Grafos

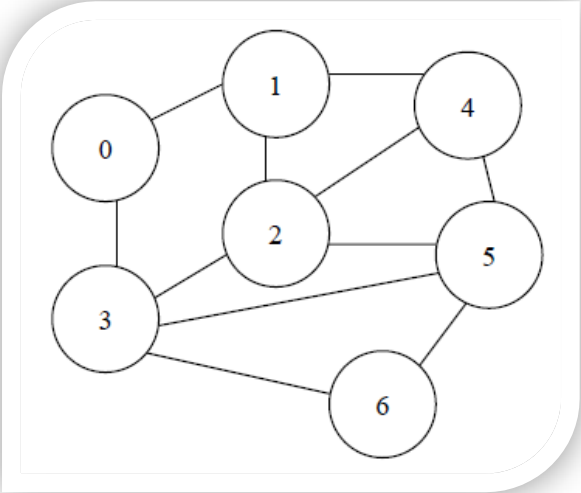
Definiciones Generales

Definición de grafos

- En términos informales, un grafo es un conjunto finito de nodos o vértices que pueden o no estar conectados por líneas a las que denominamos aristas.

- Cada vértice del grafo se identifica con una etiqueta numérica o alfanumérica que lo diferencia de todos los demás.

- Por ejemplo, en el grafo de la figura siguiente, los vértices están etiquetados con valores numéricos consecutivos, comenzando desde cero.



Grafos conexos y no conexos

- Si, partiendo desde un vértice y desplazándonos a través de las aristas podemos llegar a cualquier otro vértice del grafo decimos que el grafo es **“conexo”.**

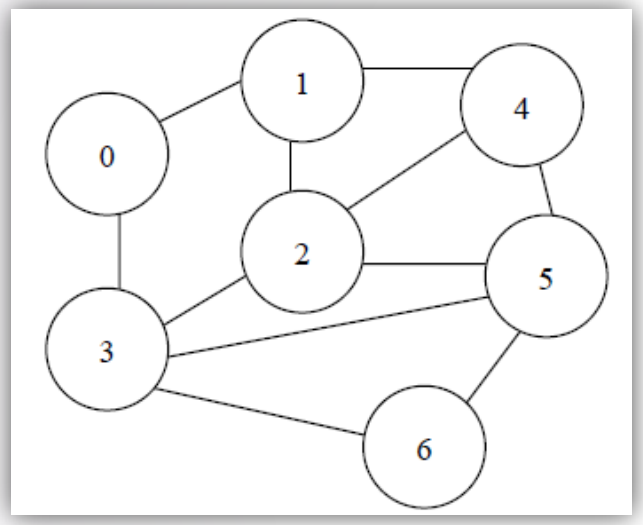
- En cambio, si existe al menos un vértice al que no se puede llegar será porque está desconectado. En este caso, diremos que el grafo es **“no conexo”.**

Grafos dirigidos y no dirigidos

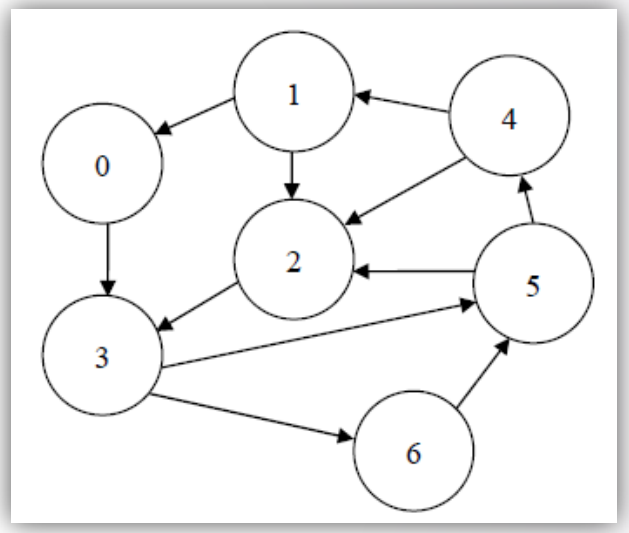
- Cuando las aristas están representadas con flechas, que indican una determinada dirección, decimos que se trata de un “dígrafo” o **“grafo dirigido”.**

- En cambio, si las aristas no especifican un sentido en particular tendremos un **“grafo no dirigido”.**

Grafo no dirigido



Grafo dirigido

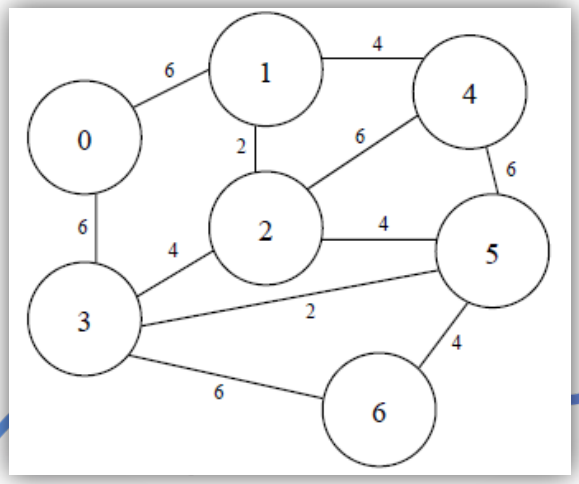


Grafos ponderados

- Sobre las aristas de un grafo, dirigido o no, se puede indicar un determinado valor para representar, por ejemplo, un costo, una ponderación o una distancia.

- En este caso, decimos que el grafo está ponderado.

- Por ejemplo, si los nodos del grafo representan ciudades entonces los valores de las aristas podrían representar las distancias que existen entre estas.
- O también podrían representar el costo que implicaría trasladarnos desde una ciudad hacia otra.



Camino entre dos vértices

- Sean **a** y **b** dos vértices de un grafo, diremos que el camino entre **a** y **b** es el trayecto o conjunto de aristas que debemos recorrer para llegar desde **a** hasta **b.**

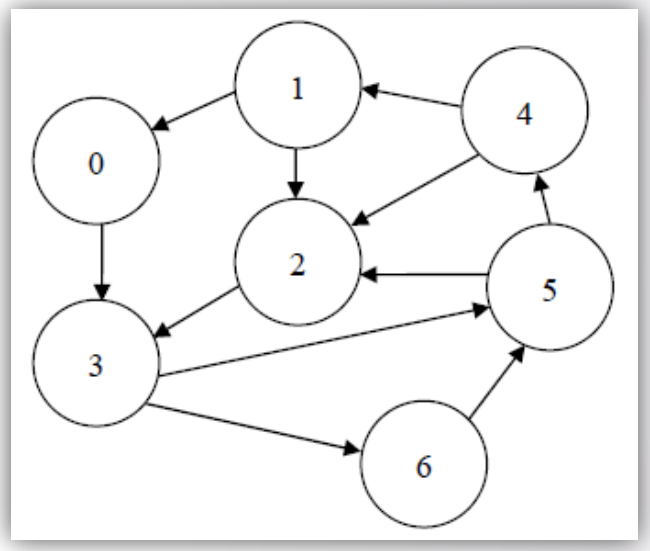
- Si el grafo tiene ciclos entonces es probable que exista más de un camino posible para unirlos.

