t). Los costes de las aristas son: 4, 4, 4; así que $m = 4$. Esta cantidad se añade en F (figura 5.34d) y se quita de G (figura 5.34c). Si intentamos buscar otro camino entre s y t, en el grafo de la figura 5.34c), que pase por aristas con peso mayor que cero, vemos que no existe ninguno. Por

[12]El camino estaría en la pila de llamadas recursivas. Lo más adecuado sería ir almacenando en un array los nodos que están en la rama actual de la llamada a bpp.

lo tanto, el algoritmo acabaría. El resultado del algoritmo es que el flujo total encontrado es 4.

En consecuencia, el algoritmo no encuentra el óptimo, que como vimos en la figura 5.33 es 6 unidades de flujo. No obstante, si los caminos hubieran sido elegidos en otro orden sí que se habría obtenido el óptimo. En concreto, se puede comprobar que el resultado de la figura 5.33 se alcanzaría si seleccionamos los siguientes caminos, por orden: (s, a, b, t) con peso 2; (s, c, d, t) con peso 2; (s, c, d, b, t) con peso 1; (s, c, b, t) con peso 1.

Algoritmo de flujo máximo deshaciendo caminos

La primera versión del algoritmo es no determinista: en el paso 2 se pueden elegir varios caminos y, dependiendo de cuál se coja, el algoritmo alcanza la solución óptima o no. Para solucionar el problema podemos hacer una pequeña modificación en el algoritmo. El sentido de esta modificación es que si se coge un camino, pero que luego resulta ser una mala decisión, se pueda **deshacer el flujo** enviado por ese camino.

En particular, la modificación afecta a la forma de actualizar C_G dentro del paso 3. Cada vez que encontramos un camino creciente, quitamos m unidades de flujo de G y las ponemos en F. Ahora, además, vamos a indicar en G que se pueden deshacer m unidades de flujo a través de las aristas del camino. El flujo que se deshace tendrá el sentido opuesto al de añadir; es decir, si se añade m unidades en $<v,w>$ en F, entonces se quitan m de $<v,w>$ en G y se añaden m unidades de deshacer para la arista $<w,v>$ en G.

En definitiva, este cambio sólo implica modificaciones dentro del paso 3 del algoritmo, que ahora debería decir:

> 3 Para cada arista $<v,w>$ del camino anterior, añadir m al coste de la arista correspondiente en F, quitarlo en G y ponerlo en G en sentido contrario. Es decir, $C_F(v, w) = C_F(v, w) + m$; $C_G(v, w) = C_G(v, w) - m$; $C_G(w, v) = C_G(w, v) + m$ para todo $<v,w>$ del camino del paso 2.

Hay que tener en cuenta que aquí estamos suponiendo que el peso de una arista inexistente es 0. De esta forma, cuando sumamos m a $C_G(w, v)$, pero $<v,w>$ no está en G, sería equivalente a crear una nueva arista con peso m.

Esta nueva versión del algoritmo no deja de ser no determinista, pero garantiza siempre la solución óptima. Aunque no lo vamos a demostrar, vamos a ver que se resuelve correctamente el problema que vimos en el ejemplo 5.6.

Ejemplo 5.7 Vamos a aplicar la segunda versión del algoritmo del flujo máximo –la que permite deshacer caminos– sobre el grafo G de la figura 5.33a). Un posible resultado del algoritmo se muestra en la figura 5.35.

Igual que en el ejemplo 5.6, consideramos que en la primera ejecución del paso 2 se encuentra el camino (s, c, b, t), con $m = 4$. Esta cantidad se añade en F (figura 5.35d). Ahora, en G se quita esa cantidad en sentido directo y se añade en sentido contrario (figura 5.35c).

A continuación podemos encontrar un nuevo camino, que pasa por la arista de "deshacer" $<b,c>$. El camino es (s, a, b, c, d, t), con pesos: 5, 2, 4, 3, 2. Por lo tanto,

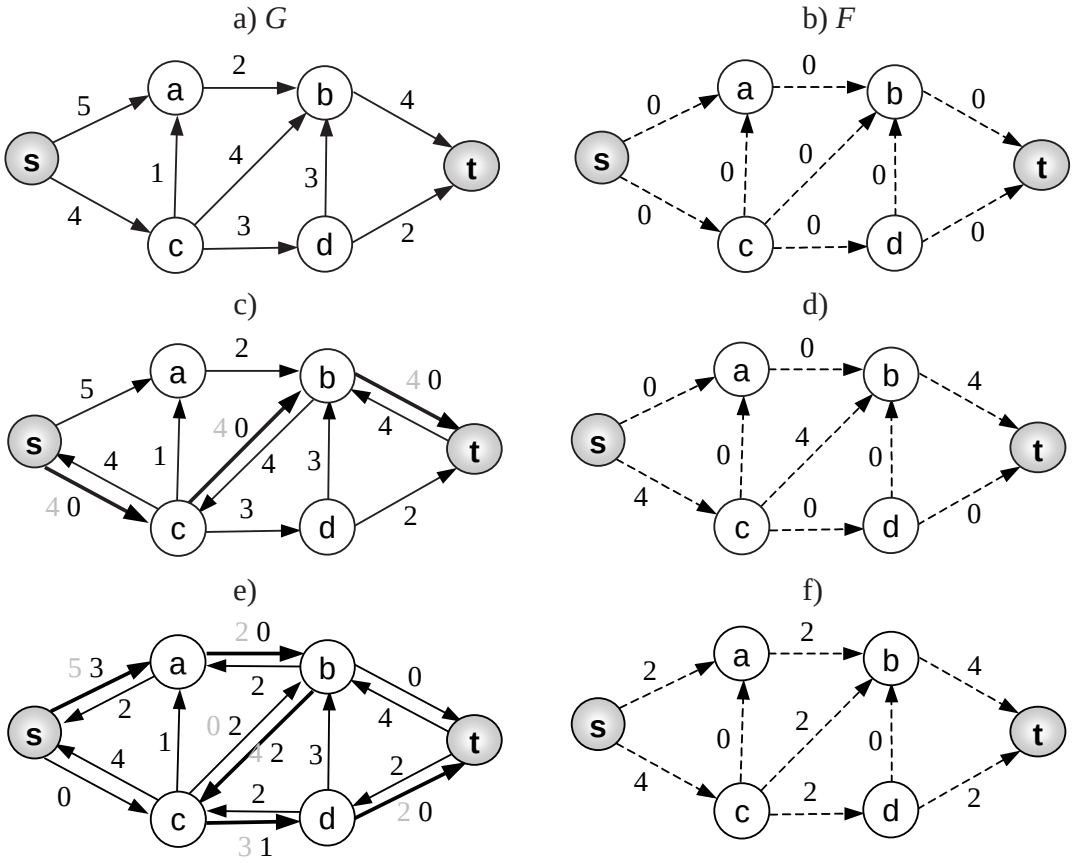

Figura 5.35: Segunda versión del algoritmo de flujo máximo en redes. a) Grafo del problema, G. b) Grafo de flujos F inicial. c),d) Grafos G y F, después de encontrar el camino (s, c, b, t). e),f) Grafo G y F, después de encontrar el camino (s, a, b, c, d, t).

