Recorridos y conectividad

Joaquim Borges
Robert Clarisó
Ramon Masià
Jaume Pujol
Josep Rifà
Joan Vancells
Mercè Villanueva

PID_00174687

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Si una palabra clave tuviera que describir el módulo, ésta sería la de *recorridos* en un grafo, es decir, maneras de "recorrer" un grafo, visitando vértices a través de aristas. Así pues, se analizan los diferentes tipos de recorridos que puede haber, la accesibilidad entre vértices y las distancias. Además, se presentan las propiedades estructurales que garantizan que ciertos tipos de recorridos siempre se podrán realizar y que la accesibilidad se conservará aunque algunas partes del grafo (vértices o aristas) desaparezcan.

La teoría y los resultados que se presentan aquí constituyen el marco teórico de importantes aplicaciones y modelados para redes de comunicaciones y problemas de optimización combinatoria.

1. Recorridos

Después de caracterizar los grafos y presentar algunos de los más importantes, conviene estudiar las diversas maneras de recorrerlos. Para ello, se introduce el concepto de recorrido. En primer lugar, será imprescindible enunciar algunos resultados fundamentales sobre recorridos.

Determinados grafos están formados por "una sola pieza", es decir, hay algún recorrido entre cualquier par de vértices del grafo. En cambio, hay otros que están formados por "más de una pieza". Por sus aplicaciones a varios campos, es importante la noción de distancia entre vértices, que se puede definir mediante la conexión entre éstos.

1.1. Concepto de recorrido

Definición 1

Un **recorrido** en un grafo simple $G = (V,A)$ es una secuencia de vértices $v_1,v_2,\ldots,v_k$ con la propiedad de que dos vértices consecutivos en la secuencia deben ser los extremos de una arista de G, es decir, $\{v_i,v_{i+1}\} \in A$. Este recorrido de extremos v_1, v_k se denomina v_1–v_k **recorrido** o, también, recorrido entre v_1 y v_k.

Si $G = (V,A)$ es un grafo dirigido, un **recorrido orientado** o **dirigido** es una secuencia de vértices $w_1,w_2,\ldots,w_k$ con la propiedad de que dos vértices consecutivos en la secuencia deben ser los extremos de un arco de G, es decir, $(w_i,w_{i+1}) \in A$.

En la secuencia de vértices no se utiliza la numeración de los vértices, sino la numeración correspondiente a su número de orden en la secuencia.

En un multigrafo o un pseudografo es posible pasar de un vértice a otro que le sea adyacente por más de una arista. Por ello, es necesario indicar un recorrido como una secuencia de aristas, $a_1,a_2,\ldots,$

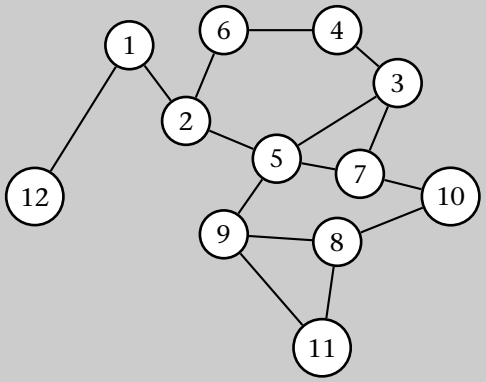
a_{k-1},a_k, de manera que dos aristas consecutivas en la secuencia tienen que compartir un extremo.

Definición 2

- Un $u - v$ recorrido es **cerrado** si los extremos coinciden, es decir, si $u = v$; en caso contrario se dice que es **abierto**.
- La **longitud de un recorrido** R, $\ell(R)$, es el número de aristas que lo componen. Se cuentan también las que pueda tener repetidas.
- Un **recorrido trivial** o de longitud 0 es el formado por un único vértice.

Ejemplo 1

Se considera el grafo representado en la figura siguiente.



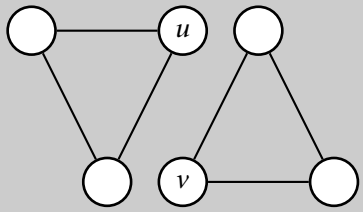
Un recorrido posible sobre este grafo sería el que corresponde a la siguiente secuencia de vértices: 12,1,2,6,4,3,7,10,8,11; éste sería un 12-11 recorrido de longitud 9.

Un recorrido cerrado sería: 6,4,3,5,2,6, de longitud 5. Observad que puede haber (dependiendo del grafo) más de un recorrido entre dos vértices: por ejemplo, dos 12-11 recorridos podrían ser: 12,1,2,5,7,10,8,9,11 y también 12,1,2,5,9,11.

Aunque dos recorridos tengan globalmente las mismas aristas, pueden ser diferentes si las aristas se utilizan en órdenes diferentes. Por ejemplo, son recorridos diferentes los siguientes: 9,5,3,7,5,2 y 9,5,7,3,5,2.

Ejemplo 2

En el grafo representado en la figura siguiente no existe ningún recorrido posible entre dos vértices, en este caso u y v; los vértices en cuestión no son mutuamente accesibles en el grafo $G = C_3 \cup C_3$.



Definición 3

Un recorrido es un **itinerario** si todas las aristas son diferentes. Se pueden destacar estos tipos de itinerario:

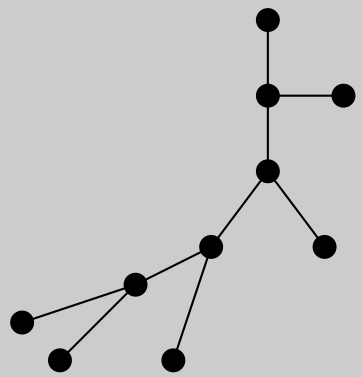
- Un **camino**, si no se repiten vértices.
- Un **circuito**, si es cerrado.
- Un **ciclo** es un circuito (cerrado) que, eliminando el primer vértice, también es un camino (no repite vértices). Los grafos que no contienen ciclos se denominan **acíclicos**.

Ejemplo 3

En el grafo del ejemplo 1 se tiene:

Ciclo: 9,8,11,9
Ciclo: 3,5,9,11,8,10,7,3
Circuito: 5,2,6,4,3,5,9,8,10,7,5
No itinerario: 12,1,2,5,3,7,5,2,1,12 (repite las aristas {1,2} y {2,5})
No camino: 1,2,5,7,3,5,9,8,11 (repite el vértice 5)

Ejemplo 4



Ejemplo 5

Un árbol, T, es un grafo que cumple que entre dos vértices cualesquiera del grafo hay un único camino que los conecta. Por ejemplo, el grafo del ejemplo 4 es un árbol. Es evidente que un árbol no puede contener ciclos.

Ejemplo de árbol

El árbol genealógico de una familia es un ejemplo de árbol, siempre y cuando dos descendientes del antepasado común no tengan nunca descendencia común.

Ved también

Los árboles son grafos muy importantes y, por ello, se estudiarán más adelante en el módulo "Árboles".

Ejercicios

1. ¿Cuál es la longitud mínima de un ciclo?

2. Un ciclo de longitud n de un grafo de orden n, ¿es un subgrafo de este grafo?

3. Indicad varios ciclos en el grafo completo K_6.

4. ¿Cuál es la longitud del grafo ciclo C_n?

5. Indicad si son acíclicos los grafos T_n, C_n, K_n, $K_{n,m}$, R_n, E_n.

6. La propiedad de aciclicidad se hereda por parte de los subgrafos de un grafo. ¿Cierto o falso?

7. ¿La eliminación de aristas o de vértices de un grafo acíclico produce un nuevo grafo acíclico?

8. Si en un recorrido no hay repetición de aristas, ¿puede haber repetición de vértices?

Soluciones

1. La longitud mínima es 3.

2. Cierto (acabad de justificar la respuesta).

3. Tiene muchos ciclos: sólo hace falta tomar tres vértices cualesquiera y unirlos dos a dos con aristas diferentes.

4. Para el grafo cíclico C_n, $\ell(C_n) = n$.

5. Los acíclicos son T_n, E_n, K_n ($n \leq 2$) y $K_{n,m}$ ($n = 1$ o $m = 1$).

6. Cierto (acabad de justificar la respuesta).

7. Sí (acabad de justificar la respuesta).

8. Sí (acabad de justificar la respuesta).

1.2. Resultados fundamentales

Proposición 1

Dados dos vértices diferentes u, v de un grafo $G = (V,A)$, entonces todo $u-v$ recorrido contiene un $u-v$ camino, es decir, un recorrido entre los vértices sin repetición de vértices.

Demostración: Sea R un $u-v$ recorrido del grafo G. Ya que u y v son diferentes se puede suponer que R es abierto, y sea por ejemplo $R: u = w_0 w_1, \ldots, w_k = v$, pudiendo contener vértices repetidos. Si no hay ninguno repetido, se ha llegado al final, puesto que R cumple la condición requerida. Se supone, por lo tanto, que hay algún vértice repetido, y sean $i < j$ tales que $w_i = w_j$; entonces se suprimen los vértices $w_i w_{i+1}, \ldots, w_{j-1}$, y resulta un nuevo $u-v$ recorrido R_1 de longitud estrictamente inferior con una repetición de $w_i = w_j$ eliminada (aun cuando todavía pueden quedar más repeticiones del mismo vértice). Si R_1 es un camino, se ha acabado; de lo contrario, el proceso continúa y necesariamente se llega a la conclusión indicada, puesto que la longitud inicial es finita. ∎

Ejemplo 6

En el grafo del ejemplo 1, a partir del recorrido 2,5,3,7,5,9,8,11 se puede extraer un camino: 2,5,9,8,11. Para hacerlo, únicamente es necesario descartar la parte del recorrido que hay entre las dos visitas al vértice 5.

Proposición 2

Si en un grafo $G = (V,A)$ hay dos caminos diferentes que conectan un par de vértices, entonces el grafo contiene algún ciclo.

Demostración: Se consideran dos $u-v$ caminos diferentes: $R_1: u = w_1 w_2 \ldots w_p = v$ y $R_2: u = t_1 t_2 \ldots t_q = v$. Informalmente, los caminos pueden ir compartiendo vértices hasta llegar al primer vértice a partir del cual se separan, es decir, empiezan a ser diferentes. Sea, por lo tanto, i el primer índice para el cual $w_i \neq t_i$. Tarde o temprano los caminos tienen que confluir, finalmente, en el vértice terminal común, que es v, y esto lo harán en el mismo vértice o antes; pero, en todo caso, a partir de un cierto lugar en lo sucesivo, los dos caminos vuelven a compartir vértices (y aristas, puesto que el grafo es simple). Ahora bien, en un cierto vértice posterior a la bifurcación anterior tienen que volver a coincidir por primera vez. Sean, pues, j, k mínimos tales que $i < j \leq p$, $i < k \leq q$, $w_j = t_k$. Entonces el camino $w_{i-1} w_i w_{i+1} \ldots w_j t_{k-1} t_{k-2} \ldots t_i w_{i-1}$ es un ciclo. ∎

Teorema 3

Si todos los vértices de un grafo $G = (V,A)$ son de grado superior o igual a 2, entonces el grafo contiene algún ciclo.

Demostración: Sea n el orden del grafo. Se considera un vértice cualquiera, que se indica por w_1, de grado como mínimo 2 por hipótesis; hay un vértice w_2 adyacente a w_1. Dado que $g(w_2) \geq 2$, hay un vértice w_3 adyacente a w_2 y diferente de w_1. De este modo se podrá construir una secuencia de vértices encadenados. Sea ahora w_i diferente de todos los anteriores y adyacente a w_{i-1}. Si w_i es adyacente a alguno de los anteriores (que no sea w_{i-1}), se tiene un ciclo y se ha acabado el problema. Si no es así, dado que es de grado superior o igual a 2, hay un nuevo vértice w_{i+1} diferente de todos los anteriores y adyacente a w_i. Por este proceso, necesariamente finito, o bien se produce en algún momento un ciclo y se ha acabado, o bien se van incorporando todos los vértices en un camino que los contiene todos, $w_1,w_2,\ldots w_n$, y se mantienen las propiedades anteriores. Ahora bien, dado que $g(w_1) \geq 2,\ldots,g(w_n) \geq 2$, tienen que ser necesariamente adyacentes y, por lo tanto, se forma un ciclo. ∎

Ejercicios

9. ¿Es cierto que un grafo acíclico sin vértices aislados contiene vértices de grado 1?

10. Si un grafo es acíclico, entonces entre cada pareja de vértices diferentes hay como máximo un camino. ¿Cierto o falso?

11. Demostrad que todo grafo r-regular ($r \geq 2$) contiene un ciclo.

12. Poned un ejemplo de grafo acíclico tal que su complementario también sea acíclico.

13. Demostrad que si en un grafo de orden n el grado máximo de los vértices es menor que $n-2$, entonces el grafo complementario contiene un ciclo.

Soluciones

9. Cierto. Si no tuviera ningún vértice de grado 1 entonces todos los vértices tendrían grado mayor o igual a 2 y tendría que contener un ciclo, por el teorema 3.

10. Cierto. Si hubiera más de un camino, entonces contendría algún ciclo, por la proposición 2.

11. Si el grafo es r-regular ($r \geq 2$), entonces todos los vértices son de grado superior o igual a 2 y el grafo contendrá un ciclo, por el teorema 3.

12. Por ejemplo, el grafo T_3.

13. Si G^c es el grafo complementario, entonces $g_{G^c}(v) = n - 1 - g_G(v) \geq 2$ para todo $v \in V$ y, por lo tanto, contiene un ciclo, por el teorema 3.