- En un grafo dirigido debemos respetar la dirección de las aristas. Así, en el dígrafo el camino que une los vértices 5 y 0 es único.

- Partiendo desde 5 llegamos hasta 0 pasando por los siguientes vértices: c = {5, 4, 1, 0}.

- Este camino también podemos expresarlo como el conjunto de aristas por el que nos tendremos que desplazar:

$$c = \{(5,4),\ (4,1),\ (1,0)\}$$



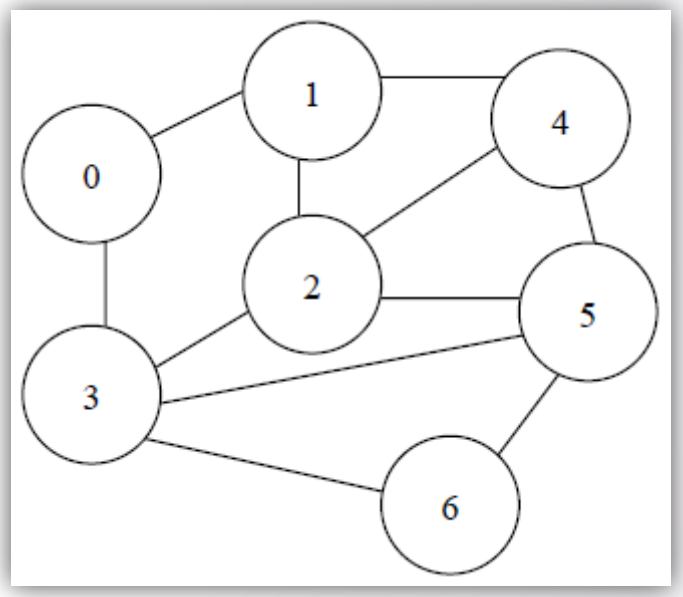
- En el caso del grafo no dirigido que vemos en la parte izquierda de la figura, podemos encontrar varios caminos que nos permiten unir los vértices 5 y 0.

- Como las aristas no especifican ninguna dirección, entonces, podemos transitarlas en cualquier sentido. Algunos de estos caminos son:

c1 = {5, 2, 1, 0}

c2 = {5, 2, 3, 0}

c3 = {5, 3, 0}



Ciclos

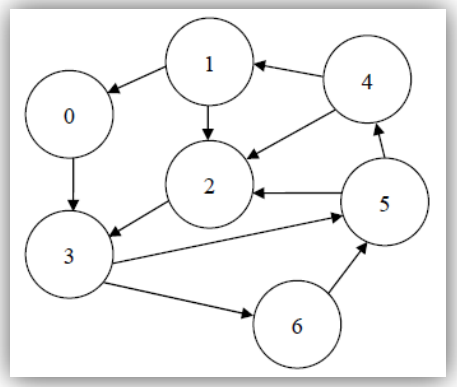
- Llamamos ciclo a un camino que, sin pasar dos veces por la misma arista, comienza y finaliza en el mismo vértice.
- Por ejemplo, en el dígrafo de la figura anterior algunos de los ciclos que podemos identificar son:

c1 = {0, 3, 6, 5, 4, 1, 0}

c2 = {0, 3, 5, 4, 1, 0}

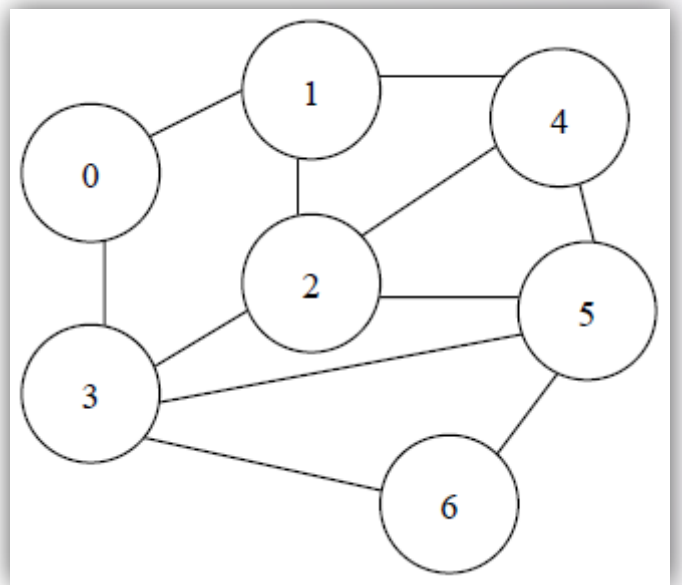
c3 = {3, 6, 5, 2, 3}

c4 = {5, 4, 1, 2, 3, 5}



- Y en el grafo no dirigido ubicado a la izquierda podemos destacar los siguientes ciclos:

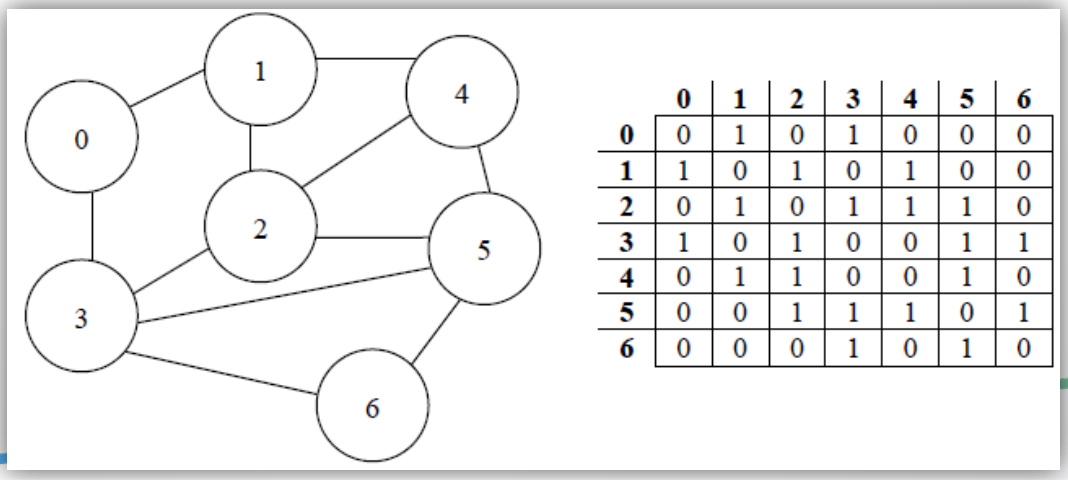
  c1 = {0, 1, 4, 2, 3, 0}
  c2 = {2, 1, 0, 3, 2}
  c3 = {6, 5, 4, 2, 1, 0, 3, 6}