$m = 2$. Se añade a F[13] (figura 5.35f) y se actualiza G (figura 5.35e). En el siguiente paso, ya no existe ningún camino creciente, luego acaba el algoritmo.

Si comparamos la solución obtenida con la mostrada en la figura 5.33, vemos que no coinciden. No obstante, ambas tienen el mismo valor de flujo total, 6, y ambas son óptimas. En perfectamente posible, como en este ejemplo, que la solución óptima no sea única.

5.7. Algoritmos sobre grafos no dirigidos

En esta sección plantearemos dos problemas específicos de grafos no dirigidos: la búsqueda de los puntos de articulación y los circuitos de Euler. Vamos a ver que ambos problemas se pueden resolver utilizando como herramienta la búsqueda primero en profundidad.

[13]Hay que notar un detalle sutil. Cuando en el grafo de la figura 5.35d) se añade el flujo de "deshacer" entre b y c, no se añade realmente 2 a $C_F(b, c)$, sino que se resta 2 de $C_F(c, b)$. ¿Por qué?

5.7.1. Puntos de articulación y componentes biconexos

Como sabemos, un grafo no dirigido se dice que es conexo si existen caminos entre todos sus nodos. Pero en muchas aplicaciones se requiere un nivel más de conexión; se necesita no sólo que los nodos estén conectados sino que si falla algún nodo o enlace, los nodos sigan conectados. Buscamos lo que se podría denominar *tolerancia a fallos*. En la figura 5.36 se muestran dos ejemplos de aplicación. El grafo de la figura 5.36a) muestra las estrategias de pase del balón de un equipo de fútbol; la figura 5.36b) muestra una red de ordenadores y las conexiones entre los mismos.

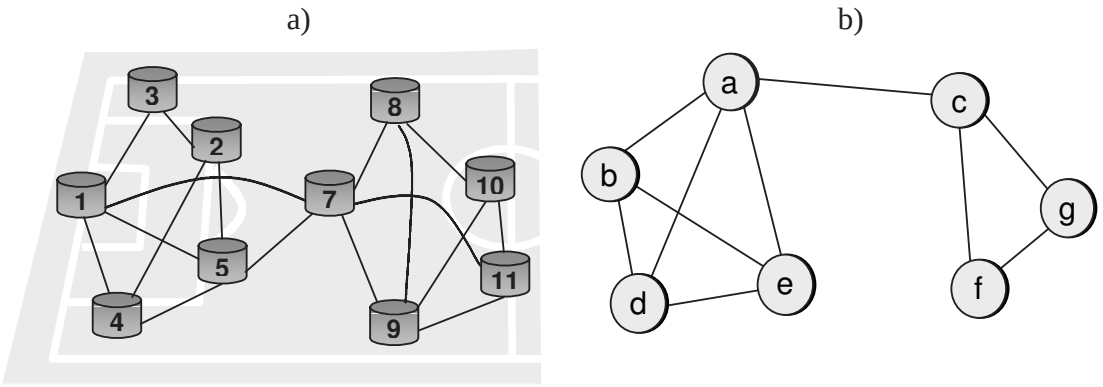

Figura 5.36: Ejemplos de grafos no dirigidos. a) Jugadores de un equipo de fútbol (nodos) y estrategias de pase del balón (aristas). b) Red de ordenadores.

En este tipo de aplicaciones surge el estudio de los puntos de articulación y los componentes biconexos.

Definición 5.16 Sea un grafo no dirigido y conexo, $G = (V, A)$. Un **punto de articulación** es un vértice v tal que cuando se elimina de G junto con todas las aristas incidentes en él, se divide una componente conexa de G en dos o más.

El punto de articulación es un nodo *crítico* del grafo, en el sentido de que si falla tendremos graves problemas. Por ejemplo, en la figura 5.36a) el jugador 7 es un punto de articulación; si conseguimos eliminarlo cortaremos la circulación del balón entre la defensa y la delantera. En la figura 5.36b) los ordenadores `a` y `c` son puntos de articulación; si cualquiera de ellos cae, quedarán trozos de red incomunicados.

Un grafo no dirigido se dice que es **biconexo** si no tiene puntos de articulación. En los anteriores apartados hemos hablado informalmente de grafos más o menos conectados. Podemos definir el concepto de conectividad de un grafo de la siguiente forma.

Definición 5.17 Un grafo no dirigido se dice que tiene **conectividad** k si la eliminación de $k - 1$ vértices cualesquiera, junto con las aristas incidentes en ellos, no desconecta el grafo resultante.

De acuerdo con la definición, un grafo tiene conectividad 2 o más si y sólo si no tiene puntos de articulación, es decir, si es biconexo. Por ejemplo, los grafos de la figura 5.36

no son biconexos, por lo que su conectividad es 1. Cuanto mayor sea la conectividad del grafo, más fácil será que *sobreviva* al fallo de alguno de sus vértices. Por otro lado, según la definición, si un grafo tienen conectividad k, también tendrá conectividad $k-1$, $k-2$, ..., 1.

Ejemplo 5.8 ¿Cuánto es la máxima conectividad posible de un grafo? ¿A qué grafo corresponde? ¿Cuántas aristas debe tener un grafo como mínimo para ser biconexo?

La máxima conectividad posible sería la de un grafo completo. Podemos eliminar todos los vértices sin desconectarlo. Sólo cuando eliminemos todos los nodos conseguimos suprimir un componente conexo. Podemos decir que el grafo completo tiene conectividad n, siendo n el número de nodos del grafo.