2. Algoritmos de exploración de grafos

Para determinar las propiedades de un grafo, a menudo es necesario explorar cada uno de los vértices y de las aristas del grafo en un orden determinado. Desde un punto de vista algorítmico hay básicamente dos procedimientos para hacerlo: el **algoritmo de búsqueda primeramente en profundidad** (**DFS** o *depth first search*) y el **algoritmo de búsqueda primeramente en anchura** (**BFS** o *breadth first search*).

El algoritmo de búsqueda primeramente en profundidad se aplica a varios problemas de conectividad (buscar componentes conexas o biconexas) y, también, puede comprobar si un grafo es acíclico. Otros algoritmos, como el de Dijkstra y el de Prim, utilizan ideas similares al DFS.

El algoritmo de búsqueda primeramente en anchura se aplica a varios problemas de coloración (coloración de grafos bipartitos), a la búsqueda del ciclo más corto de un grafo o al cálculo de distancias.

2.1. Algoritmo DFS

Se inicia la exploración del grafo en un vértice prefijado y a partir de éste se exploran todos a través de las aristas, sin repetición, de una manera determinada. Esencialmente, se trata de avanzar lo más profundamente posible, sin explorar todas las posibilidades que se dan a partir del vértice actual. De este modo, del vértice actual se pasa a un vértice adyacente no visitado, de éste a un tercero aún no visitado, y así sucesivamente. Si se llega a un vértice desde el cual no se puede seguir, es necesario volver atrás (retroceso o *backtracking*) y eliminarlo. Este proceso se repite hasta volver al punto inicial, donde se habrá acabado la exploración.

Para resolver problemas es útil conocer la formulación del algoritmo básico, que explora todos los vértices sin repetición. Claro está que se puede completar con acciones adicionales y formular nuevas variantes.

2.1.1. Formulación del algoritmo DFS

Estructuras necesarias para la formulación del algoritmo:

- Un grafo $G = (V,A)$ representado con una lista de adyacencias.

- Un conjunto P de los vértices que se han visitado, en el orden en que se ha hecho, que permita dar marcha atrás. La estructura de datos adecuada es la de una pila con las operaciones habituales: *apilar*(P,v), *desapilar*(P), *cima*(P).
- Una tabla de vértices (*estado*) que registra los vértices que se van visitando.
- Una lista R que contiene los vértices visitados hasta el momento (y que finalmente coincidirá con el conjunto de todos los vértices).

Estructura de pila

En la estructura de pila los elementos se desapilan en orden inverso al que se apilan, y la cima es el último elemento apilado.

Cuando es posible visitar más de un vértice, se elige siempre el de índice menor en la ordenación de los vértices disponibles, según la ordenación general de los vértices del grafo (por ejemplo, si los vértices se representan por letras, entonces se hace la elección del primero disponible por orden alfabético).

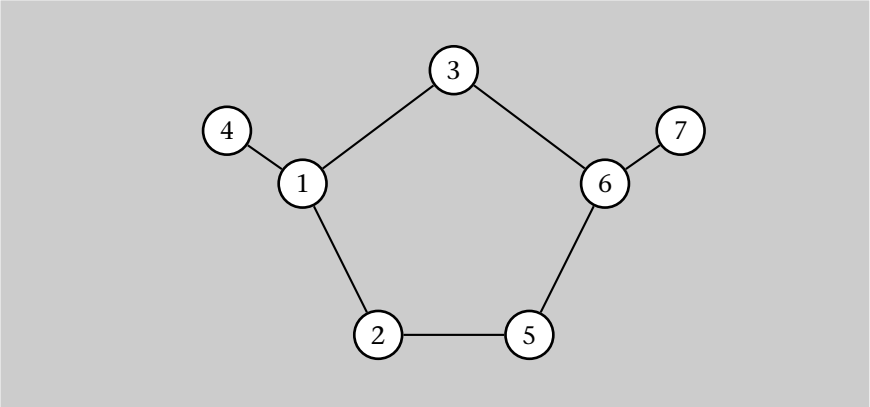
La pila P se inicializa con el vértice de partida y se van apilando los vértices que son visitados. Cuando se llega a un vértice desde el que no se puede continuar, se van desapilando vértices hasta encontrar uno a partir del cual se puede volver a avanzar, con lo que se reinicia, así, el proceso. Si la pila queda vacía, se para el proceso.

```
Entrada : G = (V,A), v ∈ V
Salida : R, vértices visitados
algoritmo DFS(G,v)
  inicio
    P ← ∅
    R ← [v]
    para w ∈ V
          estado[w] ← 0
    finpara
    estado[v] ← 1
    apilar(P,v)
    mientras P ≠ ∅
                w ← cima(P)
                si w es adyacente a u con estado[u] = 0
                   entonces apilar(P,u)
                            estado[u] ← 1
                            añadir(R,u)
                        sino desapilar(P)
                finsi
    finmientras
    retorno (R)
  fin
```

2.1.2. Simulación del algoritmo

Ejemplo 7

Se considera el grafo representado en la figura siguiente.



Esta tabla registra el funcionamiento del algoritmo de exploración en profundidad (DFS) para este grafo, con vértice de inicio $v = 1$.

P	Vértice añadido	Vértice eliminado	R
1	1	-	[1]
12	2	-	[1,2]
125	5	-	[1,2,5]
1256	6	-	[1,2,5,6]
12563	3	-	[1,2,5,6,3]
1256	-	3	[1,2,5,6,3]
12567	7	-	[1,2,5,6,3,7]
1256	-	7	[1,2,5,6,3,7]
125	-	6	[1,2,5,6,3,7]
12	-	5	[1,2,5,6,3,7]
1	-	2	[1,2,5,6,3,7]
14	4	-	[1,2,5,6,3,7,4]
1	-	4	[1,2,5,6,3,7,4]
$\emptyset$	-	1	[1,2,5,6,3,7,4]

2.1.3. Análisis del algoritmo DFS

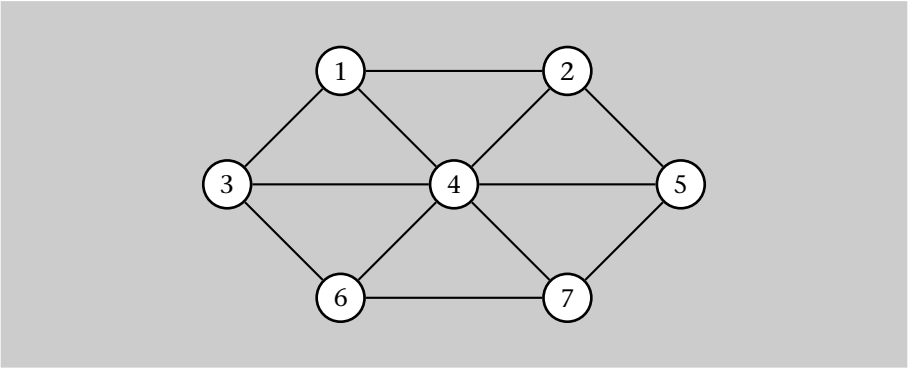
Si el grafo G tiene orden n y medida m, se puede calcular, de manera aproximada, la complejidad de este algoritmo observando que cada vértice es visitado dos veces (una cuando se añade a la pila y otra cuando se elimina). Además, cuando se accede a un vértice de la pila, se analizan todos sus adyacentes para buscar los vértices aún no visitados. En total, se repetirá tantas veces como aristas haya en la lista de adyacencias del grafo.

Así el número total de operaciones será proporcional a $2n + 2m$ y el algoritmo tendrá una complejidad $O(n + m)$.

Ejercicios

14. Encontrad la lista de los vértices visitados por el algoritmos DFS en el grafo del ejemplo 7 empezando por el vértice 6. Construid, también, la tabla que registra el funcionamiento del algoritmo.

15. Encontrad la lista de los vértices visitados por el algoritmo DFS en el grafo siguiente empezando por el vértice 1. Construid, también, la tabla que registra el funcionamiento del algoritmo.



16. Al mismo tiempo que el algoritmo DFS visita todos los vértices de un grafo, también visita las aristas incidentes en estos vértices. Modificad el algoritmo para que devuelva la lista de aristas visitadas por el DFS. En lugar de la lista R será una lista S de aristas visitadas.

17. Utilizad el algoritmo del ejercicio anterior para obtener las aristas visitadas en el grafo del ejemplo 7. Construid también la tabla de funcionamiento del algoritmo con el encabezamiento siguiente:

P	Arista añadida	S

Soluciones

14. La lista de vértices visitados es R = [6,3,1,2,5,4,7].

15. La lista de vértices visitados es R = [1,2,4,3,6,7,5].

16. La versión del DFS para las aristas será:

```
Entrada : G = (V,A), v ∈ V
Salida : S, aristas visitadas
algoritmo DFS(G,v)
  inicio
    P ← ∅
    S ← ∅
    para w ∈ V
          estado[w] ← 0
    finpara
    estado[v] ← 1
    apilar(P,v)
    mientras P ≠ ∅
                w ← cima(P)
                si w es adyacente a u con estado[u] = 0
                   entonces apilar(P,u)
                             estado[u] ← 1
                             añadir(S,{w,u})
                        sino desapilar(P)
                finsi
    finmientras
    retorno (S)
  fin
```

17. La tabla de funcionamiento del algoritmo será:

P	Arista añadida	S
1	-	$\emptyset$
12	{1,2}	[{1,2}]
125	{2,5}	[{1,2},{2,5}]
1256	{5,6}	[{1,2},{2,5},{5,6}]
12563	{6,3}	[{1,2},{2,5},{5,6},{6,3}]
1256	-	[{1,2},{2,5},{5,6},{6,3}]
12567	{6,7}	[{1,2},{2,5},{5,6},{6,3},{6,7}]
1256	-	[{1,2},{2,5},{5,6},{6,3},{6,7}]
125	-	[{1,2},{2,5},{5,6},{6,3},{6,7}]
12	-	[{1,2},{2,5},{5,6},{6,3},{6,7}]
1	-	[{1,2},{2,5},{5,6},{6,3},{6,7}]
14	{1,4}	[{1,2},{2,5},{5,6},{6,3},{6,7},{1,4}]
1	-	[{1,2},{2,5},{5,6},{6,3},{6,7},{1,4}]
$\emptyset$	-	[{1,2},{2,5},{5,6},{6,3},{6,7},{1,4}]

Observación

El grafo (V,S) siempre es un árbol.

2.2. Algoritmo BFS

El algoritmo es similar al DFS, con la diferencia importante de que cuando se llega al vértice actual que se visita, primero se hace una visita sistemática a todos sus vértices adyacentes que todavía no han sido visitados, escogidos por orden según la ordenación que haya.

Es útil conocer la formulación del algoritmo básico, que se puede completar con acciones adicionales para formular nuevas variantes.

2.2.1. Formulación del algoritmo BFS

Estructuras necesarias para la formulación del algoritmo:

- Un grafo $G = (V,A)$ representado mediante una lista de adyacencias.
- Un conjunto Q de los vértices que se han visitado, en el orden en el que se ha hecho. La estructura de datos adecuada es la de una cola con las operaciones habituales: *añadir*(Q,v), *eliminar*(Q), *primero*(Q).
- Una tabla de vértices (*estado*) que registra los vértices que se van visitando.
- Una lista R que contiene los vértices visitados hasta el momento (y que finalmente coincidirá con el conjunto de todos los vértices).

Estructura de cola

En la estructura de cola los elementos se eliminan en el mismo orden en el que se añaden.

Cuando es posible visitar más de un vértice, siempre se escoge el de índice mínimo en la ordenación de los vértices disponibles, según la ordenación general de los vértices del grafo (si los vértices se representan por letras, entonces se hace la elección del primero disponible por orden alfabético).

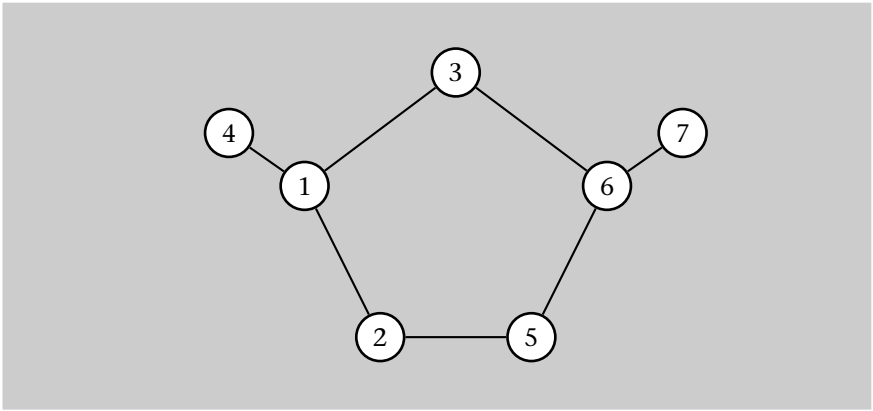
La cola Q se inicia con el vértice de partida y los vértices que son visitados se van añadiendo a la cola. Cuando se han visitado todos los vértices adyacentes al de partida, se elimina de la cola y se continúa con el vértice siguiente de ésta. Cuando la cola queda vacía, se para el proceso.

```
Entrada : G = (V,A), v ∈ V
Salida : R, vértices visitados
algoritmo BFS(G,v)
  inicio
    Q ← ∅
    R ← [v]
    para w ∈ V
          estado[w] ← 0
    finpara
    estado[v] ← 1
    añadir(Q,v)
    mientras Q ≠ ∅
                w ← primero(Q)
                para u adyacente a w
                      si estado[u] = 0
                         entonces añadir(Q,u)
                                  estado[u] ← 1
                                  añadir(R,u)
                      finsi
                finpara
                eliminar(Q)
    finmientras
    retorno (R)
  fin
```

2.2.2. Simulación del algoritmo

Ejemplo 8

Se considera el grafo representado en la figura siguiente.