Vértices adyacentes y matriz de adyacencias

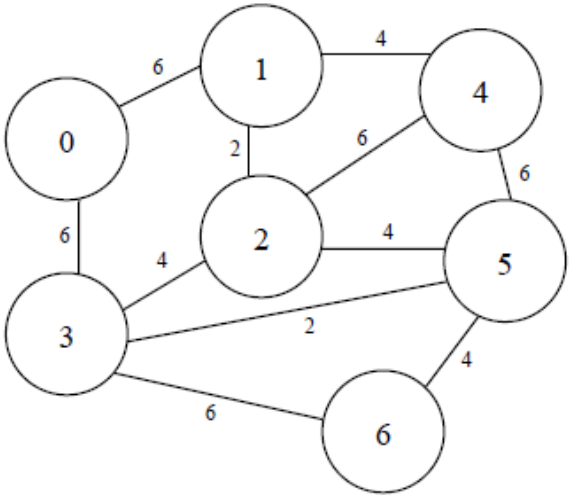
- Cuando dos vértices están unidos por una arista decimos que son “vértices adyacentes”.
- Luego, utilizando una matriz, podemos representar las adyacencias entre los vértices de un grafo dirigido, no dirigido, ponderado o no ponderado.
- La matriz de adyacencias es una matriz cuadrada, con tantas filas y columnas como vértices tenga el grafo que representa.
- Cada celda que se origina en la intersección de una fila i con una columna j representa la relación de adyacencia existente entre los vértices homólogos.

- Para representar un grafo no dirigido ni ponderado, utilizaremos una matriz booleana.
- Las celdas de esta matriz tendrán el valor 1 (o true) para indicar que los vértices representados por la intersección fila/columna son adyacentes o 0 (o false) para indicar que ese par de vértices no lo son.



	0	1	2	3	4	5	6
0	0	1	0	1	0	0	0
1	1	0	1	0	1	0	0
2	0	1	0	1	1	1	0
3	1	0	1	0	0	1	1
4	0	1	1	0	0	1	0
5	0	0	1	1	1	0	1
6	0	0	0	1	0	1	0

- La matriz de adyacencias de un grafo no dirigido es simétrica ya que, para todo par de vértices **a** y **b**, si se verifica que **a** es adyacente con **b** entonces **b** será adyacente con **a.**
- Veamos ahora la matriz de adyacencias de un grafo no dirigido y ponderado.



	0	1	2	3	4	5	6
0	0	6	0	6	0	0	0
1	6	0	2	0	4	0	0
2	0	2	0	4	6	4	0
3	6	0	4	0	0	2	6
4	0	4	6	0	0	6	0
5	0	0	4	2	6	0	4
6	0	0	0	6	0	4	0

- En este caso, la relación de adyacencia entre dos vértices **a** y **b** se ve reflejada cuando en la celda que surge como intersección de la fila **a** y la columna **b** existe un valor mayor que cero, que coincide con el valor de la arista que los une.

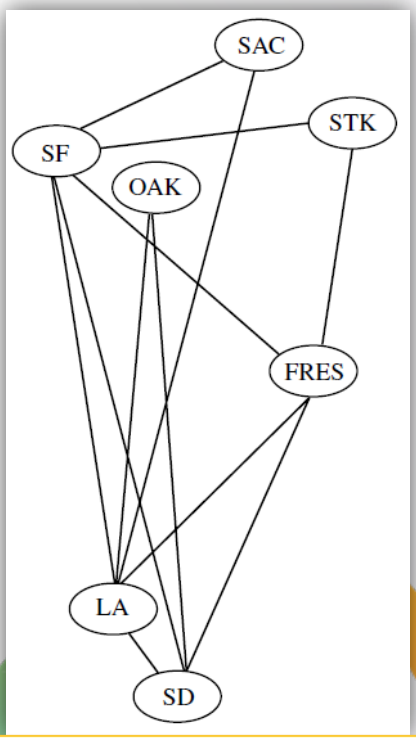
- La no adyacencia de **a** con **b** puede representarse con el valor **0** en la celda intersección de la fila **a** con la columna **b** o con el valor “infinito”.

Ejemplos

Mapa de rutas de aerolíneas

- Un mapa de las rutas de una aerolínea se puede representar con un grafo no dirigido.

- Los puntos son las ciudades; una línea conecta dos ciudades si y sólo si hay un vuelo sin escalas entre ellas en ambas direcciones.

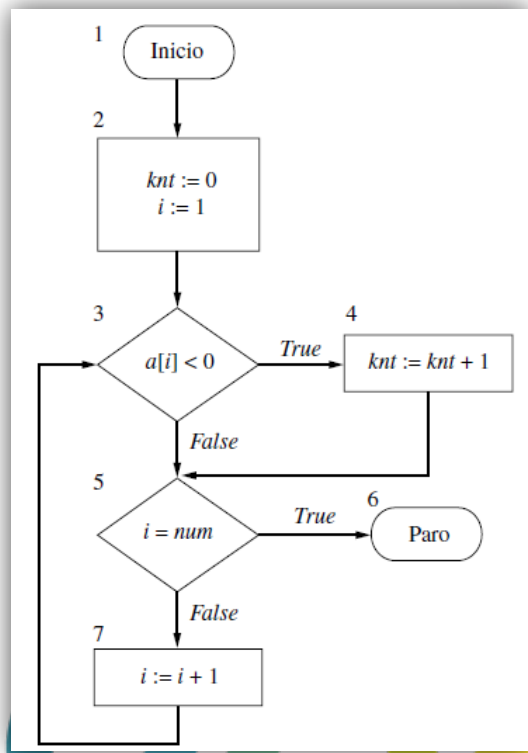


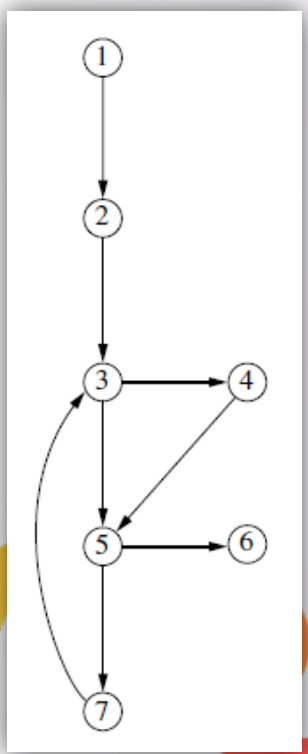
Grafo hipotético de vuelos sin escalas entre ciudades de California

Diagramas de flujo

- Un diagrama de flujo representa el flujo de control en un procedimiento, o el flujo de datos o materiales en un proceso.