En cuanto al mínimo número de aristas, sabemos que un grafo con forma de árbol es el menor grafo conexo posible. Un grafo de ese tipo tiene $n-1$ aristas, pero no es biconexo. Todos los nodos, excepto las hojas, serían puntos de articulación. Consideremos un grafo con forma de anillo. Tiene n aristas y es biconexo, ya que no tiene puntos de articulación. Por lo tanto, el mínimo número de aristas para un grafo biconexo sería n. Ojo, esto no garantiza que cualquier grafo no dirigido con n aristas sea biconexo.

Algoritmo para calcular los puntos de articulación

A falta de una idea mejor, una posible solución para calcular los puntos de articulación de un grafo podría ser la siguiente: eliminar los vértices del grafo uno por uno; para cada nodo eliminado, comprobar si el grafo resultante sigue siendo conexo o no; en caso negativo, tenemos un punto de articulación. La comprobación de si el grafo resultante es conexo se podría hacer con una búsqueda primero en profundidad. En consecuencia, el tiempo de esta solución sería $O(n^3)$ con matrices de adyacencia y $O(n\ (a+n))$ con listas. Pero vamos a ver que el problema se puede resolver con una simple búsqueda en profundidad.

Supongamos que hacemos la búsqueda en profundidad de un grafo no dirigido y conexo, como el de la figura 5.36b). Obtenemos un árbol abarcador en profundidad, como el mostrado en la figura 5.37.

En el resultado aparecen dos tipos de arcos: los del árbol y los que no son del árbol, que serán de avance/retroceso. Podemos interpretar que tenemos dos tipos de caminos entre los nodos: los caminos a través del árbol, hacia los padres, y los caminos moviéndonos a través de los arcos de retroceso. Los primeros, a través del árbol, siempre existirán; los segundos, a través de arcos de retroceso, representan caminos *alternativos*. Decimos que son alternativos en el sentido de que si eliminamos un nodo en el camino del primer tipo, tenemos una vía alternativa.

El algoritmo se basa en el cálculo de los caminos alternativos. Se realiza una búsqueda primero en profundidad, numerando los nodos en el orden de recorrido. Al mismo tiempo se calculan los "caminos alternativos". El resultado se guarda en un array *bajo*: **array** [1..*n*] **de** entero. El valor de *bajo* de un nodo *v* indica lo más arriba que podemos llegar en el árbol a través de un camino alternativo. El camino alternativo para un nodo consistirá en moverse a través de un arco de retroceso, o bien hacia abajo en el árbol y luego hacia arriba por un arco de retroceso. En la figura 5.37b) se muestra un ejemplo.

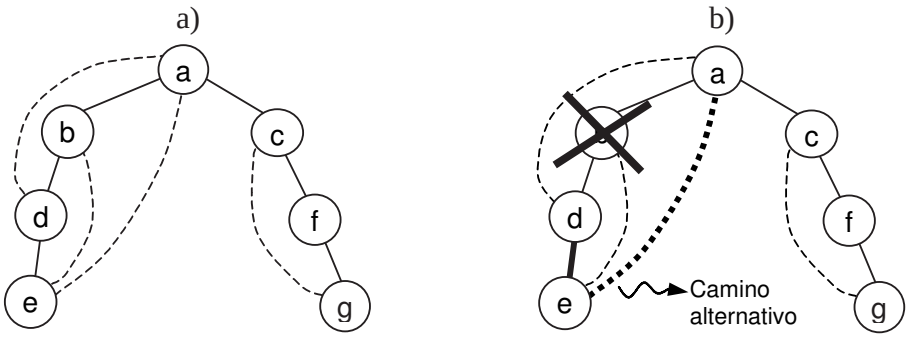

Figura 5.37: Caminos alternativos. a) Árbol de expansión en profundidad del grafo de la figura 5.36b). b) Si eliminamos el nodo b, existe para d y para e un camino alternativo para llegar a la raíz. Conclusión: b no es punto de articulación.

Tendremos un punto de articulación si, al eliminar un nodo, alguno de sus hijos no tiene un camino alternativo para llegar más arriba del nodo eliminado. Es decir, v será un punto de articulación si tiene un hijo tal que su valor de *bajo* es menor o igual que el número de búsqueda en profundidad de v. La condición es distinta para la raíz; será punto de articulación si tiene dos o más hijos. Como ya vimos, no pueden haber arcos de cruce, por lo que si tiene dos o más hijos estos sólo se pueden comunicar a través de la raíz.

En definitiva, el **algoritmo para calcular los puntos de articulación** de un grafo no dirigido tendría la siguiente estructura:

1. Realizar una búsqueda primero en profundidad, numerando los nodos en el orden en que son recorridos. Supongamos que guardamos en el array *nbpp*: **array** [1..*n*] **de** entero, el orden en que es visitado cada nodo.

2. Calcular los valores *bajo*[*v*] para cada nodo visitado, según la fórmula:
   bajo[*v*]:= mínimo {*nbpp*[*v*];
   nbpp[*z*] | para todo *z* tal que exista un arco de retroceso (*z*,*v*);
   bajo[*y*] | para todo *y* hijo de *v* en el árbol de expansión}

3. La raíz del árbol es un punto de articulación si y sólo si tiene dos o más hijos en el árbol de expansión en profundidad.

4. Un nodo *v*, distinto de la raíz, es un punto de articulación si y sólo si tiene algún hijo *w* en el árbol tal que *bajo*[*w*] $\geq$ *nbpp*[*v*].

Realmente, el cálculo de los valores de *bajo* y la comprobación de las condiciones no se deben hacer como pasos separados, sino dentro del mismo procedimiento de bpp. Como los cálculos añadidos se pueden hacer en tiempo constante, el orden de complejidad del algoritmo viene dado por el orden de la búsqueda primero en profundidad: $O(n^2)$ con matrices de adyacencia y $O(n + a)$ con listas de adyacencia.