Esta tabla registra el funcionamiento del algoritmo de exploración en anchura (BFS) para este grafo, con vértice de inicio $v = 1$.

Q	Vértice añadido	Vértice eliminado	R
1	1	-	[1]
12	2	-	[1,2]
123	3	-	[1,2,3]
1234	4	-	[1,2,3,4]
234	-	1	[1,2,3,4]
2345	5	-	[1,2,3,4,5]
345	-	2	[1,2,3,4,5]
3456	6	-	[1,2,3,4,5,6]
456	-	3	[1,2,3,4,5,6]
56	-	4	[1,2,3,4,5,6]
6	-	5	[1,2,3,4,5,6]
67	7	-	[1,2,3,4,5,6,7]
7	-	6	[1,2,3,4,5,6,7]
$\emptyset$	-	7	[1,2,3,4,5,6,7]

2.2.3. Análisis del algoritmo BFS

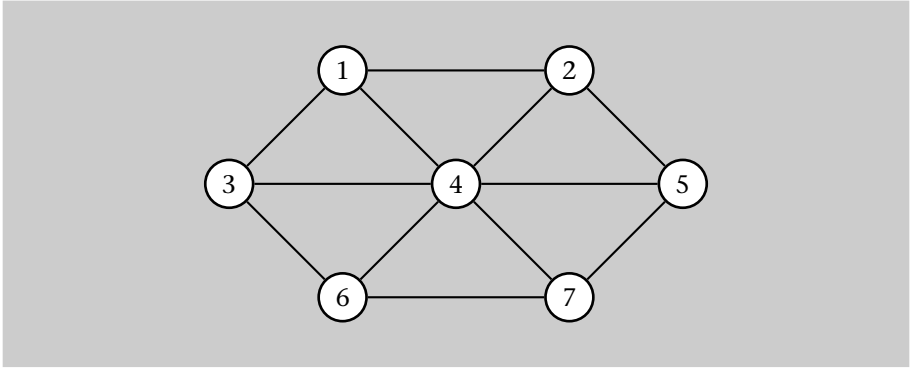
Observad que si el grafo G tiene orden n, entonces cada vértice se añade a la cola una vez. Además, cuando se accede a un vértice de la cola, se analiza su lista de adyacencias para buscar los vértices aún no visitados. En total, se habrán analizado $2m$ aristas, siendo m la medida del grafo.

Así, el número total de operaciones será proporcional a $n + 2m$ y el algoritmo tendrá una complejidad $O(n + m)$.

Ejercicios

18. Encontrad la lista de los vértices visitados por el algoritmo BFS en el grafo del ejemplo 8 empezando por el vértice 6. Construid, también, la tabla que registra el funcionamiento del algoritmo.

19. Encontrad la lista de los vértices visitados por el algoritmo BFS en el grafo siguiente empezando por el vértice 1. Construid, también, la tabla que registra el funcionamiento del algoritmo.



20. Como en el caso del DFS, al mismo tiempo que el algoritmo BFS visita todos los vértices de un grafo, también visita las aristas incidentes en estos

vértices. Modificad el algoritmo para que devuelva la lista de aristas visitadas por el BFS. En lugar de la lista R será una lista S de aristas visitadas.

21. Usad el algoritmo del ejercicio anterior para obtener las aristas visitadas en el grafo del ejemplo 8. Construid también la tabla de funcionamiento del algoritmo con la cabecera siguiente:

Q	Arista añadida	S

Soluciones

18. La lista de vértices visitados es R = [6,3,5,7,1,2,4].

19. La lista de vértices visitados es R = [1,2,3,4,5,6,7].

20. La versión del BFS para las aristas será:

```
Entrada : G = (V,A), v ∈ V
Salida : S, aristas visitadas
algoritmo BFS(G,v)
  inicio
    Q ← ∅
    S ← ∅
    para w ∈ V
          estado[w] ← 0
    finpara
    estado[v] ← 1
    añadir(Q,v)
    mientras Q ≠ ∅
                w ← primero(Q)
                para u adyacente a w
                      si estado[u] = 0
                         entonces añadir(Q,u)
                                  estado[u] ← 1
                                  añadir(S,{w,u})
                      finsi
                finpara
                eliminar(Q)
    finmientras
    retorno (S)
  fin
```

21. La tabla de funcionamiento del algoritmo será:

Q	Arista añadida	S
1	-	$\emptyset$
12	{1,2}	[{1,2}]
123	{1,3}	[{1,2},{1,3}]
1234	{1,4}	[{1,2},{1,3},{1,4}]
234	-	[{1,2},{1,3},{1,4}]
2345	{2,5}	[{1,2},{1,3},{1,4},{2,5}]
345	-	[{1,2},{1,3},{1,4},{2,5}]
3456	{3,6}	[{1,2},{1,3},{1,4},{2,5},{3,6}]
456	-	[{1,2},{1,3},{1,4},{2,5},{3,6}]
56	-	[{1,2},{1,3},{1,4},{2,5},{3,6}]
6	-	[{1,2},{1,3},{1,4},{2,5},{3,6}]
67	{6,7}	[{1,2},{1,3},{1,4},{2,5},{3,6},{6,7}]
7	-	[{1,2},{1,3},{1,4},{2,5},{3,6},{6,7}]
$\emptyset$	-	[{1,2},{1,3},{1,4},{2,5},{3,6},{6,7}]

Observación

El grafo (V,S) resultante es un árbol.

3. Conectividad

Uno de los conceptos fundamentales de la teoría de grafos es el de conectividad, definido a partir de la posibilidad de establecer un recorrido entre dos vértices cualesquiera de un grafo. El concepto de conectividad es especialmente importante en aquellos problemas donde es necesario medir distancias en una red de distribución o en problemas de vulnerabilidad de una red de interconexión.

Ved también

En el módulo "Grafos eulerianos y grafos hamiltonianos", también se ve que la conectividad es útil para averiguar si un grafo es hamiltoniano o euleriano, aunque en este último caso es más útil conocer el grado de cada vértice.

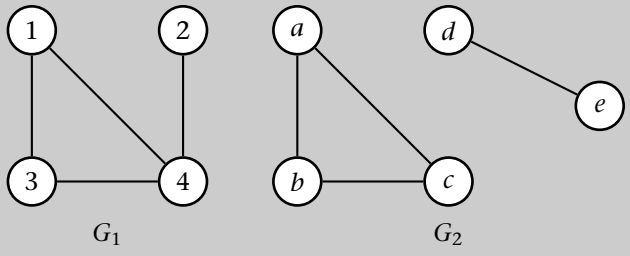
3.1. Conexión entre vértices

Definición 4

Un grafo $G = (V,A)$ es **conexo** si para cada par de vértices u y v de G existe un $u - v$ camino.

Ejemplo 9

La figura siguiente muestra dos grafos G_1 y G_2. El primero es conexo, puesto que entre cada pareja de vértices hay un camino que los une. El segundo no es conexo, puesto que, por ejemplo, entre los vértices a y e no hay ningún recorrido que los una.



Cuando un grafo G no sea conexo, se deberán determinar aquellos subgrafos de G que son conexos y que son maximales respecto a esta propiedad.

Definición 5

En el conjunto de vértices V de un grafo $G = (V,A)$ se define la relación siguiente:

$$\forall u,v \in V, \quad u \equiv v \Leftrightarrow \text{existe un } u-v \text{ camino de } G.$$

Proposición 4

Esta relación es una relación de equivalencia en el conjunto de vértices del grafo.

Relación de equivalencia

Recordad que una relación $x\ R\ y$ es de equivalencia si y sólo si para cualquier x, y, z se cumple:
(1) R es reflexiva ($x\ R\ x$),
(2) R es simétrica ($x\ R\ y \Rightarrow y\ R\ x$) y
(3) R es transitiva ($x\ R\ y$ y $y\ R\ z \Rightarrow x\ R\ z$).
La igualdad ($x = y$) es un ejemplo de relación de equivalencia.

En consecuencia, se establece una partición de $V = V_1 \cup \ldots \cup V_k$: los vértices de una misma clase son mutuamente accesibles por algún camino; vértices de clases diferentes son inaccesibles. Evidentemente, también se puede establecer, de manera parecida, una partición del conjunto de aristas $A = A_1 \cup \ldots \cup A_k$.

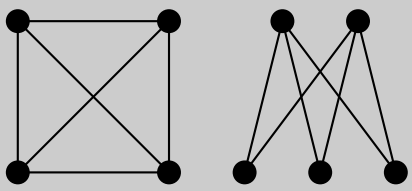
Definición 6

Las **componentes conexas** del grafo G son los subgrafos $G_i = \langle V_i \rangle$ generados por cada una de estas clases de equivalencia V_i; así, se puede expresar el grafo como unión de componentes conexas:

$$G = G_1 \cup \ldots \cup G_k$$

Ejemplo 10

El grafo G no es conexo. Tiene dos componentes conexas (el grafo completo de orden 4, y el grafo bipartito completo de orden 3+2). Así, $G = K_4 \cup K_{3,2}$.



Por la propia definición, toda componente conexa de un grafo G es un grafo conexo. Así, un grafo es conexo si, y sólo si, tiene una única componente conexa.

Comparando el orden y la medida de un grafo con los órdenes y las medidas de sus componentes conexas se obtiene la relación siguiente.

Proposición 5

Si un grafo es de orden n y medida m, con k componentes conexas, de órdenes y medidas respectivas n_i, m_i ($i = 1, \ldots, k$), entonces:

$$n = \sum_{i=1}^{k} n_i, \quad m = \sum_{i=1}^{k} m_i \,.$$

El resultado que se muestra a continuación establece la medida mínima de un grafo conexo.

Proposición 6

Si G es un grafo conexo de orden n y medida m, entonces $m \geq n - 1$.

Demostración: Se demostrará por inducción respecto al orden n del grafo. Si $n = 1$, el resultado es inmediato.

Se supone que la propiedad es cierta para todo grafo conexo de orden $n-1 \geq 1$ y se prueba que también lo es para todo grafo conexo de orden n. Sea $G = (V,A)$ un grafo conexo de orden n y medida m, se debe probar que $m \geq n - 1$.

Si todos los vértices de G tienen grado mayor o igual que 2, entonces se aplica la fórmula de los grados,

$$2m = \sum_{v \in V} g(v) \geq 2n$$

y, por lo tanto, $m \geq n > n - 1$.

En caso de que G tenga un vértice v de grado 1, entonces el grafo $G' = G - v$ es un grafo conexo de orden $n - 1$ y medida $m - 1$. Aplicando la hipótesis de inducción al grafo G' se tiene que $m-1 \geq n-2$, de donde resulta que $m \geq n-1$, como se quería probar. ∎

Como consecuencia directa de este resultado se obtiene el corolario siguiente.

Proposición 7

Si G es un grafo que tiene orden n, medida m y k componentes conexas, entonces $m \geq n-k$.

Ejemplo 11

Todos los grafos con secuencia de grados 4,4,3,3,3,3,3,3 son conexos.

Se supone que exista algún grafo no conexo con esta secuencia de grados y sea $G = G_1 \cup \ldots \cup G_k$, $k \geq 2$, la descomposición del grafo como reunión de componentes conexas. Sea $n = 8$ el orden de G y sea n_i el orden de la componente conexa G_i, considerada como grafo. Obviamente se tiene $n_1 + \ldots + n_k = n = 8$. Ahora bien, los vértices son, como mínimo, de grado 3 en cada componente conexa y, en consecuencia, cada componente conexa es como mínimo de orden 4, es decir, $n_1 + \ldots + n_k \geq 4k$. A partir de la igualdad anterior se tiene que $k = 2$, $n_1 = n_2 = 4$. Ahora bien, siendo las componentes conexas de orden 4, ninguna de ellas puede contener vértices de orden 4. Por lo tanto, no existe ningún grafo no conexo de estas características.

Ejemplo 12

Cuando se elimina un vértice de un grafo conexo, el número de componentes conexas que se producen no puede superar el grado del vértice eliminado.

La afirmación es muy intuitiva; se debe expresar formalmente y demostrar. Sea $G = (V,A)$ un grafo conexo de orden $n \geq 3$ y sea $u \in V$ de grado $g(u)$; entonces el número de componentes conexas de $G-u$ es menor o igual que $g(u)$.

En efecto, sea $G' = G - u = G_1 \cup \ldots \cup G_k$, expresión del grafo como reunión de componentes conexas. Se verá que cada G_i contiene como mínimo un vértice u_i adyacente a u en G, cosa que demostraría la afirmación. Si no fuese así para algún G_{i_0}, entonces, por la conexión del grafo G, algún vértice $w \in V(G_{i_0})$ será adyacente (en G) a algún otro vértice $q \in V((G_j),(q \neq u,(j \neq i_0)))$, puesto que de lo contrario G sería no conexo. Ahora bien, si fuera así, con la eliminación de u no se crearía la componente conexa G_{i_0}.

Ejercicios

22. Demostrad que un grafo con la secuencia de grados 3,1,1,1,1,1 no puede ser conexo.

23. Encontrad el número de componentes conexas de un grafo que tiene como lista de adyacencias,

a : f, i, j
b : c, g
c : b, e, g
d : h
e : c, g, j, f
f : a, i, e
g : b, c, e
h : d
i : a, f
j : a, e

24. Si se elimina una arista de un ciclo de un grafo, ¿se puede incrementar el número de componentes conexas?

25. ¿Cuáles de los siguientes grafos son conexos?

$$T_n,\ C_n,\ \ N_n,\ N_n + N_m, T_2 \times C_n$$

En caso de que no sean conexos, indicad cuántas componentes conexas tienen.