- Los puntos son los rectángulos del diagrama de flujo; las flechas que los conectan son las flechas del diagrama de flujo.

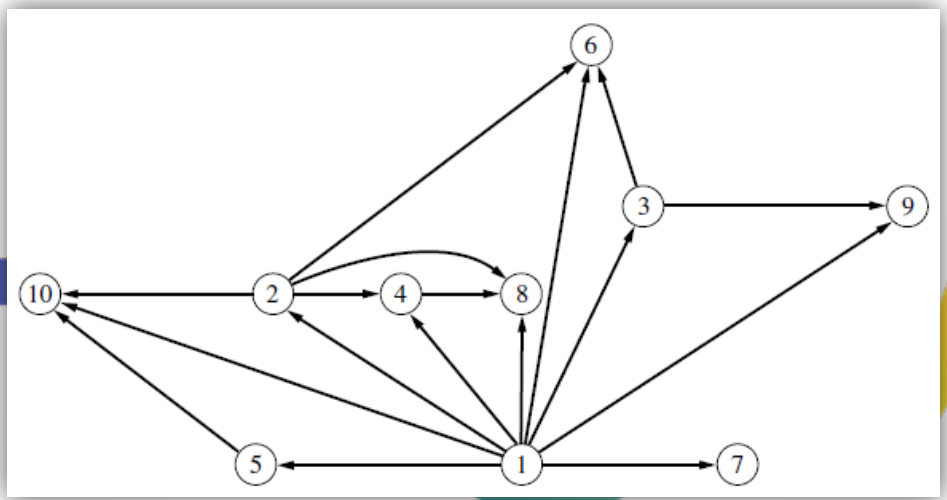




Una relación binaria

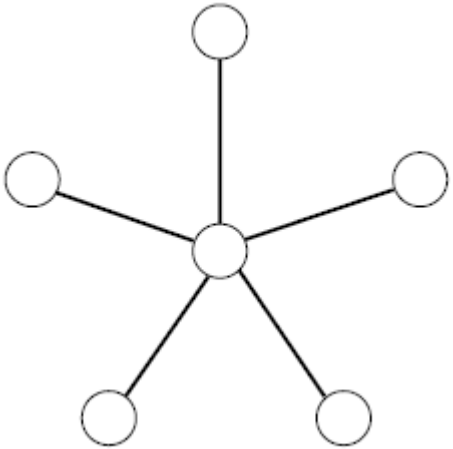
- Definimos R como la relación binaria sobre el conjunto **S = {1, . . . , 10}** que consiste en pares ordenados **(x, y)** en los que **x** es un factor propio de **y**; es decir, **x ≠ y** y el residuo de **y/x** es **0**.

- Recordemos que **xRy** es una notación alterna para **(x, y) ∈ R**.

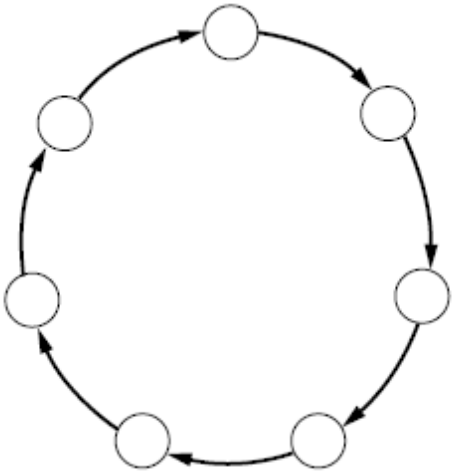


Redes de computadoras

- Los puntos son las computadoras.
- Las líneas (si el grafo es no dirigido) o las flechas (si el grafo es dirigido) son los enlaces de comunicación.



Red de estrella

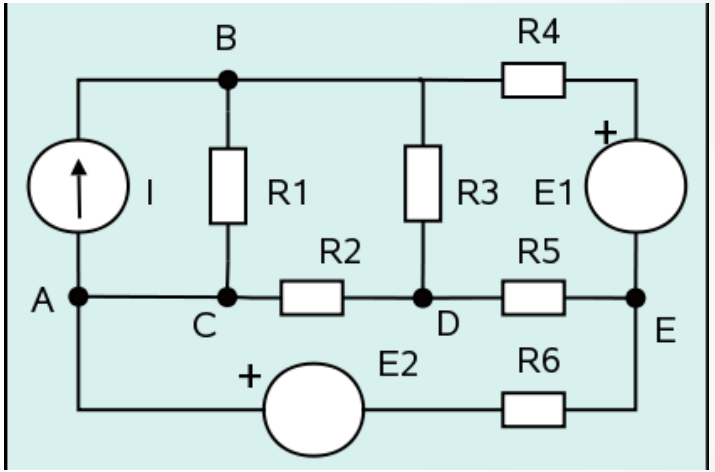


Red de anillo

Un circuito eléctrico

- Los puntos podrían ser diodos, transistores, condensadores, interruptores, etc.

- Dos puntos están conectados por una línea si hay una conexión eléctrica.



Definiciones Formales

Grafo dirigido

- Un grafo dirigido, o dígrafo, es un par **G = (V, E)** donde **V** es un conjunto cuyos elementos se llaman vértices y **E** es un conjunto de pares ordenados de elementos de **V**.
- Los vértices también suelen llamarse nodos.
- Los elementos de **E** se llaman aristas, o aristas dirigidas, o arcos.
- Para la arista dirigida ***(v, w)*** en **E**:
  - **v** es su cola
  - **w** es su cabeza;
  - **(v, w)** se representa en los diagramas con la flecha **v → w**.