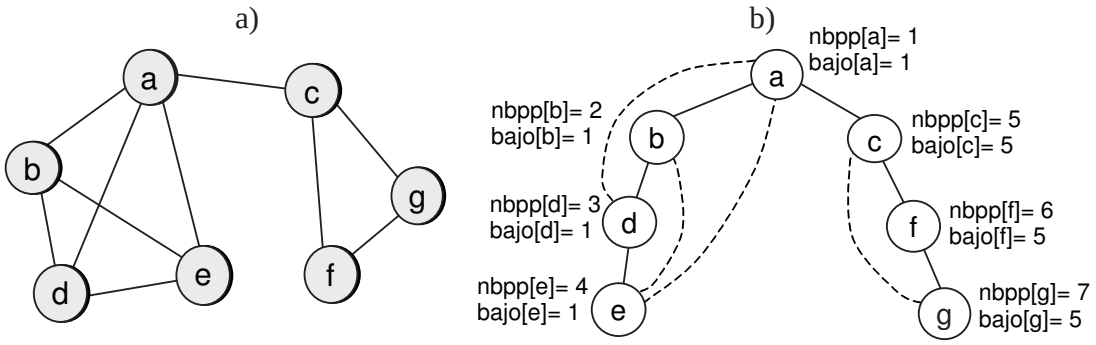

Figura 5.38: Búsqueda de los puntos de articulación. a) Grafo no dirigido. b) Aplicación del algoritmo. Se indican los valores de *bajo* y *nbpp*.

Ejemplo 5.9 En la figura 5.38 se muestra un ejemplo de aplicación de este algoritmo, para el mismo grafo de la figura 5.36a).

¿Cuáles son los puntos de articulación del ejemplo de la figura 5.38? Para la raíz, a, se cumple la condición del punto 3, puesto que tiene dos hijos. Por lo tanto, a es un punto de articulación. Para los demás nodos, tenemos que comprobar la condición del punto 4. Vemos que se cumple únicamente para el nodo c; tiene como hijo el nodo f, cuyo $bajo[f]$=5 $\geq$ $nbpp[c]$=5. En consecuencia, f es otro punto de articulación.

Componentes biconexos

La definición de componente biconexo es similar a la de componente conexo. Un **componente biconexo** de un grafo G es un subgrafo biconexo y maximal de G. Si G es de por sí biconexo, entonces tendrá un sólo componente biconexo. Si G tiene puntos de articulación, entonces aparecerán distintos componentes biconexos. ¿Cómo encontrarlos?

En primer lugar, deberíamos calcular los puntos de articulación de G. Después, el algoritmo sería similar al cálculo de los componentes conexos que vimos en el ejemplo 5.4. Recordemos que este algoritmo se basaba en una simple búsqueda primero en profundidad. La única diferencia es que no se deberían hacer llamadas recursivas al llegar a un nodo que sea punto de articulación.

En la figura 5.39 se muestran los componentes biconexos del grafo del ejemplo 5.9. Se puede ver que algunos nodos están en más de un componente biconexo. Esto ocurrirá, precisamente, para los nodos que sean puntos de articulación.

5.7.2. Circuitos de Euler

Los problemas de circuitos de Euler aparecen cuando se utilizan grafos para representar dibujos de líneas. En la figura 5.40 se muestran tres de estos ejemplos. En estos grafos, cada nodo representa un punto del dibujo y una arista entre dos nodos indica que existe una línea entre los dos puntos correspondientes.

El problema del circuito de Euler trata de responder a la cuestión: ¿es posible dibujar la figura con un bolígrafo, pintando cada línea una sola vez, sin levantar el bolígrafo y

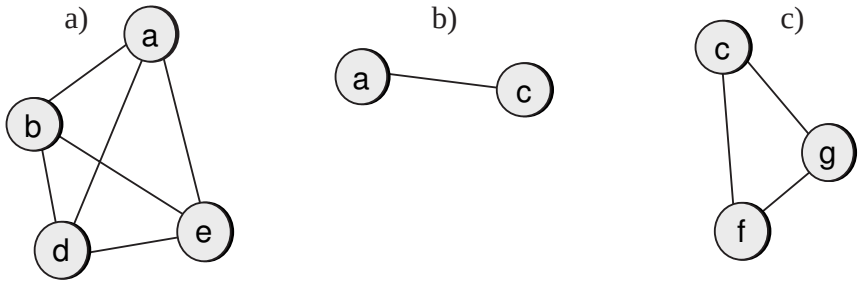

Figura 5.39: Componentes biconexos del grafo del ejemplo 5.9.

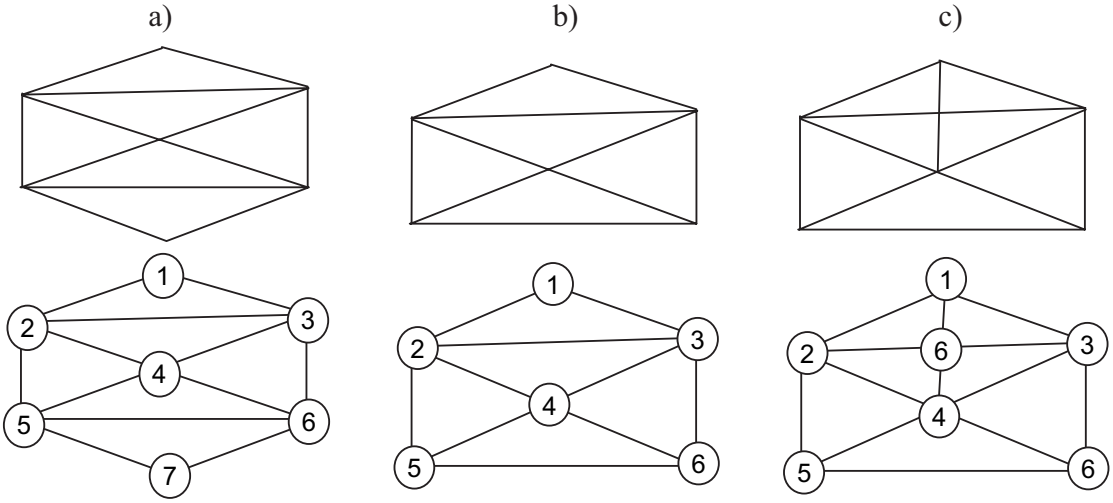

Figura 5.40: Circuitos y caminos de Euler. Arriba: dibujo de líneas. Abajo: grafo correspondiente. a) Tiene un circuito de Euler. b) Tiene un camino de Euler. c) No tiene circuito ni camino de Euler.

acabando en el mismo punto donde se empezó?

Formulado con terminología de grafos, decimos que un **circuito de Euler** es un ciclo, no necesariamente simple, que visita todas las aristas exactamente una vez. Es decir, los nodos se pueden visitar varias veces, pero las aristas se tienen que visitar una y solo una vez. Así que dado un grafo, tenemos que decidir si existe un circuito de Euler.

Hay claramente dos condiciones necesarias para que exista un circuito de Euler:

- El grafo debe ser conexo.
- El grado de todos los nodos –es decir, número de aristas incidentes– debe ser par. La razón es que si el camino pasa varias veces por un nodo, siempre que entra por una arista debe salir por otra.