Soluciones

22. En primer lugar se puede verificar, si se utiliza el algoritmo de Havel-Hakimi, que la secuencia 3,1,1,1,1,1 es gráfica, ya que el grafo tiene orden 6; si el grafo fuese conexo, su medida tendría que ser mayor o igual a 5. Pero, por la fórmula de los grados, $2m = 3 + 1 + 1 + 1 + 1 + 1 = 8$ y $m = 4$.

23. El grafo tiene dos componentes conexas.

24. No; en particular, en un grafo conexo nunca se produce desconexión con la eliminación de una arista que pertenece a un ciclo (puesto que siempre queda un recorrido alternativo para cada pareja de vértices que se comuniquen por un recorrido que utilice aristas del ciclo).

25. Todos son conexos menos N_n ($n > 1$), que contiene n componentes conexas, una para cada vértice.

3.2. Test de conexión

Se puede utilizar el algoritmo DFS introducido en el subapartado 2.1. para comprobar si un grafo $G = (V,A)$ es conexo. Se debe recordar que el algoritmo DFS devuelve la lista R de todos los vértices accesibles a partir de un vértice v fijado. Si la lista contiene todos los vértices del grafo, el grafo será conexo. De lo contrario, será no conexo.

```
Entrada : G = (V,A), v ∈ V
Salida : CIERTO si G es conexo. FALSO en caso contrario
algoritmo TestConexión(G,v)
  inicio
    conexo ← CIERTO
    R ← DFS(G,v)
    si |R| ≠ |V|
      entonces conexo ← FALSO
    finsi
    retorno (conexo)
  fin
```

Ya que el DFS tiene una complejidad $O(n+m)$, se puede afirmar lo siguiente: comprobar si un grafo es conexo tiene una complejidad $O(n+m)$.

Ejemplo 13

Se puede observar la aplicación del algoritmo sobre el grafo G_2 del ejemplo 9 empezando por el vértice a:

P	Vértice añadido	Vértice eliminado	R
a	a	-	[a]
ab	b	-	[a,b]
abc	c	-	[a,b,c]
ab	-	c	[a,b,c]
a	-	b	[a,b,c]
$\emptyset$	-	a	[a,b,c]

Como el grafo G_2 tiene orden 5 y R sólo contiene 3 vértices, se puede concluir que G_2 no es conexo.

Este ejemplo pone en evidencia el resultado siguiente.

Proposición 8

Sea $G = (V,A)$ un grafo y v un vértice de G. El subgrafo inducido por los vértices visitados de G empleando el algoritmo DFS es la componente conexa que contiene v.

Ejercicios

26. ¿Se podría utilizar el BFS en lugar del DFS en el test de conexión?

27. Utilizad el test de conexión y comprobad si el grafo G definido por la matriz de adyacencias siguiente es conexo.

$$B = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

28. Modificad el test de conexión para que devuelva el número de componentes conexas de un grafo G.

Soluciones

26. Sí, puesto que los dos algoritmos devuelven todos los vértices accesibles a partir de un vértice fijado.

27. Suponiendo que los vértices del grafo forman el conjunto $V = \{a,b,c,d,e\}$, se aplica el test de conexión:

P	Vértice añadido	Vértice eliminado	R
a	a	-	[a]
ab	b	-	[a,b]
abc	c	-	[a,b,c]
abcd	d	-	[a,b,c,d]
abc	-	d	[a,b,c,d]
abce	e	-	[a,b,c,d,e]
abc	-	e	[a,b,c,d,e]
ab	-	c	[a,b,c,d,e]
a	-	b	[a,b,c,d,e]
$\emptyset$	-	a	[a,b,c,d,e]

Del test se deduce que el grafo es conexo.

28. En este caso se debe repetir el test de conexión para cada componente conexa del grafo:

Entrada : $G = (V,A)$
Salida : número de componentes conexas del grafo G.
algoritmo *ComponentesConexas*(G)
  inicio
    $ncomponentes \leftarrow 0$
    $U \leftarrow V$
    mientras $U \neq \emptyset$
      $v \leftarrow$ primer vértice de U
      $R \leftarrow DFS(G,v)$
      $ncomponentes \leftarrow ncomponentes + 1$
      $U \leftarrow U - R$
    finmientras
    retorno (*ncomponentes*)
  fin

Observad que la complejidad continúa siendo $O(n+m)$, puesto que se exploran todos los vértices y las aristas de G.

4. Distancias en un grafo

En una red de carreteras en la que hay varios itinerarios para unir dos ciudades es natural preguntarse qué itinerario es el más corto, o cuál es el que logra la mínima distancia entre las dos ciudades.

En este apartado se definirá el concepto de distancia entre dos vértices de un grafo conexo. Se procederá a calcular esta distancia y también se resolverá el problema de la distancia y el camino mínimo en un grafo ponderado (grafo con un peso asociado a cada arista). Finalmente, se resolverá el problema más general de encontrar la distancia entre un vértice y el resto de los vértices del grafo, y un camino que logre esta distancia, mediante los algoritmos de Dijkstra y Floyd.

4.1. Distancia entre vértices

Definición 7

Dado un grafo conexo $G = (V,A)$, la **distancia** entre dos de sus vértices es la mínima de las longitudes de los caminos que conecten estos vértices:

$$d_G(u,v) = \min\{\ell(C) \mid C \text{ es un } u-v \text{ camino}\}$$

En el caso no conexo, la distancia entre dos vértices de una misma componente conexa se define como en el caso anterior. En el caso de vértices mutuamente inaccesibles, se asigna el valor convencional infinito (∞).

Cuando quede claro en qué grafo se calcula la distancia entre dos vértices u y v se pondrá, simplemente, $d(u,v)$.

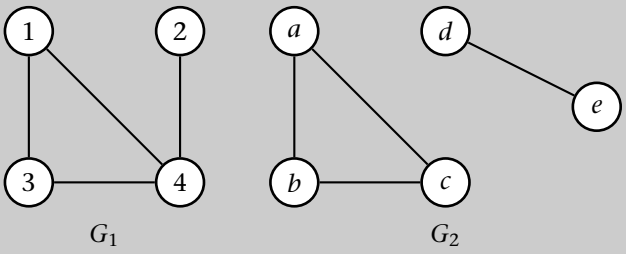
Definición 8

Dado un grafo conexo $G = (V,A)$, el **diámetro** $D(G)$ es

$$D(G) = \max\{d_G(u,v) | u,v \in V\}$$

Ejemplo 14

Se consideran los grafos G_1 y G_2 del ejemplo 9.



En G_1, $d(1,2) = 2$, $d(1,3) = 1$, $d(1,4) = 1$. $D(G_1) = 2$.

En G_2, $d(a,c) = d(a,b) = d(b,c) = 1$, $d(a,d) = \infty$, por lo tanto, $D(G_2) = \infty$.

Ejemplo 15

Se puede observar que, dado un grafo cualquiera no conexo G, el grafo complementario G^c es conexo y de diámetro ≤ 2.

Sean x, y dos vértices de componentes conexas diferentes de G que, por lo tanto, no son adyacentes en G; en consecuencia lo son en el complementario G^c y $d_{G^c}(x,y) = 1$.

Sean ahora x, y de la misma componente conexa G_i de G. Dado que el grafo no es conexo por hipótesis, existe otra componente conexa diferente G_j que contiene un vértice $z \in V(G_j)$; entonces x, y no son adyacentes a z en el grafo G y, en consecuencia, lo son en el complementario. Por lo tanto, en el complementario los vértices x, y están conectados por un camino a través de z, concretamente el camino x,z,y, y resulta, pues, que $d_{G^c}(x,y) \leq 2$ (observad que no se puede afirmar que la distancia sea 2, puesto que podría pasar que x, y fueran no adyacentes en G, caso en el que serían adyacentes en el complementario).

A partir de la distancia entre vértices es posible una caracterización de los grafos bipartitos.

Teorema 9

Un grafo es bipartito si, y sólo si, todos los ciclos son de longitud par.

Demostración: Se deben que probar dos implicaciones:

1) Si el grafo es bipartito, entonces todos los ciclos son de longitud par.

2) Si todos los ciclos de un grafo son de longitud par, entonces el grafo es bipartito.

Se supone que el grafo es $G = (V,A)$.

1) Se supone que el grafo es (V_1,V_2)-bipartito; la conclusión se deriva con facilidad del hecho de que todo camino ha de utilizar alternativamente vértices de V_1 y de V_2.

2) Se debe definir adecuadamente una bipartición (V_1,V_2) del conjunto de vértices de manera que el grafo resulte ser (V_1,V_2)-bipartito.

Sea $u_0 \in V$; se definen los conjuntos

$$V_1 = \{v \in V \mid d(u_0,v) \text{ es par}\} \quad V_2 = \{v \in V \mid d(u_0,v) \text{ es impar}\}$$

Estos conjuntos determinan una partición del conjunto de vértices del grafo y ahora se trata de ver que el grafo es (V_1,V_2)-bipartito.

Para comprobarlo, se debe concluir que no hay aristas que conecten vértices de un mismo conjunto de la partición; se probará para parejas de vértices de V_1 y análogamente se procedería para V_2. Sean, pues, $u,v \in V_1$ y se supone que existe la arista $a = \{u,v\} \in A$. Se verá que se llega a alguna contradicción, a partir del hecho de que todos los ciclos son de longitud par. Sean R_1 un $u_0 - u$ camino de longitud mínima y R_2 un $u_0 - v$ camino de longitud mínima, los dos de longitudes pares, por definición de V_1. Recorriendo los caminos anteriores desde el inicio u_0, se logrará un vértice w_0 que es el último que comparten los caminos R_1, R_2. Por ser mínimo, los subcaminos $u_0 - w_0$ sobre R_1, R_2 tienen que ser de la misma longitud k. Se consideran ahora los subcaminos $S_1 : w_0 - u$ sobre R_1 y $S_2 : w_0 - u$ sobre R_2; entonces resulta que $\ell(R_1) = k + \ell(S_1)$ y $\ell(R_2) = k + \ell(S_2)$, de donde la diferencia $\ell(S_2) - \ell(S_1)$ es par y, en consecuencia, $\ell(S_1),\ell(S_2)$ tienen que ser de la misma paridad. Considerando el camino formado por estas secciones (pasando por w_0), resulta un $u - v$ camino de longitud par. Conjuntamente con la arista $a = \{u,v\}$ se crearía un ciclo de longitud impar, que contradiría la hipótesis.

∎

Ejercicios

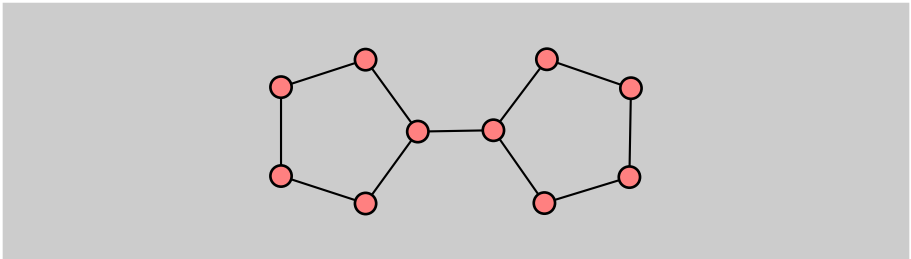
29. Comprobad que la distancia dada entre vértices de un grafo conexo satisface las propiedades generales de una métrica:

a) $d(u,v) \geq 0$ y $d(u,v) = 0$ si, y sólo si, $u = v$;

b) $d(u,v) = d(v,u)$;

c) $d(u,v) \leq d(u,w) + d(w,v)$ (desigualdad triangular).

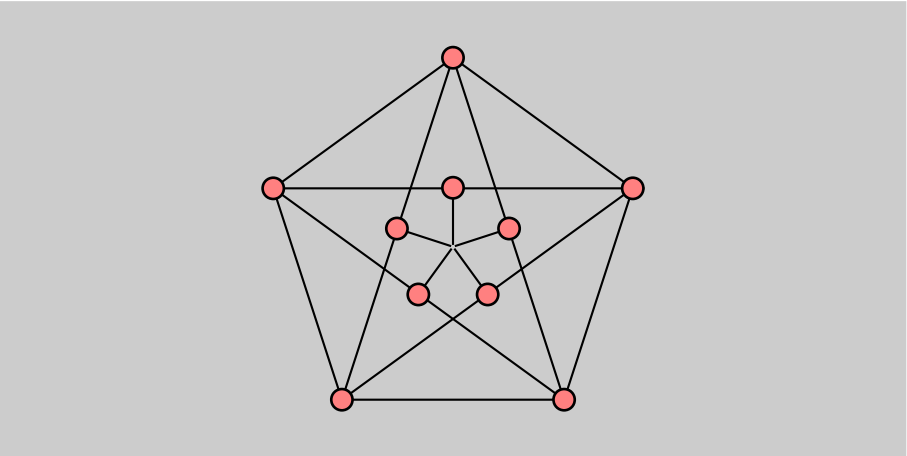
30. Indicad cuál es el diámetro del grafo siguiente.



31. ¿Es cierta la propiedad: $u \approx v \Leftrightarrow d(u,v) = 1$?

32. Calculad el diámetro de los grafos K_n, $K_{n,m}$, C_n, T_n.

33. Utilizad la caracterización de los grafos bipartitos para decidir si el grafo siguiente es bipartito.



34. Los grafos acíclicos son bipartitos. ¿Cierto o falso?

Soluciones

29. Las propiedades *a* y *b* son obvias. Para demostrar la propiedad *c*, sea C el camino que logra la $d(u,v)$. Sea C_1 el camino que logra la $d(u,w)$ y C_2 el camino que logra la $d(w,v)$. Entonces, si se une C_1 y C_2 se tiene un camino que une u y v. Por la definición de distancia, la longitud de este camino tiene que ser mayor o igual que la $d(u,v)$.