$V = \{1, 2, \ldots, 10\},$

$E = \{(1,2), \ldots, (1,10), (2,4), (2,6), (2,8), (2,10), (3,6), (3,9), (4,8), (5,10)\}.$

Para el ejemplo de una relación binaria

Grafo no dirigido

- Un grafo no dirigido es un par ***G = (V, E)*** donde **V** es un conjunto cuyos elementos se llaman vértices y **E** es un conjunto de pares no ordenados de elementos distintos de **V**.
- Los vértices también suelen llamarse nodos.
- Los elementos de **E** se llaman aristas, o aristas no dirigidas.
- Cada arista se puede considerar como un subconjunto de **V** que contiene dos elementos; así, **{v, w}** denota una arista no dirigida.
- En los diagramas esta arista es la línea **v-w**.

$$V = \{\text{SF, OAK, SAC, STK, FRES, LA, SD}\},$$

$$E = \left\{ \begin{array}{llllll} \{\text{SF, STK}\}, & \{\text{SF, SAC}\}, & \{\text{SF, LA}\}, & \{\text{SF, SD}\}, & \{\text{SF, FRES}\}, & \{\text{SD, OAK}\}, \\ \{\text{SAC, LA}\}, & \{\text{LA, OAK}\}, & \{\text{LA, FRES}\}, & \{\text{LA, SD}\}, & \{\text{FRES, STK}\}, & \{\text{SD, FRES}\} \end{array} \right\}.$$

- Subgrafo

Un *subgrafo* de un grafo $G = (V, E)$ es un grafo $G' = (V', E')$ tal que $V' \subseteq V$ y $E' \subseteq E$. Por la definición de "grafo", también es obligatorio que $E' \subseteq V \times V'$.

- Grafo dirigido simétrico

Un *grafo dirigido simétrico* es un grafo dirigido tal que, por cada arista vw existe también la arista inversa wv. Todo grafo no dirigido tiene un grafo dirigido simétrico correspondiente si se interpreta cada arista no dirigida como un par de aristas dirigidas en direcciones opuestas.

- Grafo completo

Un *grafo completo* es un grafo (normalmente no dirigido) que tiene una arista entre cada par de vértices.

Decimos que la arista vw *incide* en los vértices v y w, y viceversa. ■

Relación de adyacencia

Las aristas de un grafo dirigido o no dirigido $G = (V, E)$ inducen una relación llamada *relación de adyacencia*, A, sobre el conjunto de vértices. Sean v y w elementos de V. Entonces vAw (que se lee "w está *adyacente* a v") si y sólo si vw está en E. En otras palabras, vAw implica que es posible llegar a w desde v desplazándose a lo largo de una arista de G. Si G es un grafo no dirigido, la relación A es simétrica. (Es decir, wAv si y sólo si vAw.) ■

Camino en un grafo

Un *camino de* v *a* w en un grafo $G = (V, E)$ es una sucesión de aristas $v_0v_1, v_1v_2, \ldots, v_{k-1}v_k$, tal que $v = v_0$ y $v_k = w$. La longitud del camino es k. Un vértice v sólo se considera un camino de longitud cero de v a v. Un *camino simple* es un camino tal que $v_0, v_1, \ldots, v_k$ son todos distintos.

Decimos que un vértice w es *asequible* desde v si existe un camino de v a w. ■

Conectado, fuertemente conectado

Las definiciones de *conectividad* requieren atención porque difieren entre los grafos dirigidos y no dirigidos.

Un grafo no dirigido está *conectado* si y sólo si, para cada par de vértices v y w, existe un camino de v a w.

Un grafo dirigido está *fuertemente conectado* si y sólo si, para cada par de vértices v y w, existe un camino de v a w. ■

Ciclo en un grafo

En un grafo dirigido, un *ciclo* no es más que un camino no vacío tal que el primer vértice y el último sean el mismo, y un *ciclo simple* es un ciclo en el que ningún vértice se repite, con la salvedad de que el primero y el último son idénticos.

Un grafo es *ácíclico* si no tiene ciclos.

Un grafo acíclico *no dirigido* se denomina *bosque no dirigido*. Si el grafo también está conectado, es un *árbol libre* o *árbol no dirigido*.

Grafo ponderado

Un *grafo ponderado* es una tripleta (V, E, W) en la que (V, E) es un grafo (dirigido o no dirigido) y W es una función de E sobre $\mathbf{R}$, los reales. (En algunos problemas podría ser apropiado que los pesos fueran de otro tipo, como números racionales o enteros.) Para una arista e, $W(e)$ es el *peso* de e. ■

Representaciones matriciales y basadas en listas

Representación de matriz de adyacencias

- Sea G (V, E) un grafo con n = |V|, m = |E| y V = {$v_1, v_2, \ldots, v_n$}.

- Podemos representar **G** con una matriz A = (a_{ij}) de n×n elementos, llamada matriz de adyacencia de G.
- A está definida por:

$$a_{ij} = \begin{cases} 1 & \text{si } v_i v_j \in E \\ 0 & \text{en los demás casos} \end{cases} \qquad \text{para } 1 \leq i, j \leq n.$$

- La matriz de adyacencia de un grafo no dirigido es simétrica (y sólo es preciso almacenar la mitad).

- Si **G = (V, E, W)** es un grafo ponderado, los pesos se podrán almacenar en la matriz de adyacencia modificando su definición como sigue:

$$a_{ij} = \begin{cases} W(v_i v_j) & \text{si } v_i v_j \in E \\ c & \text{en los demás casos} \end{cases} \qquad \text{para } 1 \leq i, j \leq n.$$

Donde **c** es una constante cuyo valor depende de la interpretación de los pesos y del problema a resolver.
Si los pesos se ven como costos, se podría escoger ∞ (o algún número muy alto) para **c** porque el costo de recorrer una arista inexistente es prohibitivamente alto.

Si los pesos son capacidades, suele ser apropiado escoger **c = 0** porque nada puede viajar por una arista que no existe.

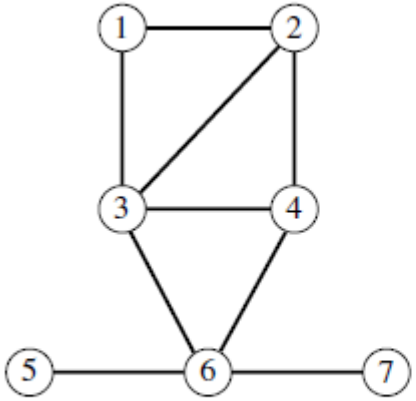
- Si se usa una representación de matriz de adyacencia, bien podríamos imaginar que un grafo tiene aristas entre todos los pares de vértices distintos, porque muchos algoritmos examinarían cada elemento de la matriz para determinar cuáles aristas existen realmente.

- Puesto que el número de posibles aristas es $\mathbf{n^2}$ en un grafo dirigido, o de $\mathbf{n(n-1)/2}$ en un grafo no dirigido, la complejidad de tales algoritmos estará en $\mathbf{\Omega(n^2)}$.