Un concepto relacionado con el circuito de Euler es el de camino de Euler. Un **camino de Euler** es un camino que visita todas las aristas exactamente una vez, pudiendo empezar y acabar en sitios distintos. Las condiciones necesarias para que exista, en este

caso, serían similares a las del circuito de Euler. La segunda condición permitiría que existan dos nodos con grado impar, o ninguno. Los demás deberían tener grado par.

Se puede comprobar que las anteriores condiciones necesarias son también suficientes. Esto es, si se cumplen entonces existe un circuito de Euler. El objetivo ahora es encontrar un algoritmo para encontrar el circuito o camino de Euler.

Algoritmo para calcular un circuito de Euler

Supongamos un grafo no dirigido G. En primer lugar deberíamos comprobar si se cumplen las condiciones necesarias y suficientes. En ese caso, tenemos garantizado que existe un circuito de Euler, y podemos encontrarlo con el siguiente algoritmo:

1. Buscar un ciclo en G empezando por un vértice v cualquiera. Puede que en este ciclo no todas las aristas hayan sido visitadas. En ese caso, seguimos con el siguiente punto.

2. Si quedan aristas por visitar, seleccionar el primer nodo w del ciclo anterior que tenga una arista no visitada. Buscar un ciclo partiendo de w que pase por aristas no visitadas.

3. Unir el ciclo del paso 1 con el obtenido en el paso 2. Repetir sucesivamente los pasos 2 y 3 hasta que no queden aristas por visitar.

Para encontrar un ciclo, en los pasos 1 y 2, podemos utilizar una búsqueda primero en profundidad, como se comenta en el apartado 5.3.1. No obstante, la diferencia ahora es que lo que se marca como visitado no son los nodos sino las aristas.

Ejemplo 5.10 Vamos a aplicar el algoritmo para calcular el circuito de Euler para el grafo de la figura 5.40.

Supongamos que empezamos por el nodo 1. Podemos tener algo como lo siguiente:

Paso 1. Encontramos el ciclo: $C = (1, 2, 5, 7, 6, 3, 1)$.

Paso 2. No todas las aristas están visitadas. El primer nodo con una arista no visitada es el 2. Encontramos el siguiente ciclo: $D = (2, 3, 4, 2)$.

Paso 3. Unimos los ciclos C y D. Lo que hacemos es, dentro de C en el lugar donde aparece 2 sustituirlo por D. El resultado es: $C = (1, 2, 3, 4, 2, 5, 7, 6, 3, 1)$.

Paso 2. No todas las aristas están visitadas, el primer nodo es 4. Encontramos el ciclo: $D = (4, 5, 6, 4)$.

Paso 3. Unimos el ciclo C con el D, Obtenemos: $C = (1, 2, 3, 4, 5, 6, 4, 2, 5, 7, 6, 3, 1)$. Todas las aristas están ya visitadas, por lo que acaba el algoritmo.

5.8. Otros problemas con grafos

Además de los problemas que hemos estudiado en los apartados anteriores, existen otros muchos tipos de problemas clásicos en teoría de grafos. Todos ellos aparecen como resultado de modelar un problema de la vida real a través de grafos. Así que disponer de una solución eficiente para los mismos resultaría deseable.

Sin embargo, existe una amplia variedad de problemas para los cuales no se conoce hasta la fecha ningún algoritmo capaz de resolverlos de forma eficiente –a pesar de los numerosos años de esfuerzo dedicados y del fenomenal avance que supondría encontrarlos. Entendemos aquí por eficiente un algoritmo con tiempos polinomiales, del tipo $O(n)$, $O(n^2)$, $O(n^{32})$, etc. Esta categoría de problemas para los cuales no se conocen soluciones eficientes –ya sean sobre grafos o no– son conocidos como **problemas NP**. Los algoritmos existentes para resolver problemas de este tipo se basan, en esencia, en comprobar todas las posibles soluciones de manera exhaustiva, dando lugar a tiempos exponenciales o factoriales. El resultado es lo que se conoce como el efecto de **explosión combinatoria**, que hace referencia a la forma en la que se dispara el tiempo de ejecución para tamaños grandes del problema. Alternativamente, se pueden diseñar algoritmos que obtengan soluciones más o menos *buenas*, no necesariamente la óptima, pero en un tiempo reducido. Estos son conocidos como **algoritmos heurísticos**.

Vamos a ver un conjunto de problemas clásicos sobre grafos, que están dentro de la categoría NP. En este apartado únicamente enunciaremos el problema, mostrando las posibles aplicaciones prácticas donde puede ser de utilidad. Por el momento, no daremos algoritmos para resolverlos. Conforme se avance hacia los capítulos de diseño de algoritmos, sería adecuado intentar plantearse la resolución de estos problemas con las técnicas que se vayan estudiando.

5.8.1. Ciclos hamiltonianos

En principio, el problema del ciclo hamiltoniano tiene una formulación muy similar a la del circuito de Euler. Veamos la definición.

Definición 5.18 Dado un grafo no dirigido G, se llama **ciclo de Hamilton** o **ciclo hamiltoniano** a un ciclo simple que visita todos los vértices.

Es decir, el ciclo hamiltoniano pasa por todos los vértices exactamente una vez. El **problema del ciclo hamiltoniano** consiste en: dado un grafo no dirigido G, determinar si posee algún ciclo hamiltoniano. Por ejemplo, en el grafo de la figura 5.41a) existe un ciclo de Hamilton, que ha sido señalado con línea más gruesa. El de la figura 5.41b) no posee ningún ciclo hamiltoniano, lo cual ha sido verificado comprobando exhaustivamente todos los posibles caminos. El de la figura 5.41c) se deja como pasatiempo.

Aunque parece similar al del circuito de Euler, la complejidad implícita del problema del ciclo de Hamilton resulta muchísimo mayor. Mientras que el circuito de Euler se puede resolver aplicando varias búsquedas en profundidad, para el ciclo de Hamilton no se conoce ningún algoritmo capaz de resolverlo en un tiempo polinomial[14]. Se encuentra dentro de los problemas NP.