30. El diámetro es 5.

31. Sí (acabad de justificar la respuesta).

32. $D(K_n) = 1$, porque la distancia de un vértice a cualquier otro siempre es 1.

$D(K_{n,m}) = 2$, porque se puede ir de un vértice a cualquier otro recorriendo dos aristas.

$D(C_n) = \frac{n}{2}$ si n es par. $D(C_n) = \frac{n-1}{2}$ si n es impar.

$D(T_n) = n - 1$, que es la distancia entre los dos extremos.

33. No es bipartito, puesto que hay ciclos de longitud 5.

34. Cierto, en aplicación de la caracterización de grafos bipartitos en términos de los ciclos, puesto que no hay ningún ciclo y, en particular, ninguno de longitud impar.

4.2. El problema del camino mínimo en un grafo

Un ejemplo típico de un grafo es una red de carreteras que conectan un conjunto de ciudades. El problema algorítmico más natural en

este grafo es encontrar el camino más corto (**distancia**) entre un par de ciudades (**vértices**).

La distancia entre dos vértices en un grafo no ponderado se puede encontrar, de manera eficiente, si se aplica el algoritmo de búsqueda primeramente en anchura (**BFS** o *breadth first search*). Si el grafo es ponderado, entonces la distancia entre dos vértices no se puede calcular directamente y hay que aplicar algoritmos específicos.

El **algoritmo de Dijkstra** encuentra la distancia entre dos vértices construyendo un árbol desde el vértice inicial u_0 a cada uno de los otros vértices del grafo.

El **algoritmo de Floyd** encuentra la distancia entre todos los pares de vértices de un grafo.

Observación

Un grafo simple se puede interpretar como un grafo ponderado si se asocia a cada arista un peso 1.

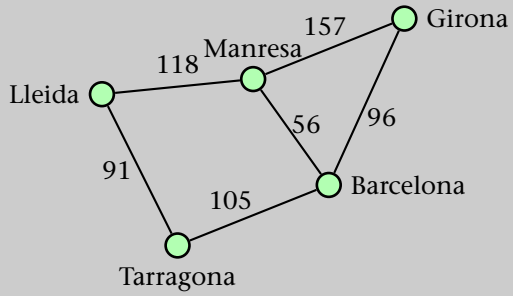
Definición 9

Un **grafo ponderado** es un par (G,w) donde $G = (V,A)$ es un grafo y w es una función $w : A \to \mathbb{R}$ que asigna pesos a las aristas del grafo.

Ejemplo 16

En una red de comunicaciones puede ser interesante la asignación a cada arista de un peso indicativo del tiempo que cuesta recorrer la arista en cuestión, o los kilómetros, o el coste económico correspondiente en otros aspectos que dependerán del problema y del modelo que se haya llegado a construir.

El grafo siguiente es un grafo ponderado. El peso de cada arista es la distancia kilométrica entre dos ciudades.



Normalmente los pesos son positivos o nulos, pero también se pueden considerar situaciones en las que sean negativos.

Definición 10

Dado un grafo ponderado (G,w) y un camino $C : v_0,v_1,\ldots,v_k$ se definen el **peso del camino** C como $w(C) = \sum_{i=1}^{k} w(v_{i-1},v_i)$, y la **distancia** entre dos vértices $u,v \in G$ como

$$d_G(u,v) = \min\{w(C) \mid C \text{ es un } u - v \text{ camino }\}$$

Si el grafo G no es ponderado (o todos los pesos son iguales a 1), entonces esta definición coincide con la que se ha dado anteriormente (definición 7).

Como antes, en el caso de vértices mutuamente inaccesibles se asigna el valor convencional ∞ a su distancia.

Ejemplo 17

En el grafo ponderado del ejemplo 16, el camino C_1 : Barcelona, Manresa, Lleida tiene un peso 174. En cambio, el camino C_2 : Barcelona, Tarragona, Lleida tiene un peso 196. La distancia entre Barcelona y Lleida es 174 pasando por Manresa.

Variantes del problema del camino mínimo

El problema del camino mínimo admite diversas variantes que se pueden resolver utilizando adaptaciones de los mismos algoritmos:

1) Camino mínimo desde un vértice inicial (*single source shortest path*): dado (G,w) y $s \in V$, buscad la $d(s,v)$ para todo $v \in V$.

2) Camino mínimo hasta un vértice destino (*single destination shortest path*): dado (G,w) y $t \in V$, buscad la $d(v,t)$ para todo $v \in V$.

3) Camino mínimo entre un par de vértices (*single pair shortest path*): dado (G,w) y $s,t \in V$, buscad la $d(s,t)$.

4) Camino mínimo entre todos los pares de vértices (*all pairs shortest path*): dado (G,w), buscad la $d(u,v)$ para todo $u,v \in V$.

Ejemplo 18

De nuevo, se utiliza en este ejemplo el grafo ponderado del ejemplo 16.

Si se ordenan las ciudades alfabéticamente, entonces la solución a las cuatro variantes del problema del camino mínimo en este grafo será:

1) Camino mínimo desde Barcelona:

(0,Barcelona), (96,Girona), (174,Lleida), (56,Manresa), (105,Tarragona).

2) Camino mínimo hasta Tarragona:

(105,Barcelona), (201,Girona), (91,Lleida), (161,Manresa), (0,Tarragona).

3) Camino mínimo entre Girona y Lleida: 270 pasando por Barcelona y Manresa.

4) Camino mínimo entre todos los pares de ciudades:

	Barcelona	Girona	Lleida	Manresa	Tarragona
Barcelona	0	96	174	56	105
Girona	96	0	270	152	201
Lleida	174	270	0	118	91
Manresa	56	152	118	0	161
Tarragona	105	201	91	161	0

A continuación, se estudiarán los algoritmos específicos para resolver las variantes 1 y 4. Se pospondrá para los ejercicios la resolución de las variantes 2 y 3.

4.2.1. Algoritmo de Dijkstra

El algoritmo de Dijkstra se aplica sobre un grafo (o digrafo) ponderado y calcula la distancia desde un vértice inicial s al resto de los vértices del grafo. A cada paso, se etiquetarán los vértices con $(dist(u),v)$, donde $dist(u)$ es la distancia mínima actual del vértices s al vértices u, y v es el predecesor de u en el camino mínimo que une s y u.

Formulación del algoritmo de Dijkstra

Estructuras necesarias para la formulación del algoritmo:

- Un grafo ponderado (G,w) representado mediante una lista de adyacencias.
- Un conjunto U de los vértices que se han visitado, en el orden en el que se ha hecho.
- Una tabla de distancias, $dist(\cdot)$, indexada por los vértices de G, que registra la distancia del vértice inicial a los vértices que se van visitando.
- Al final, la tabla $dist(\cdot)$ registra la distancia desde el vértice inicial al resto de los vértices.

Cuando es posible visitar más de un vértice, siempre se elige el de índice mínimo en la ordenación de los vértices disponibles.

A cada paso se fija la distancia de uno de los vértices del grafo. Así, tras n pasos se habrá calculado la distancia a todos los vértices del grafo.

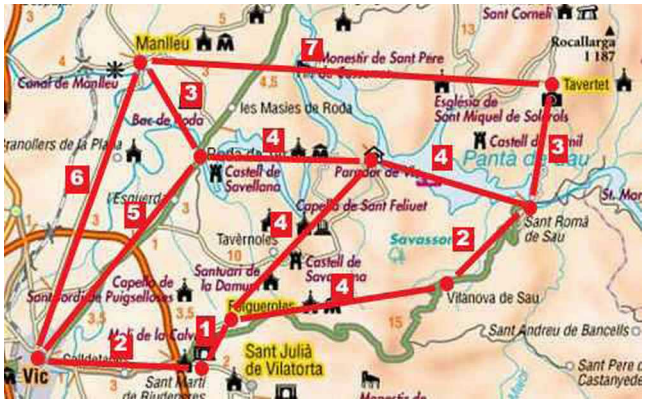
Entrada : (G,w) de orden n y un vértice inicial s.
Salida : La distancia, $dist(\cdot)$, de s al resto de los vértices.
algoritmo *Dijkstra*(G,s)
  inicio
    $U \leftarrow \emptyset$
    para $v \in V \backslash \{s\}$
      $dist(v) \leftarrow \infty$
      Se etiqueta v con $(dist(v),s)$
    finpara
    $dist(s) \leftarrow 0$
    Se etiqueta s con $(dist(s),s)$
    para $i \leftarrow 0$ **hasta** $n-1$
      u_i vértice que logra lo $\min\{dist(v) \mid v \in V - U\}$
      $U \leftarrow U \cup \{u_i\}$
      para $v \in V - U$ adyacente a u_i
        si $dist(u_i) + w(u_i,v) < dist(v)$
          entonces $dist(v) \leftarrow dist(u_i) + w(u_i,v)$
            Se etiqueta v con $(dist(v),u_i)$
        finsi
      finpara
    finpara
    retorno $(dist)$
  fin

El algoritmo se puede utilizar para obtener un camino de longitud mínima entre el vértice inicial y cualquier otro vértice, puesto que después de aplicar el algoritmo, todo vértice v tiene asociada una etiqueta $(dist(v),u_i)$ indicativa de la distancia del vértice v al vértice de partida, $dist(v) = d(s,v)$, y del camino seguido para calcular la distancia. Si $v \neq s$ se obtiene un s–v camino de longitud mínima con $s = q_0,q_1,\ldots,q_k = v$, donde los q_i están etiquetados con $(dist(q_i),q_{i-1})$ para $i = 1,\ldots,k$.

Simulación del algoritmo de Dijkstra

Ejemplo 19

Considerad el grafo definido sobre el mapa siguiente, que representa las distancias aproximadas entre varias localizaciones de la comarca de Osona:



Se puede utilizar el algoritmo de Dijkstra para encontrar la distancia mínima entre Vic y el resto de las localidades. Se utiliza una tabla como la siguiente para representar los diferentes pasos del algoritmo de Dijkstra:

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	0	0	0	0	0	0	0	0	0
$(dist(\cdot),\cdot)$	$(0,V)$	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)

Esta tabla representa el estado inicial del algoritmo de Dijkstra. Los vértices son: **V**ic, **M**anlleu, **C**astell de **S**avellana, **S**ant **M**artí de Riudeperes, **F**olgueroles, **V**ilanova de **S**au, **S**ant **R**omà de Sau, **P**arador de **V**ic, Tavertet. El conjunto U está representado por un **mapa de bits**, es decir, contiene un **1** si el vértice correspondiente pertenece a U o **0** en caso contrario.

i = 0. El vértice de peso mínimo es el vértice V. Se pueden etiquetar los vértices M, CS, SM.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	**1**	0	0	0	0	0	0	0	0
$(dist(\cdot),\cdot)$	$(0,V)$	$\mathbf{(6,V)}$	$\mathbf{(5,V)}$	$\mathbf{(2,V)}$	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)

i = 1. El vértice de peso mínimo es el vértice SM. Se puede etiquetar el vértice F.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	0	0	**1**	0	0	0	0	0
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$\mathbf{(3,SM)}$	(∞,V)	(∞,V)	(∞,V)	(∞,V)

i = 2. El vértice de peso mínimo es el vértice F. Se pueden etiquetar los vértices VS y PV.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	0	0	1	**1**	0	0	0	0
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$(3,SM)$	$\mathbf{(7,F)}$	(∞,V)	$\mathbf{(7,F)}$	(∞,V)

i = 3. El vértice de peso mínimo es el vértice CS, pero no se etiqueta ningún nuevo vértice.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	0	**1**	1	1	0	0	0	0
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$(3,SM)$	$(7,F)$	(∞,V)	$(7,F)$	(∞,V)

i = 4. El vértice de peso mínimo es el vértice M. Se puede etiquetar el vértice T.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	**1**	1	1	1	0	0	0	0
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$(3,SM)$	$(7,F)$	(∞,V)	$(7,F)$	**$(13,M)$**

i = 5. El vértice de peso mínimo es el vértice VS. Se puede etiquetar el vértice SR.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	1	1	1	1	**1**	0	0	0
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$(3,SM)$	$(7,F)$	**$(9,VS)$**	$(7,F)$	$(13,M)$

i = 6. El vértice de peso mínimo es el vértice PV, pero no se etiqueta ningún nuevo vértice.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	1	1	1	1	1	0	**1**	0
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$(3,SM)$	$(7,F)$	$(9,VS)$	$(7,F)$	$(13,M)$

i = 7. El vértice de peso mínimo es el vértice SR. Se puede etiquetar el vértice T.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	1	1	1	1	1	**1**	1	0
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$(3,SM)$	$(7,F)$	$(9,VS)$	$(7,F)$	**$(12,SR)$**

i = 8. Finalmente, el vértice de peso mínimo es el vértice T, pero no se etiqueta ningún nuevo vértice.