Representación con arreglo de listas de adyacencia

- Una alternativa respecto a la representación con matriz de adyacencia es un arreglo indizado por número de vértice que contiene listas ligadas llamadas listas de adyacencia.
- Por cada vértice **i**, el i-ésimo elemento del arreglo contiene una lista con información acerca de todas las aristas de **G** que “salen de” **i**.
- En un grafo dirigido esto implica que **i** es la cola de la arista.
- En un grafo no dirigido, la arista incide en **i**.
- La lista de **i** contiene un elemento por arista.

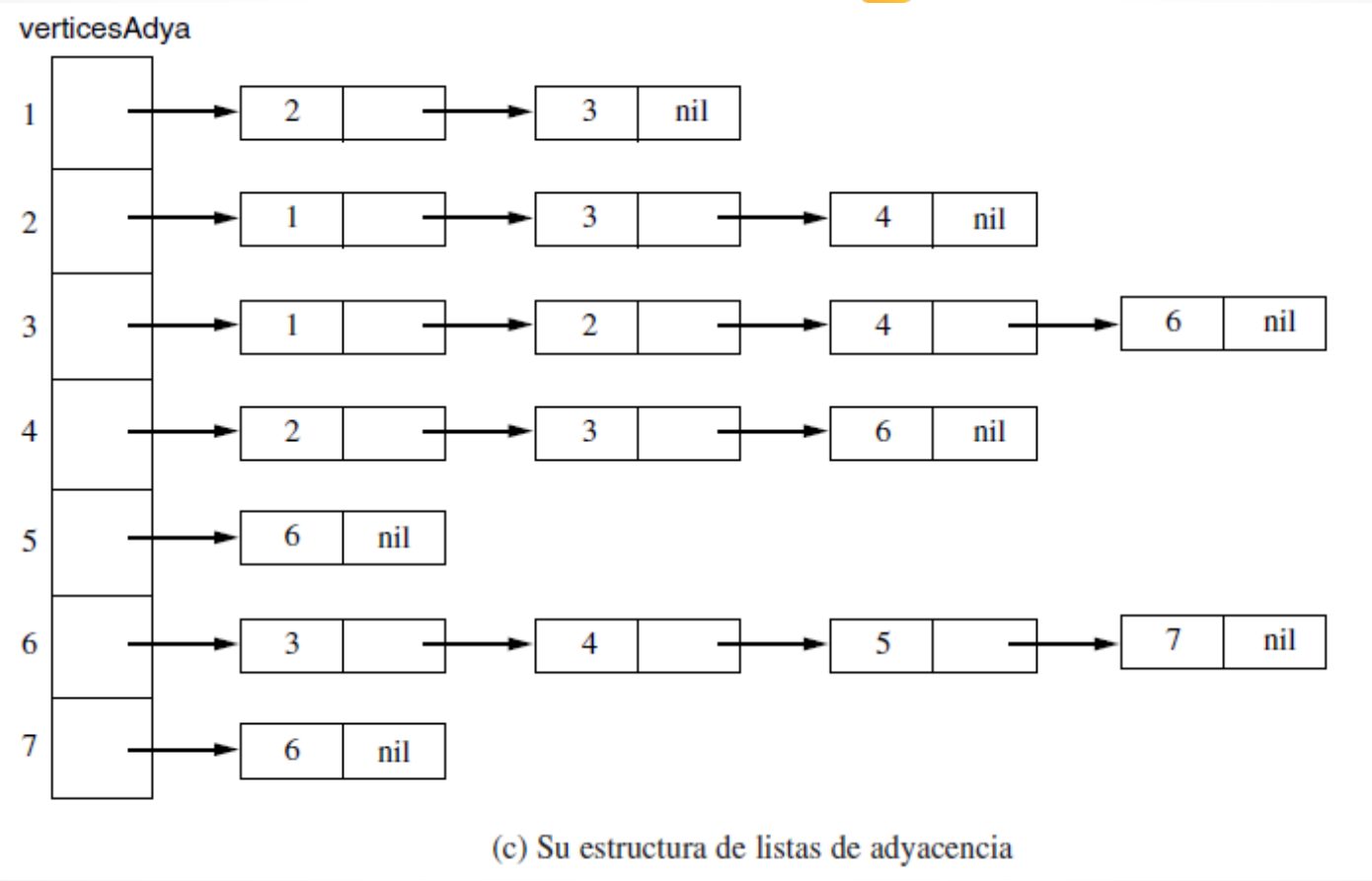
Grafo no dirigido



(a) Un grafo no dirigido

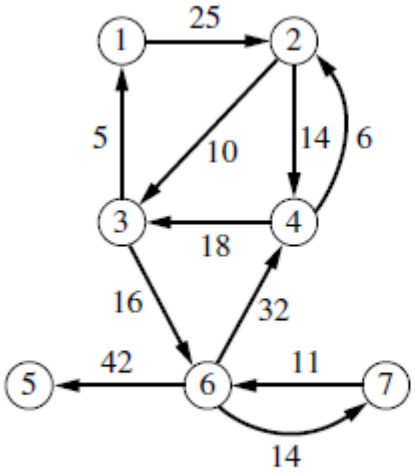
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(b) Su matriz de adyacencia