La solución para este problema consistiría en encontrar todos los posibles caminos simples, comprobando si alguno de ellos es un ciclo hamiltoniano. Si el grafo es completo, el número de caminos simples distintos sería un $(n-1)!$. Para reducir el tiempo, podemos diseñar algún algoritmo heurístico. Por ejemplo, podemos aplicar una búsqueda en profundidad en la cual de todos los adyacentes a un nodo se visitan primero los que tengan

[14]Lo cual, como veremos en el último capítulo, no quiere decir que no exista.

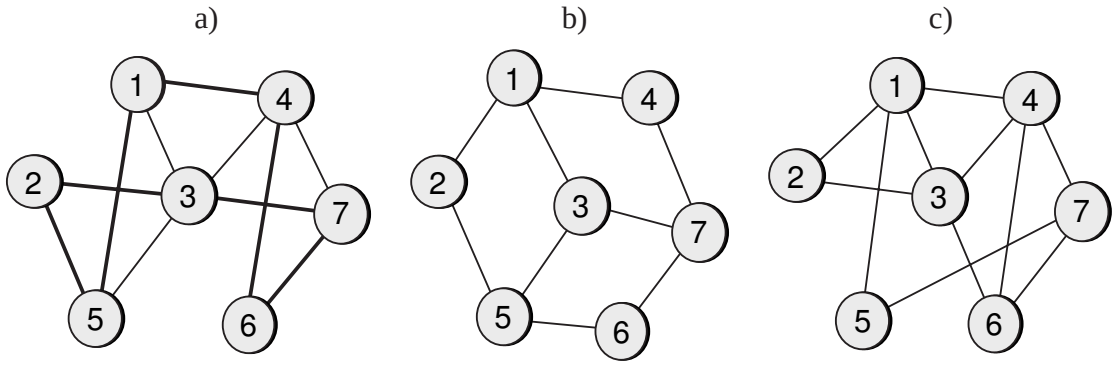

Figura 5.41: Problema del ciclo hamiltoniano. a) Grafo con un ciclo hamiltoniano. b) Grafo que no posee ningún ciclo de Hamilton. c) ¿Existe algún ciclo hamiltoniano?

menor grado. Si lo aplicamos sobre el grafo de la figura 5.41a), encontramos la solución. Empezando por el 1, nos vamos al 5 (grado 3), luego al 2, 3, 7, 6, 4 y cerramos el ciclo. Pero este algoritmo no siempre garantiza la solución. Y si no la encuentra puede que exista o puede que no, como ocurre con el grafo de la figura 5.41c).

5.8.2. Problema del viajante

El **problema del viajante**, también conocido como **problema del agente viajero**, es uno de los más recurridos en las aplicaciones que utilizan grafos para representar caminos con costes asociados[15]. La formulación del problema es la siguiente: dado un grafo no dirigido, completo y con pesos, G, encontrar el ciclo simple de coste mínimo que recorra todos los nodos.

El ciclo al que se refiere el problema sería un ciclo hamiltoniano. Pero la dificultad ahora no está en determinar si existe o no ese ciclo –ya que, al ser completo, trivialmente se sabe que existirá siempre– sino en encontrar el que tenga menor coste de todos ellos. En la figura 5.42a) se muestra un grafo de ejemplo, y en la 5.42b) se muestra una posible solución. ¿Es la solución óptima?

Las aristas del grafo pueden tener coste $+\infty$. Así que decir que el grafo debe ser completo es sólo una forma de interpretar los datos: si el grafo no es completo, las aristas faltantes son consideradas con coste $+\infty$.

Ejemplo 5.11 El problema del viajante está subyacente en aplicaciones de tipo "reparto de mercancías". Por ejemplo, un camionero distribuye pimientos murcianos por toda la región. Tiene que pasar por varios supermercados, n, pero el orden le es indiferente. Eso sí, el camión debe volver al mismo sitio de donde salió. Cada camino necesita un tiempo determinado. El objetivo es planificar la ruta que tiene que seguir, de forma que el tiempo total sea el menor.

La equivalencia con el problema del viajante es inmediata: los n supermercados son los nodos del grafo; los caminos son las aristas, siendo el tiempo el peso de las mismas; y

[15] Además de los problemas de caminos mínimos vistos en el apartado 5.5, claro.

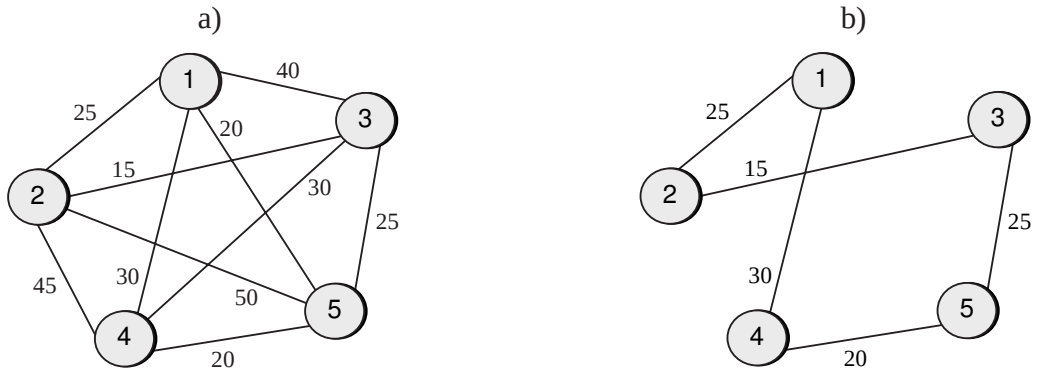

Figura 5.42: Problema del viajante. a) Grafo no dirigido completo y con pesos. b) Posible solución al problema.

el objetivo buscado es el menor ciclo simple que visita todos los nodos.

Teniendo en cuenta que existen $(n-1)!$ posibles ciclos simples, un algoritmo óptimo sencillo estaría en un orden de complejidad factorial. Se han desarrollado numerosas técnicas heurísticas para abordar este problema, como algoritmos voraces, búsqueda local, algoritmos genéticos e incluso computación con ADN. Pero ninguno de ellos garantiza que se obtenga el óptimo, o que la solución obtenida sea próxima a la óptima.

Igual que para el problema del ciclo hamiltoniano, no se conoce ningún algoritmo eficiente que lo resuelva de forma óptima. Es más, se puede demostrar que ambos problemas son equivalentes. La equivalencia se obtiene viendo que es posible transformar un problema en el otro y viceversa. Por ejemplo, para resolver el problema del ciclo hamiltoniano en un grafo no dirigido $G = (V, A)$, podemos considerar el grafo $G' = (V, A', W)$, donde A' contiene todas las aristas posibles, y $W(v, w) = 1$ si la arista $(v, w) \in A$ y $W(v, w) = +\infty$ en caso contrario. Si resolvemos el problema del viajante en G', entonces existirá un ciclo hamiltoniano siempre que el coste sea menor que $+\infty$.