Vértices	V	M	CS	SM	F	VS	SR	PV	T
U	1	1	1	1	1	1	1	1	**1**
$(dist(\cdot),\cdot)$	$(0,V)$	$(6,V)$	$(5,V)$	$(2,V)$	$(3,SM)$	$(7,F)$	$(9,VS)$	$(7,F)$	$(12,SR)$

La última tabla muestra la distancia mínima entre Vic y el resto de las ciudades. Observad que también es posible reconstruir el itinerario que habría que seguir para desplazarse de Vic a cualquiera de las ciudades. Por ejemplo, la distancia de Vic a Tavertet es 12 y el itinerario se reconstruye a partir del etiquetado de la última tabla: T(12,SR), SR(9,VS), VS(7,F), F(3,SM),

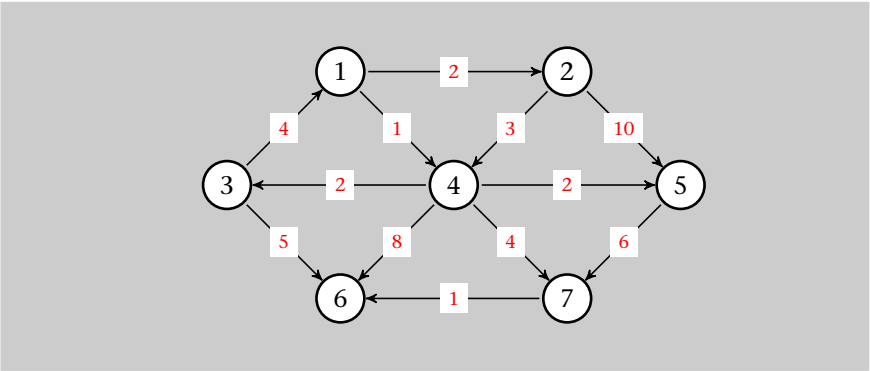
SM(2,V). El itinerario será, Vic, Sant Martí de Riudeperes, Folgueroles, Vilanova de Sau, Sant Romà de Sau, Tavertet.

A partir de estas tablas que presentan cada uno de los pasos, se puede construir la **tabla del algoritmo de Dijkstra**, que incluye cada una de las filas $(dist(\cdot),\cdot)$ (se marca con un asterisco el vértice que se visita):

V	M	CS	SM	F	VS	SR	PV	T
(0,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)
(0,V)*	(6,V)	(5,V)	(2,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)	(∞,V)
(0,V)	(6,V)	(5,V)	(2,V)*	(3,SM)	(∞,V)	(∞,V)	(∞,V)	(∞,V)
(0,V)	(6,V)	(5,V)	(2,V)	(3,SM)*	(7,F)	(∞,V)	(7,F)	(∞,V)
(0,V)	(6,V)	(5,V)*	(2,V)	(3,SM)	(7,F)	(∞,V)	(7,F)	(∞,V)
(0,V)	(6,V)*	(5,V)	(2,V)	(3,SM)	(7,F)	(∞,V)	(7,F)	(13,M)
(0,V)	(6,V)	(5,V)	(2,V)	(3,SM)	(7,F)*	(9,VS)	(7,F)	(13,M)
(0,V)	(6,V)	(5,V)	(2,V)	(3,SM)	(7,F)	(9,VS)	(7,F)*	(13,M)
(0,V)	(6,V)	(5,V)	(2,V)	(3,SM)	(7,F)	(9,VS)*	(7,F)	(12,SR)
(0,V)	(6,V)	(5,V)	(2,V)	(3,SM)	(7,F)	(9,VS)	(7,F)	(12,SR)*

Ejemplo 20

Considerad el digrafo definido por el gráfico siguiente y calculad las distancias desde el vértice 1 al resto de los vértices.



En el caso de los digrafos, la distancia de un arco sólo tiene sentido en la dirección indicada por el arco. Si no hay ningún arco en sentido contrario, se utiliza como distancia ∞, igual que en el caso donde no hay ninguna arista.

En este caso, la tabla del algoritmo de Dijkstra del digrafo es ésta (se señala con un asterisco el vértice que se visita):

1	2	3	4	5	6	7
(0,1)	(∞,1)	(∞,1)	(∞,1)	(∞,1)	(∞,1)	(∞,1)
(0,1)*	(2,1)	(∞,1)	(1,1)	(∞,1)	(∞,1)	(∞,1)
(0,1)	(2,1)	(3,4)	(1,1)*	(3,4)	(9,4)	(5,4)
(0,1)	(2,1)*	(3,4)	(1,1)	(3,4)	(9,4)	(5,4)
(0,1)	(2,1)	(3,4)*	(1,1)	(3,4)	(8,3)	(5,4)
(0,1)	(2,1)	(3,4)	(1,1)	(3,4)*	(8,3)	(5,4)
(0,1)	(2,1)	(3,4)	(1,1)	(3,4)	(6,7)	(5,4)*
(0,1)	(2,1)	(3,4)	(1,1)	(3,4)	(6,7)*	(5,4)

De la última fila de la tabla se deduce que la distancia del vértice 1 al resto de los vértices vale $d(1,1) = 0, d(1,2) = 2, d(1,3) = 3, d(1,4) = 1, d(1,5) = 3, d(1,6) = 6$, $d(1,7) = 5$.

Análisis del algoritmo de Dijkstra

Para analizar este algoritmo se dividirá en dos partes:

1) Se inicia una tabla de tamaño n con una complejidad $O(n)$.

2) El bucle principal se ejecuta n veces. En el paso i-ésimo se calcula el mínimo de una lista que contiene $n–i$ elementos. Esto se puede hacer con $n - i$ comparaciones.

En el bucle más interno se actualizan las etiquetas de los vértices adyacentes al vértice analizado. $n - i - 1$ es el número máximo de vértices adyacentes que se actualizan en el paso i-ésimo.

Resumiendo, en el bucle principal se hacen

$$\sum_{i=0}^{n-1} n - i + \sum_{i=0}^{n-1} n - i - 1 = n^2$$

operaciones elementales.

Esto da una complejidad $O(n^2)$.

Todo el algoritmo, pues, tendrá una complejidad $\max\{O(n), O(n^2)\} = O(n^2)$, independientemente del número de aristas del grafo.

4.2.2. Camino mínimo en un grafo no ponderado

Si el grafo es no ponderado (o todas las aristas tienen peso 1), entonces se puede utilizar el algoritmo de exploración en anchura (BFS) para calcular la distancia entre un vértice inicial y el resto de los vértices del grafo.

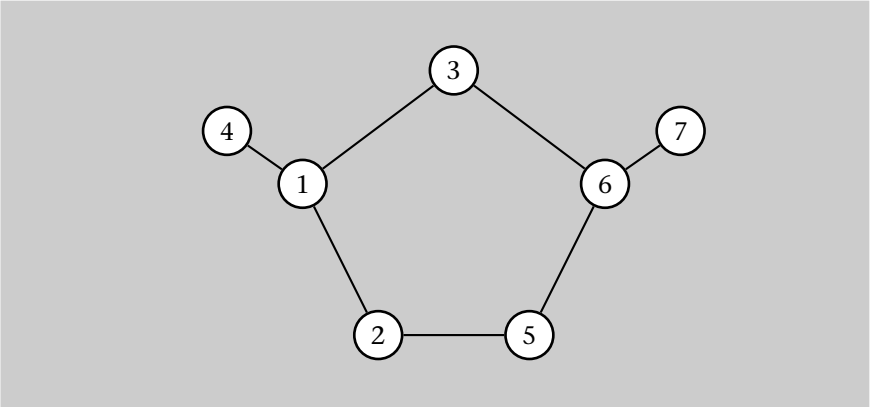
Las estructuras de datos necesarios para formular el algoritmo son las mismas que en el BFS si se añade una tabla $dist(\cdot)$ que almacena las distancias del vértice inicial al resto de los vértices.

Entrada : $G = (V,A)$ no ponderado, $s \in V$
Salida : La distancia, $dist(\cdot)$, de s al resto de los vértices.

```
algoritmo Distancias_no_ponderado(G,s)
  inicio
    Q ← ∅
    para w ∈ V
          estado[w] ← 0
          dist[w] ← ∞
    finpara
    estado[s] ← 1
    dist[s] ← 0
    añadir(Q,s)
    mientras Q ≠ ∅
                w ← primero(Q)
                para u adyacente a w
                      si estado[u] = 0
                         entonces añadir(Q,u)
                                  estado[u] ← 1
                                  dist[u] ← dist[w] + 1
                      finsi
                finpara
                eliminar(Q)
    finmientras
    retorno (dist)
  fin
```

Ejemplo 21

Se considera el grafo no ponderado representado en la figura.



La tabla registra el funcionamiento del algoritmo para este grafo, con vértice de inicio $s = 1$.

Q	Vértice añadido	Vértice eliminado	*dist*
1	1	-	[0,∞,∞,∞,∞,∞,∞]
12	2	-	[0,1,∞,∞,∞,∞,∞]
123	3	-	[0,1,1,∞,∞,∞,∞]
1234	4	-	[0,1,1,1,∞,∞,∞]
234	-	1	[0,1,1,1,∞,∞,∞]
2345	5	-	[0,1,1,1,2,∞,∞]
345	-	2	[0,1,1,1,2,∞,∞]
3456	6	-	[0,1,1,1,2,2,∞]
456	-	3	[0,1,1,1,2,2,∞]
56	-	4	[0,1,1,1,2,2,∞]
6	-	5	[0,1,1,1,2,2,∞]
67	7	-	[0,1,1,1,2,2,3]
7	-	6	[0,1,1,1,2,2,3]
∅	-	7	[0,1,1,1,2,2,3]

Si se compara este algoritmo con el algoritmo de Dijkstra aplicado a un grafo no ponderado, se puede observar que mientras Dijkstra tiene una complejidad $O(n^2)$, el BFS tiene una complejidad $O(n+m)$. Para grafos poco densos (pocas aristas), el algoritmo BFS tiene un comportamiento más eficiente que el algoritmo de Dijkstra.

4.2.3. Algoritmo de Floyd

El problema de buscar los caminos mínimos entre todos los pares de vértices de un grafo se puede resolver si se aplica n veces el algoritmo de Dijkstra:

Entrada : (G,w) de orden n
Salida : La distancia, $d(\cdot,\cdot)$, entre todos los pares de vértices.
algoritmo *todos_los_pares*(G)
  inicio
    para $s \in V$
      $d(s,\cdot) \leftarrow Dijkstra(G,s)$
    finpara
    retorno (d)
  fin

que tendría una complejidad $O(n^3)$.

Objetivo del algoritmo de Floyd

El algoritmo de Floyd busca los caminos mínimos entre todos los pares de vértices de un grafo.

Otra alternativa es utilizar un algoritmo específico, de comparable eficiencia al algoritmo de Dijkstra, pero con un comportamiento mejor para grafos densos. Es el **algoritmo de Floyd**.

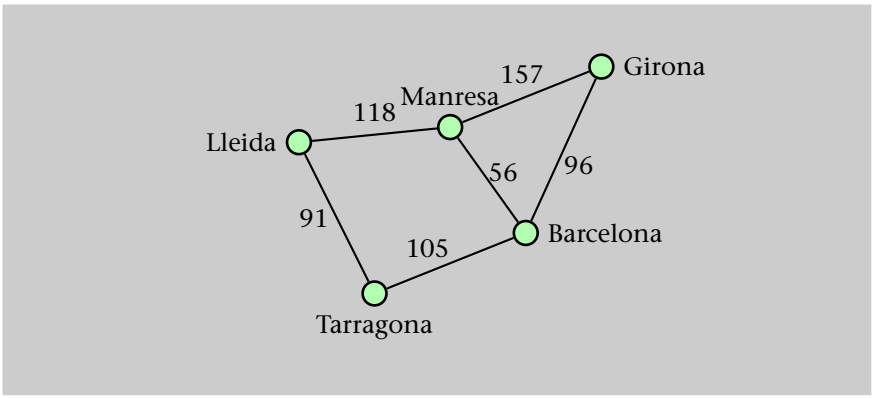
El algoritmo de Floyd considera los vértices ordenados y, en el paso k-ésimo, compara el peso del camino obtenido hasta el momento utilizando los $k-1$ vértices anteriores, con el camino obtenido añadiendo el vértice k-ésimo.

Se etiquetaran los vértices $V = \{1,2,3,\ldots,n\}$ y se utilizará una matriz bidimensional d_{ij} $(1 \leq i,j \leq n)$ para almacenar las distancias.

Entrada : (G,w) de orden n
Salida : La distancia, $d(\cdot,\cdot)$, entre todos los pares de vértices.
algoritmo *Floyd*(G)
  inicio
    para $i \leftarrow 1$ **hasta** n
      para $j \leftarrow 1$ **hasta** n
        si $i = j$ **entonces** $d_{ij}^0 \leftarrow 0$ **finsi**
        si $(i,j) \in A$ **entonces** $d_{ij}^0 \leftarrow w(i,j)$ **finsi**
        si $(i,j) \notin A$ **entonces** $d_{ij}^0 \leftarrow \infty$ **finsi**
      finpara
    finpara
    para $k \leftarrow 1$ **hasta** n
      para $i \leftarrow 1$ **hasta** n
        para $j \leftarrow 1$ **hasta** n
          $d_{ij}^k \leftarrow \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
        finpara
      finpara
    finpara
    retorno (d_{ij}^n)
  fin

Ejemplo 22

Si se aplica el algoritmo de Floyd al grafo del ejemplo 16,



se obtiene la serie de matrices bidimensionales (se supone que las ciudades están ordenadas alfabéticamente):

$$d^0 = \begin{pmatrix} 0 & 96 & \infty & 56 & 105 \\ 96 & 0 & \infty & 157 & \infty \\ \infty & \infty & 0 & 118 & 91 \\ 56 & 157 & 118 & 0 & \infty \\ 105 & \infty & 91 & \infty & 0 \end{pmatrix} \quad d^1 = \begin{pmatrix} 0 & 96 & \infty & 56 & 105 \\ 96 & 0 & \infty & 152 & 201 \\ \infty & \infty & 0 & 118 & 91 \\ 56 & 152 & 118 & 0 & 161 \\ 105 & 201 & 91 & 161 & 0 \end{pmatrix}$$

$$d^2 = \begin{pmatrix} 0 & 96 & \infty & 56 & 105 \\ 96 & 0 & \infty & 152 & 201 \\ \infty & \infty & 0 & 118 & 91 \\ 56 & 152 & 118 & 0 & 161 \\ 105 & 201 & 91 & 161 & 0 \end{pmatrix} \quad d^3 = \begin{pmatrix} 0 & 96 & \infty & 56 & 105 \\ 96 & 0 & \infty & 152 & 201 \\ \infty & \infty & 0 & 118 & 91 \\ 56 & 152 & 118 & 0 & 161 \\ 105 & 201 & 91 & 161 & 0 \end{pmatrix}$$

$$d^4 = \begin{pmatrix} 0 & 96 & 174 & 56 & 105 \\ 96 & 0 & 270 & 152 & 201 \\ 174 & 270 & 0 & 118 & 91 \\ 56 & 152 & 118 & 0 & 161 \\ 105 & 201 & 91 & 161 & 0 \end{pmatrix} \quad d^5 = \begin{pmatrix} 0 & 96 & 174 & 56 & 105 \\ 96 & 0 & 270 & 152 & 201 \\ 174 & 270 & 0 & 118 & 91 \\ 56 & 152 & 118 & 0 & 161 \\ 105 & 201 & 91 & 161 & 0 \end{pmatrix}$$

Observad que d^5 coincide con la tabla obtenida en el ejemplo 18.