(c) Su estructura de listas de adyacencia

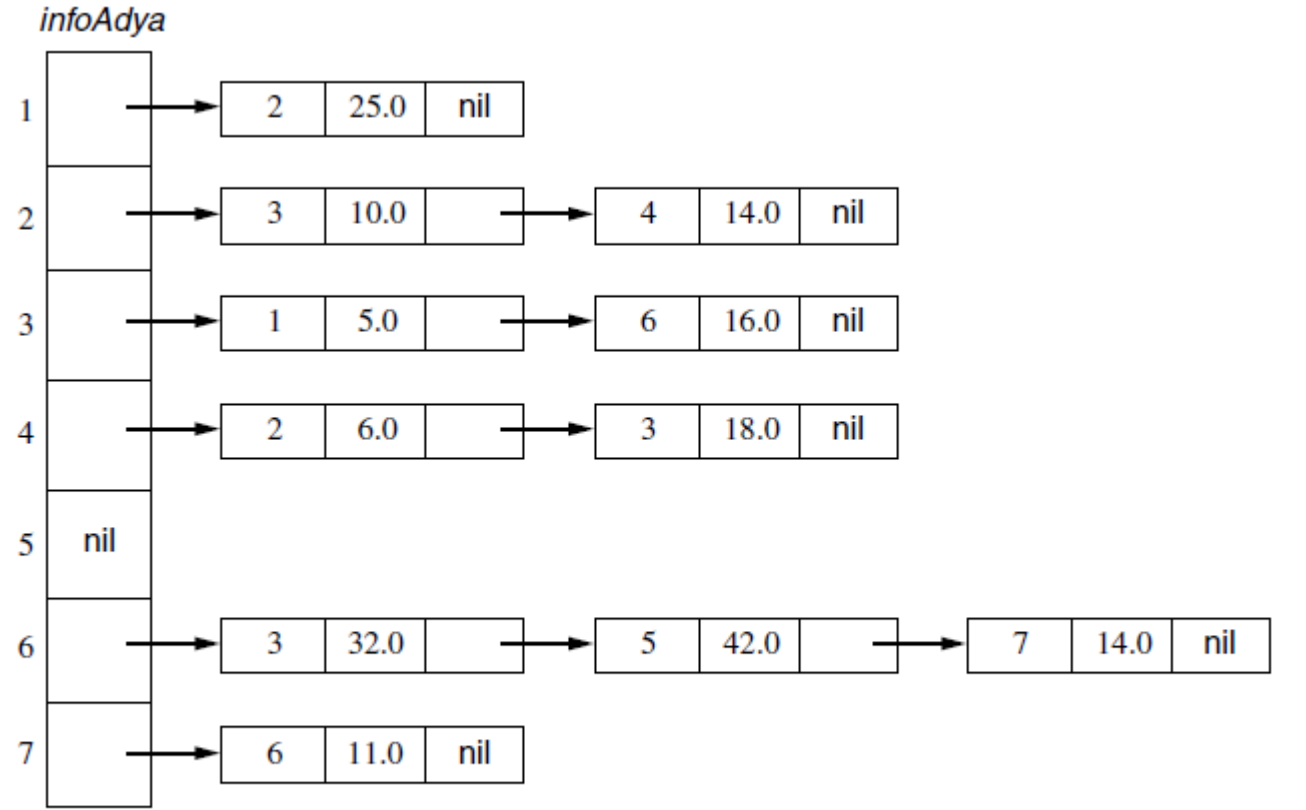
Grafo dirigido ponderado



(a) Un grafo dirigido ponderado

$$\begin{pmatrix} 0 & 25.0 & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 10.0 & 14.0 & \infty & \infty & \infty \\ 5.0 & \infty & 0 & \infty & 16.0 & \infty & \infty \\ \infty & 6.0 & 18.0 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 32.0 & 42.0 & 0 & 14.0 \\ \infty & \infty & \infty & \infty & \infty & 11.0 & 0 \end{pmatrix}$$

(b) Su matriz de adyacencia



(c) Su estructura de listas de adyacencia

Recorrido de grafos

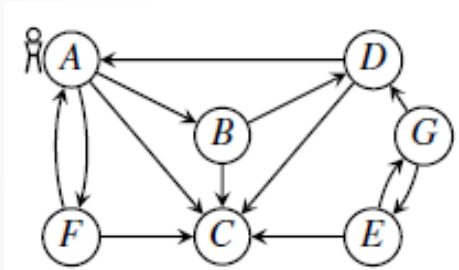
Recorrido de grafos

- Casi todos los algoritmos para resolver problemas con un grafo examinan o procesan cada vértice y cada arista.
- La búsqueda primero en amplitud y la búsqueda primero en profundidad son dos estrategias de recorrido que permiten “visitar” de forma eficiente cada vértice y arista exactamente una vez.
- Por consiguiente, muchos algoritmos basados en tales estrategias se ejecutan en un tiempo que crece linealmente al crecer el tamaño del grafo de entrada.

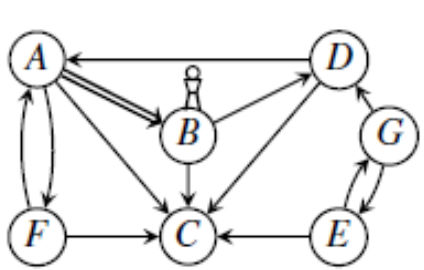
Búsqueda primero en profundidad

- La búsqueda primero en profundidad es una generalización del recorrido general de un árbol.
- El vértice inicial podría depender del problema o escogerse arbitrariamente.
- Al igual que con el recorrido de un árbol, resulta útil visualizar la búsqueda primero en profundidad como un viaje alrededor del grafo.
- La analogía con el recorrido de árboles se ve más claramente en el caso de los grafos dirigidos porque las aristas tienen una dirección, igual que las aristas de los árboles.

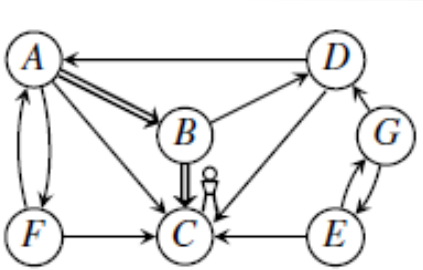
- Iniciemos una búsqueda primero en profundidad en el vértice A del grafo siguiente.
- Por sencillez, cuando podamos escoger qué arista explorar, las escogeremos en orden alfabético.



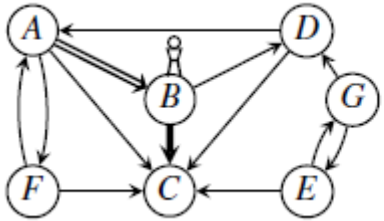
Inicia en A



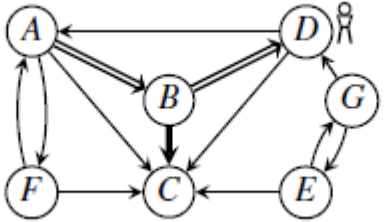
Explora a B



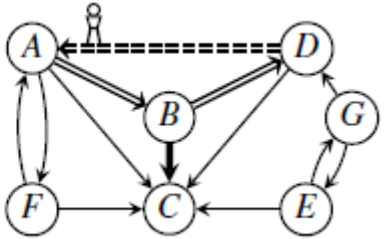
Explora a C
Ya no tiene a donde explorar