5.8.3. Coloración de grafos

En los problemas de coloración de grafos, las aristas no representan *caminos* sino *incompatibilidades*. Los nodos representan cierto tipo de objeto y existe una arista (v, w) entre dos nodos si los objetos v y w son incompatibles. La coloración de un grafo consiste en asignar un **color** o **etiqueta** a cada nodo, de forma que dos nodos incompatibles no tengan el mismo color. Formalmente se puede definir de la siguiente manera.

Definición 5.19 Dado un grafo $G = (V, A)$ no dirigido, una **coloración del grafo** es una función $C : V \to N$, tal que si $(v, w) \in A$ entonces $C(v) \neq C(w)$.

El **problema de la coloración** de grafos consiste en: dado un grafo no dirigido, encontrar una coloración del mismo utilizando el mínimo número de colores distintos. En la figura 5.43 se muestra un ejemplo del problema. Las figuras 5.43b) y 5.43c) son dos

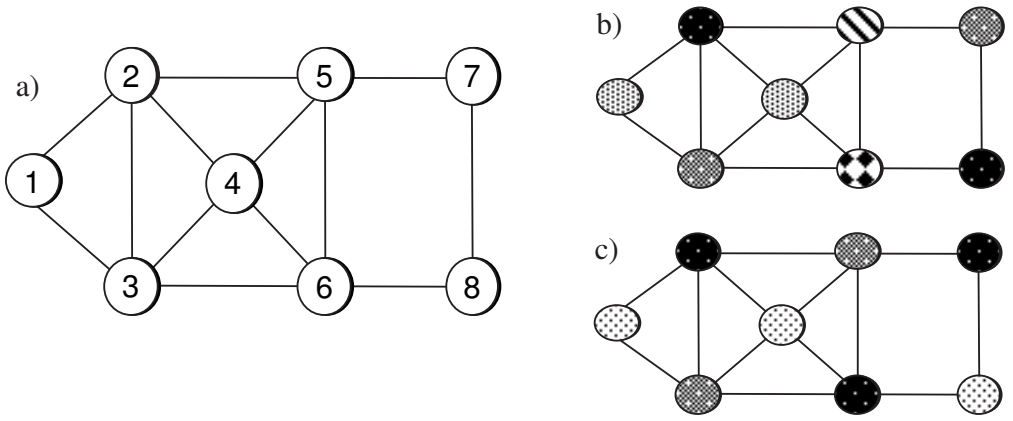

Figura 5.43: Problema de coloración de grafos. a) Grafo no dirigido del problema. b) Coloración usando 5 colores. c) Coloración usando 3 colores.

posibles coloraciones del grafo. La coloración de la figura 5.43c) utiliza menos colores, y se puede comprobar que es la solución óptima de este ejemplo.

El problema de coloración de grafos es también un problema NP. La solución consistiría en comprobar todas las posibles coloraciones y quedarnos con la que use menos colores.

Ejemplo 5.12 En un mapa geopolítico aparecen dibujadas distintas regiones, cada una de las cuales es fronteriza con otras regiones. Queremos rellenar de color las regiones del mapa, de forma que dos regiones fronterizas no tengan el mismo color. Para ahorrar costes de producción, la editorial pide que se use el mínimo número de colores distintos. ¿Cuántos colores diferentes necesitamos, como mínimo, para colorear el mapa?

El problema puede ser modelado usando grafos. Cada región del mapa se corresponderá con un nodo del grafo. Habrá una arista entre dos nodos si las regiones asociadas son fronterizas. En la figura 5.44 aparece un ejemplo de transformación de un problema de mapas a un problema de grafos.

Está claro que el resultado del problema será la coloración mínima del grafo. Los grafos que surgen en este tipo de aplicación son llamados **grafos planos**. Un grafo se dice que es plano si se puede dibujar en papel sin que se crucen las aristas. Está demostrado que cualquier grafo plano puede dibujarse usando como máximo cuatro colores. Pero para los grafos no planos el número de colores depende del caso.

5.8.4. Isomorfismo de grafos

El isomorfismo de grafos es una *generalización* de la igualdad entre dos grafos. Por definición, dados dos grafos G y F, se dice que son iguales cuando $V(G) = V(F)$ y $A(G) = A(F)$. Pero lo que normalmente interesa no es la comparación de grafos en igualdad, sino conocer si los grafos tienen una estructura *equivalente*. Esto es lo que llamamos isomorfismo.

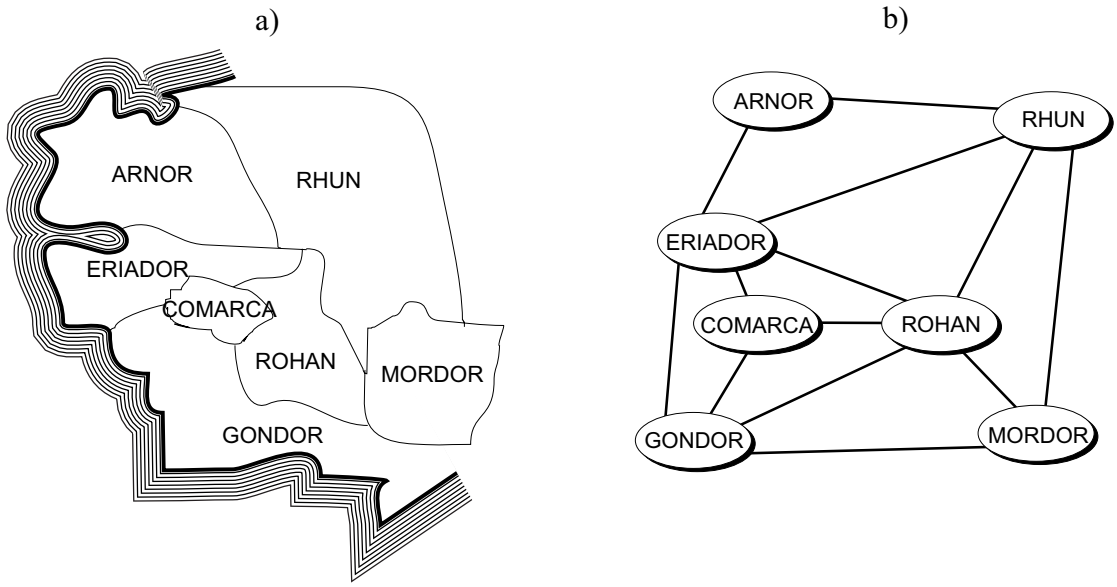

Figura 5.44: Problema de coloración de un mapa. a) Mapa de la Tierra Media. b) Transformación a un problema de grafos.