Análisis del algoritmo de Floyd

El algoritmo de Floyd es muy fácil de analizar. Básicamente tiene dos partes: la iniciación de la matriz de distancias y el cálculo de las distancias.

La iniciación tiene una complejidad $O(n^2)$ y el cálculo de las distancias tiene una complejidad $O(n^3)$. Así, todo el algoritmo tendrá una complejidad $O(n^3)$, independientemente del número de aristas y comparable a la eficiencia del algoritmo de Dijkstra aplicado n veces.

Ejercicios

35. La tabla siguiente representa la distancia entre varios aeropuertos unidos por una línea aérea.

	A	B	C	D	E	F	G
A	0	5	3	2	-	-	-
B	5	0	2	-	3	-	1
C	3	2	0	7	7	-	-
D	2	-	7	0	2	6	-
E	-	3	7	2	0	1	1
F	-	-	-	6	1	0	-
G	-	1	-	-	1	-	0

a) ¿Cuál es la distancia mínima entre el aeropuerto A y el resto de los aeropuertos?

b) ¿Cuál es el número mínimo de transbordos de avión que habrá que hacer para ir del aeropuerto A al resto de los aeropuertos?

36. Si en el algoritmo de Dijkstra se modifica la condición:

$$dist(u_i) + w(u_i,v) \leq dist(v)$$

cambiando $\leq$ por $<$, ¿cómo repercutirá en el funcionamiento del algoritmo? ¿Se obtendrá la misma distancia entre dos vértices? ¿Se obtendrá el mismo camino?

37. Suponed que se numeran los vértices del grafo K_n, $V = \{1,2,3,\ldots,n\}$, y a cada arista se le asigna un peso, $w(i,j) = |j - i|$. ¿Cuál será la distancia entre dos vértices cualesquiera?

38. Modificad el algoritmo de Dijkstra para obtener la solución a la variante del problema del camino mínimo (*single pair shortest path*), es decir, dados (G,w) y dos vértices $s,t \in V$, encontrad la distancia mínima de s a t.

39. A partir del algoritmo de Dijkstra, proponed una solución a la variante del problema del camino mínimo (*single destination shortest path*), es decir, dados (G,w) y un vértice $t \in V$, encontrad la distancia mínima de todos los vértices de G a t.

40. La tabla siguiente representa el tiempo necesario para conectar directamente varios nodos de una red. El símbolo "-" significa que los dos nodos no son accesibles directamente. Utilizando el algoritmo de Floyd, encontrad el tiempo mínimo necesario para conectar dos a dos todos los pares de nodos de la red

	1	2	3	4	5
1	0	3	8	-	4
2	-	0	-	1	7
3	-	4	0	-	-
4	2	-	5	0	-
5	-	-	-	6	0

Soluciones

35. En el primer caso se debe aplicar el algoritmo de Dijkstra sobre el grafo que se obtiene a partir de la tabla de distancias:

A	B	C	D	E	F	G
(0,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)
(0,A)*	(5,A)	(3,A)	(2,A)	(∞,A)	(∞,A)	(∞,A)
(0,A)	(5,A)	(3,A)	(2,A)*	(4,D)	(8,D)	(∞,A)
(0,A)	(5,A)	(3,A)*	(2,A)	(4,D)	(8,D)	(∞,A)
(0,A)	(5,A)	(3,A)	(2,A)	(4,D)*	(5,E)	(5,E)
(0,A)	(5,A)*	(3,A)	(2,A)	(4,D)	(5,E)	(5,E)
(0,A)	(5,A)	(3,A)	(2,A)	(4,D)	(5,E)*	(5,E)
(0,A)	(5,A)	(3,A)	(2,A)	(4,D)	(5,E)	(5,E)*

La última fila de la tabla da las distancias mínimas entre el aeropuerto A y el resto de los aeropuertos.

En el segundo caso, se puede considerar el mismo grafo pero ahora sin pesos, es decir, sólo interesa saber el número de aristas que unen el aeropuerto A con el resto de los aeropuertos. En este caso, se puede aplicar el algoritmo para calcular distancias en el caso no ponderado:

Q	Vértice añadido	Vértice eliminado	*dist*
A	A	-	[0,∞,∞,∞,∞,∞,∞]
AB	B	-	[0,1,∞,∞,∞,∞,∞]
ABC	C	-	[0,1,1,∞,∞,∞,∞]
ABCD	D	-	[0,1,1,1,∞,∞,∞]
BCD	-	A	[0,1,1,1,∞,∞,∞]
BCDE	E	-	[0,1,1,1,2,∞,∞]
BCDEG	G	-	[0,1,1,1,2,2,∞]
CDEG	-	B	[0,1,1,1,2,2,∞]
DEG	-	C	[0,1,1,1,2,2,∞]
DEGF	F	-	[0,1,1,1,2,2,2]
EGF	-	D	[0,1,1,1,2,2,2]
GF	-	E	[0,1,1,1,2,2,2]
F	-	G	[0,1,1,1,2,2,2]
$\emptyset$	-	-	[0,1,1,1,2,2,2]

La lista [0,1,1,1,2,2,2] da el número de transbordos que habrá que hacer para conectar el aeropuerto A con el resto de los aeropuertos.

36. El algoritmo continúa funcionando correctamente y, por lo tanto, se obtiene la misma distancia mínima pero el camino puede ser diferente.

37. Se observa que si se aplica el algoritmo de Dijkstra sobre un grafo ponderado, a cada paso se fija la distancia a un vértice, que ya es definitiva. Si se parte del vértice i, en el primer paso se fijará la distancia del vértice $i-1$ o $i+1$, que son los que están a distancia 1 de i. En el paso siguiente, tanto si se elige $i-1$ como $i+1$, se fijará la distancia del vértice $i+2$, y así sucesivamente. Por lo tanto, la distancia entre i y j será el peso de la arista que une i y j: $d(i,j) = |j-i|$.

38. Puesto que a cada paso el algoritmo de Dijkstra fija la distancia a un vértice, que ya no se modificará, será suficiente con modificar el bucle del algoritmo para que se pare cuando se fije la distancia al vértice t:

Entrada : (G,w) de orden n y dos vértices s,t.
Salida : La distancia, *dist*, de s a t.
algoritmo *SinglePairShortestPath*(G,s,t)
  inicio
    $U \leftarrow \emptyset$
    para $v \in V$
        $dist(v) \leftarrow \infty$
    finpara
    $dist(s) \leftarrow 0$
    $i \leftarrow 0$
    mientras $(i < n) \wedge (t \notin U)$
        u_i vértice que logra lo $\min\{dist(v) \mid v \in V - U\}$
        $U \leftarrow U \cup \{u_i\}$
        para $v \in V - U$ adyacente a u_i **si** $dist(u_i) + w(u_i,v) < dist(v)$
            entonces $dist(v) \leftarrow dist(u_i) + w(u_i,v)$
                Se etiqueta v con $(dist(v),u_i)$
        finsi
        finpara
        $i \leftarrow i+1$
    finmientras
    retorno $(dist(t))$
  fin

Este algoritmo es equivalente (tiene la misma complejidad) que el algoritmo de Dijkstra, pero si sólo se necesita buscar la distancia entre dos vértices concretos, tiene un comportamiento mejor que la versión general.

39. En el caso de que G sea un grafo simple (no dirigido) entonces puede utilizarse directamente el algoritmo de Dijkstra para resolver esta variante:

Entrada : (G,w) no dirigido, de orden n y un vértice t.
Salida : La distancia, $d(\cdot)$, de todos los vértices a t.
algoritmo *SingleDestinationShortestPath*(G,t)
  inicio
    retorno (*Dijkstra(G,t)*)
  fin

Si G es un digrafo (grafo dirigido), sólo hace falta crear un nuevo grafo G' con los mismos vértices que G y de manera que sus arcos sean los arcos de G con las orientaciones cambiadas. A continuación, sólo es necesario aplicar el algoritmo de Dijkstra a este nuevo grafo con origen en el vértice t:

Entrada : (G,w) dígrafo, de orden n y un vértice t.
Salida : La distancia, $d(\cdot)$, de todos los vértices a t.
algoritmo *SingleDestinationShortestPath*(G,t)
  inicio
    Generar $G' = (V,A')$ a partir de G
    retorno (*Dijkstra(G',t)*)
  fin

40. La tabla inicial para aplicar el algoritmo de Floyd es:

$$d^0 = \begin{pmatrix} 0 & 3 & 8 & \infty & 4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & 5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Después de aplicar el algoritmo se obtiene la tabla

$$d^5 = \begin{pmatrix} 0 & 3 & 8 & 4 & 4 \\ 3 & 0 & 6 & 1 & 7 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & 5 & 5 & 0 & 6 \\ 8 & 11 & 11 & 6 & 0 \end{pmatrix}$$

donde el valor de la posición (i,j) representa el tiempo mínimo para conectar el nodo i con el j.

Ejercicios de autoevaluación

1. ¿Cuántos caminos de longitud $k = 2, 3, 4$ y 5 entre dos vértices diferentes tiene el grafo K_4? Encontrad una fórmula para calcular el número de caminos de longitud k entre dos vértices diferentes de K_n.

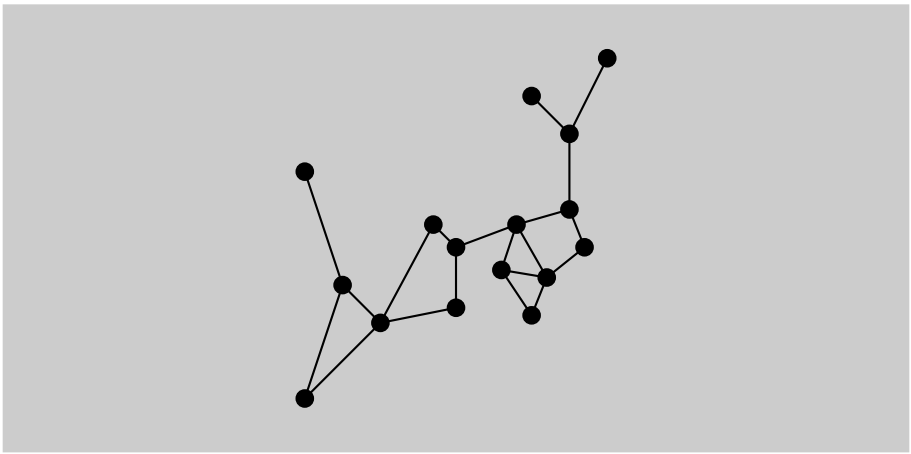
2. El algoritmo DFS utiliza una pila para recordar el orden en el que se han visitado los vértices y poder recuperarlos. La pila se puede sustituir por un llamamiento recursivo. Construid la versión recursiva del algoritmo DFS y comprobad su funcionamiento con el grafo del ejemplo 7.

3. Sea $G = (V,A)$ un grafo de diecisiete vértices con cuatro componentes conexas. Probad que por lo menos una de las componentes tiene un mínimo de cinco vértices.

4. Demostrad que un grafo con secuencia de grados 2, 2, 2, 2, 2 tiene que ser necesariamente conexo.

5. Sea $G = (V,A)$ un grafo de orden $|V| = n \geq 2$ de manera que el grado de cada vértice $v \in V$ satisface $g(v) \geq \frac{1}{2}(n-1)$. Demostrad que G es un grafo conexo. Estudiad si puede afirmarse que son conexos los grafos con las secuencias de grados 2,2,2,2; 2,2,2,2,2; 3,3,3,3,3,3.

6. Una arista a de un grafo conexo $G = (V,A)$ se dice que es una **arista-puente** si el grafo $G - a = (V,A - \{a\})$ es no conexo. Considerad el grafo de la figura siguiente e indicad cuáles son las aristas puente.



7. Sea G un grafo conexo que contenga sólo vértices de grado par. Demostrad que G no puede contener ninguna arista-puente.

8. Un proceso de manufactura empieza con un trozo de madera en bruto. La madera tiene que ser cortada, pulida, agujereada y pintada. Cortar se tiene que hacer antes de agujerear y, pulir, antes de pintar. Considerad que los requisitos de tiempo para cada operación son los siguientes: cortar pide una unidad; pulir, también una; pintar, cuatro, si la madera no está cortada y, dos, si ya lo está; agujerear pide tres unidades, si la madera no está pulida, cinco, si es pulida pero no pintada, y siete, si ya está pintada.

a) ¿Qué orden de realización de procesos hay que seguir para minimizar el tiempo total? ¿Es único?

b) ¿Qué coste tendría que tener agujerear si lo se quiere realizar tras pulir pero antes de pintar, para que el coste continúe siendo mínimo?

9. Una labradora tiene una garrafa de ocho litros llena de vino y dispone sólo de dos garrafas vacías, una de cinco y otra tres litros. ¿Cuál es el número mínimo de pasos que tiene que hacer para repartir el vino en dos partes iguales?

10. Proponed un algoritmo eficiente para calcular el diámetro de un grafo.

11. Se define el **centro** de un grafo G como el vértice tal que la suma de distancias al resto de los vértices sea mínima. Proponed un algoritmo eficiente para calcular el centro de un grafo G.

Soluciones

1. Para $k = 2$ hay dos caminos. Si $k = 3$, también hay 2. Para $k = 4$ y $k = 5$ no hay ninguno.

En general, un camino de longitud k pasará por $k+1$ vértices sin repetir ninguno. Dados dos vértices diferentes, el número de caminos entre ellos vendrá dado por las maneras diferentes de escoger el resto de $k-1$ vértices de entre los $n-2$ vértices restantes (todos excepto los dos vértices extremos fijados). O sea, el número de caminos de longitud k, si $k \leq n-1$, será $V(n-2,k-1)$, el número de $(k-1)$-muestras ordenadas sin repetición del conjunto de los $n-2$ vértices.

Notación

Recordad que el número de r-muestras ordenadas sin repetición de un conjunto X de n elementos se denota $V(n,r)$. Se puede calcular por la fórmula, $V(n,r) = n \cdot (n-1) \cdot (n-2) \cdots (n-(r-1))$.

2. Versión recursiva del algoritmo DFS:

```
Entrada : G = (V,A), v ∈ V
Salida : R, vértices visitados
algoritmo DFS(G,v)
  inicio
    R ← ∅
    para w ∈ V
        estado[w] ← 0
    finpara
    dfsrec(G,v,R,estado)
    retorno (R)
  fin

función dfsrec(G,v,R,estado)
  inicio
    estado[v] ← 1
    añadir(R,v)
    para w adyacente a v
        si estado[w] = 0
          entonces dfsrec(G,w,R,estado)
        finsi
    finpara
  fin
```

3. Sea V el conjunto de vértices y sea $|V| = 17$. Si G_1, G_2, G_3, G_4 son las componentes conexas de G y V_1, V_2, V_3, V_4 los conjuntos de vértices respectivos, entonces se tiene que $\sum_{i=1}^{4} |V_i| = |V| = 17$. Si no se cumpliera la afirmación que hay que demostrar, sería $|V_1| \leq 4$, $|V_2| \leq 4$, $|V_3| \leq 4$, $|V_4| \leq 4$, puesto que el orden de un grafo siempre es un entero. De aquí resulta una contradicción: $17 = |V_1| + |V_2| + |V_3| + |V_4| \leq 4 + 4 + 4 + 4 = 16$.

4. Sea $G = (V,A)$ un grafo de orden $n = 5$ con la secuencia de grados anterior. Si fuese no conexo tendría un mínimo de dos componentes conexas, que se puede suponer que son G_1 y G_2 (y podrían ser más). Se supone que, como grafos que son, sus órdenes son respectivamente n_1 y n_2; se cumple obviamente $n_1 + n_2 \leq n = 5$. Ahora bien, estas componentes conexas son grafos 2-regulares y, por lo tanto, los órdenes respectivos tienen que ser como mínimo 3. En consecuencia $6 = 3 + 3 \leq n_1 + n_2 \leq n = 5$, cosa que es absurda. Esto prueba que el grafo es conexo.

5. Se supone que G es no conexo y sean, por lo tanto, $C_1,\ldots,C_k$, $k \geq 2$, las componentes conexas de G, con conjuntos de vértices respectivos $V_1,\ldots,V_k$, y sea $q_i = |V_i|$, para $i = 1,\ldots,k$. Evidentemente es $q_i + \ldots + q_k = n$.

Se consideran dos vértices u,w que pertenezcan a dos componentes conexas diferentes. Se supone, por ejemplo, que $u \in C_i$, $w \in C_j$, $i \neq j$. Entonces u puede ser adyacente como mucho a los $q_i - 1$ vértices restantes a la componente conexa a la que pertenece, es decir, $g(u) \leq q_i - 1$, y análogamente para w. Teniendo en cuenta esto y aplicando la hipótesis, se puede afirmar:

$$\frac{n-1}{2} \leq g(u) \leq q_i - 1, \quad \frac{n-1}{2} \leq g(w) \leq q_j - 1$$

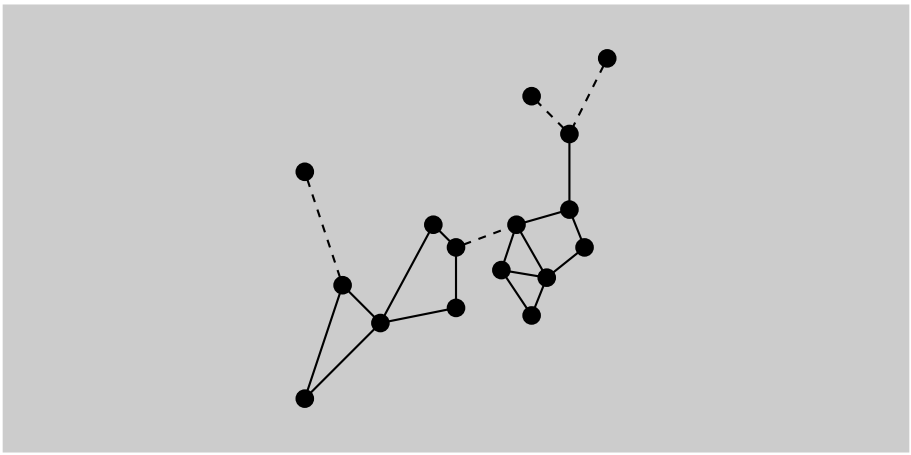
de aquí se puede obtener obtener inmediatamente

$$n - 1 = \frac{n-1}{2} + \frac{n-1}{2} \leq g(u) + g(w) \leq q_i + q_j - 2 \leq n - 2$$

que es una conclusión absurda. Por lo tanto, el grafo tiene que ser conexo.

Con respecto a los grafos con las secuencias indicadas, tienen que ser conexos en aplicación del resultado. Buscad ejemplos de grafos con estas secuencias de grados.

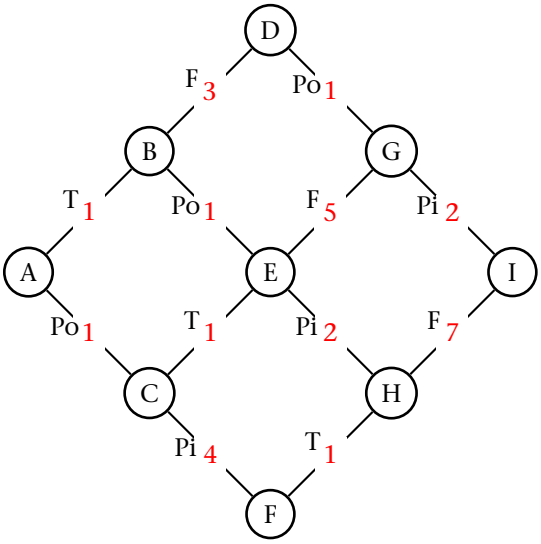
6. En la siguiente figura se indican las aristas-puente con línea discontinua.



7. Se supone que $G = (V,A)$ es conexo y sea $a = \{v,w\}$ una arista-puente. Se considera el grafo auxiliar que resulta de la eliminación de la arista anterior, es decir, $G' = G - a$.

Si en el grafo G se cumplía $g_G(v) = 2k \geq 2$ y $g_G(w) = 2k' \geq 2$, en el nuevo grafo G' se cumplirá $g_{G'}(v) = 2k-1$ y $g_{G'}(w) = 2k'-1$, grados impares, y sin que el resto de los grados varíe. Siendo a arista-puente y G grafo conexo, el grafo $G' = G_1 \cup G_2$ es reunión de dos componentes conexas y los vértices extremos de la arista a se han distribuido, uno en cada uno de estas componentes conexas. Esto conduce a una contradicción. En efecto, si por ejemplo $v \in V(G_1)$, entonces, considerado como grafo, sería un grafo con un vértice de grado impar y el resto de grado par, con lo cual habría un número impar de vértices de grado impar, lo que es imposible, como consecuencia de la fórmula de los grados.

8. El grafo siguiente muestra las diferentes situaciones que se dan en el problema:

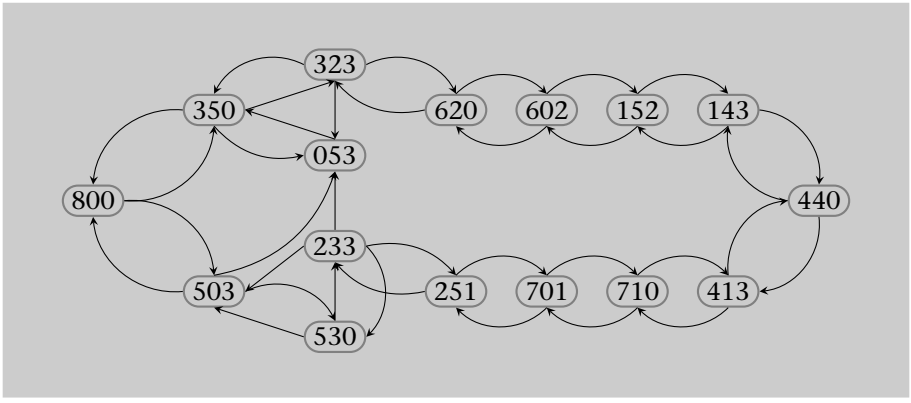


Las aristas representan cada uno de los procesos que se tienen que hacer y su coste; los vértices son los estados finales de estos procesos.

a) Siguiendo el algoritmo de Dijkstra, podéis comprobar de manera sencilla que el recorrido más barato desde A hasta I es ABDGI. El coste será de 7.

b) El coste actual de agujerear tras pulir, antes de pintar, es 5. El coste total de hacer todo el proceso de este modo es 9. Puesto que el coste mínimo es 7, se tendría que reducir en 2 unidades el coste de agujerear tras pulir, antes de pintar, para igualar el recorrido mínimo. Por lo tanto, el coste tiene que ser 3.

9. El digrafo siguiente representa las diferentes transiciones entre las garrafas. La etiqueta de cada vértice representa el contenido de cada una de la tres garrafas:



Se debe notar que no se han representado todos los estados, sino sólo aquellos más relevantes.

Dado que únicamente se quiere conocer el número mínimo de transiciones necesarias para repartir el vino, se trata de un problema de búsqueda del camino mínimo en un grafo no ponderado. Aplicando el algoritmo, se obtiene que el número mínimo de pasos es 7 y la secuencia de transiciones es: $800 \rightarrow 350 \rightarrow 323 \rightarrow 620 \rightarrow 602 \rightarrow 152 \rightarrow 143 \rightarrow 440$.

10. Tanto para calcular el diámetro como el centro de un grafo G se necesita disponer de las distancias entre todos los pares de vértices del grafo. Por lo tanto, se utilizará el algoritmo de Floyd para calcularlas.

```
Entrada : (G,w) grafo ponderado de orden n.
Salida : El diámetro, D, del grafo.
algoritmo Diámetro(G)
  inicio
    d ← Floyd(G);
    D ← 0
    para i ← 1 hasta n
        para j ← 1 hasta n
            si d(i,j) > D
              entonces D ← d(i,j)
            finsi
        finpara
    finpara
    retorno (D)
  fin
```

Para estudiar la eficiencia de este algoritmo, se debe observar que básicamente se aplica el algoritmo de Floyd (que ya se sabe que tiene una complejidad $O(n^3)$) y se busca el valor máximo de una tabla de n^2 valores, que tiene una complejidad $O(n^2)$. En total tendrá una complejidad $O(n^3)$, comparable con la del algoritmo de Floyd.

11. Para calcular el centro del grafo se necesita, además de las distancias entre todos los pares de vértices, la suma de distancias de cada vértice al resto:

```
Entrada : (G,w) grafo ponderado de orden n.
Salida : El centro, C, del grafo.
algoritmo Centro(G)
  inicio
    d ← Floyd(G);
    C ← 1
    Dmin ← ∑_{j=1}^{n} d(1,j)
    para i ← 2 hasta n
        D ← ∑_{j=1}^{n} d(i,j)
        si D < Dmin
          entonces Dmin ← D
                   C ← i
        finsi
    finpara
  finpara
  retorno (C)
fin
```

Para calcular la eficiencia se debe recordar que se calculan las distancias entre todos los pares de vértices utilizando el algoritmo de Floyd (con una complejidad $O(n^3)$) y se debe calcular la suma de todas las filas de la matriz d. Esto se hace con una complejidad $O(n^2)$. En total tendrá una complejidad $O(n^3)$ comparable con el algoritmo de Floyd.

Bibliografía

Biggs, N. L. (1994). *Matemática Discreta*. (1a. edición, traducción de M. Noy). Barcelona: Ediciones Vicens Vives.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001). *Introduction to Algorithms*. Cambridge: MIT Press.