Definición 5.20 Dados dos grafos G y F, un **isomorfismo entre grafos** es una asignación de los vértices de $V(G)$ con los vértices de $V(F)$ tal que se respetan las aristas. Es decir, es una función biyectiva: $f : V(G) \rightarrow V(F)$, tal que para todo $v, w \in V(G)$, $(v, w) \in A(G) \Leftrightarrow (f(v), f(w)) \in A(F)$.

Dos grafos se dice que son **isomorfos** si existe un isomorfismo entre ellos. Por ejemplo, entre los grafos de la figura 5.45 existen varios posibles isomorfismos. Uno de ello podría ser la asignación: $1 \rightarrow b;\ 2 \rightarrow a;\ 3 \rightarrow d;\ 4 \rightarrow e;\ 5 \rightarrow c;\ 6 \rightarrow g;\ 7 \rightarrow f$.

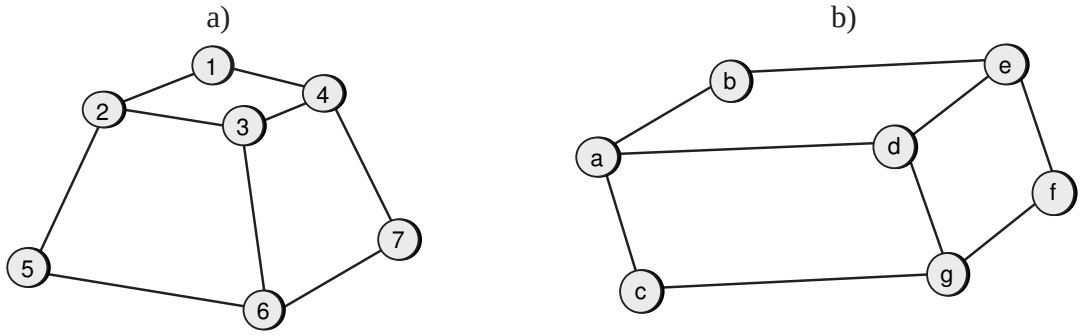

Figura 5.45: Problema del isomorfismo de grafos. a),b) Dos grafos no dirigidos isomorfos.

Nuevamente, el isomorfismo de grafos es un problema NP. La solución consistiría en ir comprobando todas las posibles asignaciones, hasta encontrar alguna válida. Esto requeriría un orden de tipo factorial. Si el grafo es etiquetado, entonces la asignación

debería respetar también los pesos o etiquetas de las aristas. En ciertas aplicaciones, puede surgir el problema del **subisomorfismo** de grafos. Intuitivamente, el subisomorfismo es la mayor asignación posible entre los nodos de los dos grafos, que *respete* las aristas. Vamos a ver un ejemplo.

Ejemplo 5.13 Una aplicación de visión artificial analiza una imagen y debe interpretar qué es lo que está viendo. Su mundo se reduce a un conjunto de p poliedros, de cada uno de los cuales posee un modelo. El problema es: dada una imagen decir dónde se encuentran los objetos.

Para resolverlo creamos un grafo a partir de la imagen, donde los vértices son puntos del dibujo y las aristas del grafo son líneas del dibujo. En la figura 5.46 se muestra un ejemplo con dos modelos de poliedros (figuras 5.46a) y b) y una escena para localizar los modelos (figura 5.46c). ¿Cómo resolver el problema algorítmicamente?

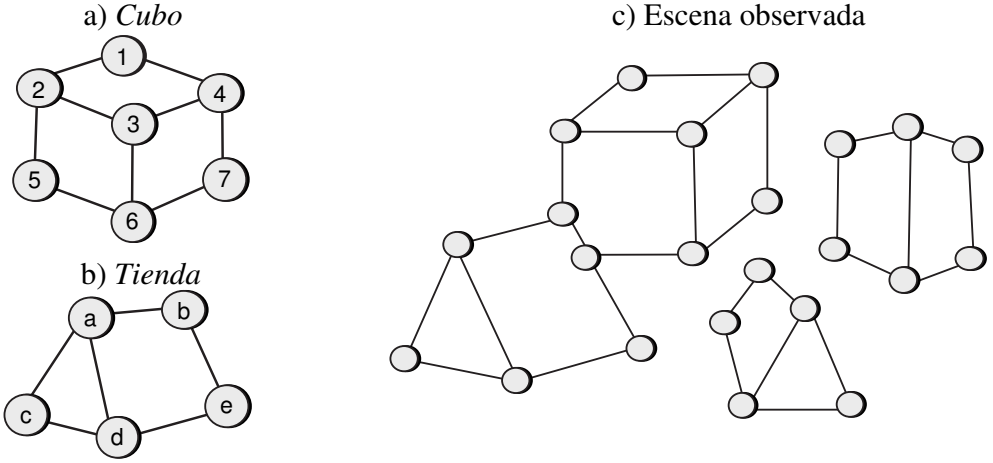

Figura 5.46: Aplicación del isomorfismo de grafos. a),b) Modelos de grafo asociados a dos poliedros. c) Grafo extraído de un dibujo de líneas.

En primer lugar, podríamos separar el grafo del dibujo en sus componentes conexos. Idealmente, cada componente debería ser isomorfo a algún grafo del modelo. Pero, puesto que puede haber ruido, imperfecciones o solapamientos, lo que tenemos realmente es un problema de subisomorfismo. Por ejemplo, de los tres componente de la figura 5.46c), en uno de ellos podemos encontrar un isomorfismo con la figura denominada *Tienda*, en otro encontramos dos subisomorfismos con *Tienda* y con *Cubo*, y en el tercero no hay ningún isomorfismo adecuado.

Ejercicios resueltos

Ejercicio 5.1 El grafo de la figura 5.47 representa una red de ordenadores, con enlaces entre los mismos. Todos ellos se pueden comunicar entre sí, directamente o a través de otros. Encontrar los ordenadores críticos, es decir los que no pueden fallar para que todos