Retroceso a B



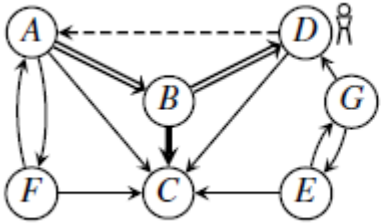
Explora a D



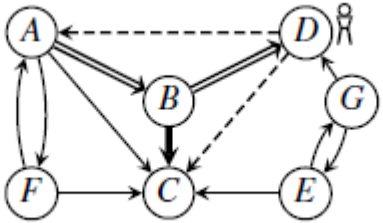
Explora a A
Se forma un ciclo

Al buscar en un grafo, es necesario recordar dónde hemos estado

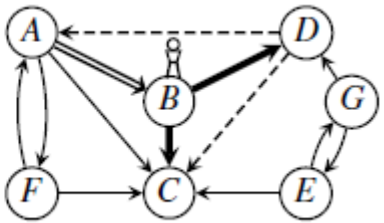
Líneas remarcadas en negro son retrocesos



Retroceso a D

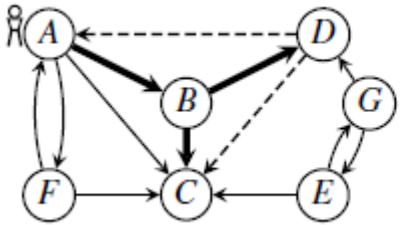


Visita C y retrocede a D ya que C es un nodo visitado anteriormente

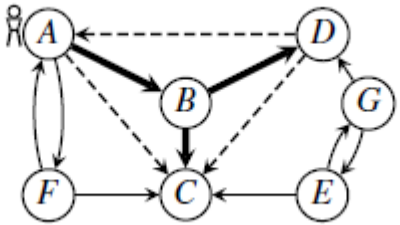


Visita B y retrocede a D ya que B es un nodo visitado anteriormente

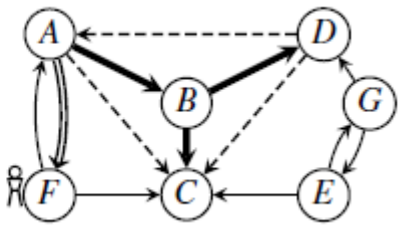
Líneas punteadas son cuando llega a nodos visitados anteriormente



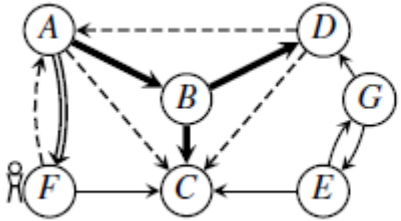
Retroceso a B y luego retroceso a A



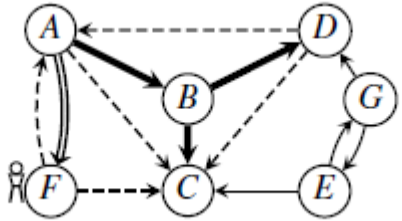
Recorre a C pero es un nodo visitado anteriormente y se regresa a A



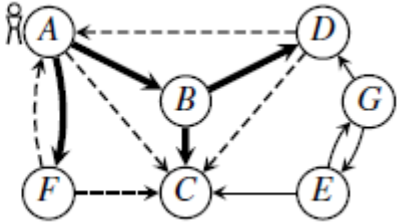
Explora a F



Recorre a A y como ya fue visitado, retrocede a F



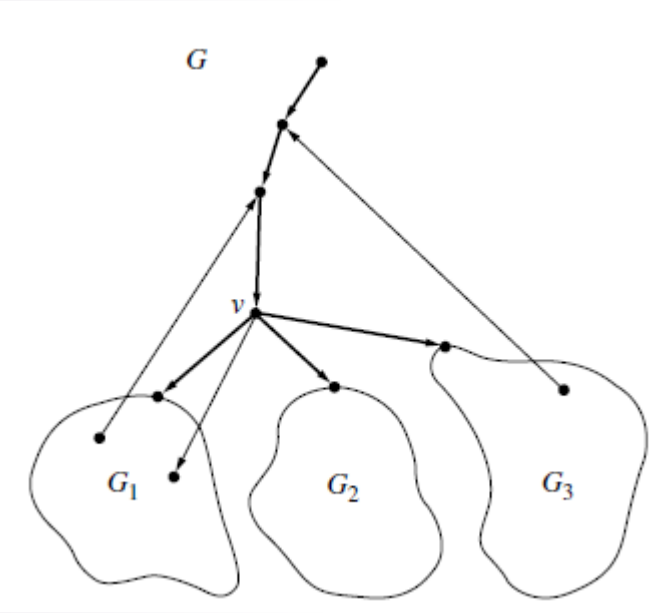
Recorre a C y como ya fue visitado, retrocede a F



Retroceso a A

Observemos que nunca pudo llegar a *E ni a G.*

- Si examinamos el último diagrama veremos que las aristas dibujadas con líneas gruesas continuas, que condujeron a vértices no descubiertos durante la búsqueda, forman un árbol.
- Esto es lógico si nos ponemos a pensar en ello, pues un vértice sólo se puede descubrir una vez, así que sólo puede llegar a él una arista de este tipo.
- Que sólo una arista llegue a cada vértice es una propiedad de los árboles.
- El árbol definido por las aristas que llevaron a vértices no descubiertos durante la búsqueda se denomina *árbol de búsqueda primero en profundidad, o árbol DFS (por sus siglas en inglés).*



Estructura de la búsqueda primero en profundidad

Se recorre G_1 totalmente antes de explorar G_2, luego G_3.

Puesto que G podría no ser un árbol, podría haber aristas que van de los subgrafos a vértices que ya se visitaron antes.

Supóngase que los vértices a los que se llegará desde v durante una búsqueda primero en profundidad se pueden dividir en varios subgrafos, G_1, G_2, G_3, tales que no existe conexión (a través de vértices no descubiertos) entre G_1, G_2 y G_3. También supondremos para este ejemplo que la lista de adyacencia de v está organizada de tal manera que algún vértice de G_1 se descubre antes que cualquier vértice de G_2, y algún vértice de G_2 se descubre antes que cualquier vértice de G_3.

La estrategia primero en profundidad de siempre explorar un camino lo más lejos posible antes de retroceder (y explorar caminos alternos lo más lejos posible antes de retroceder más) tiene el efecto de visitar todos los vértices de G_1 antes de pasar a un subgrafo nuevo adyacente a v, en este caso G_2 o G_3. Después se visitarán todos los vértices de G_2 antes de visitar cualquier vértice de G_3. Esto es análogo al recorrido de árboles, que visita todos los vértices de un subárbol antes de pasar al siguiente subárbol.

Por último, necesitamos considerar el hecho de que no es forzoso que se pueda llegar a todos los vértices de un grafo desde el vértice en el que se inició una búsqueda primero en profundidad.

```
dfs(G, v) // BOSQUEJO
```

Marcar v como "descubierto".
Para cada vértice w tal que la arista vw está en G:
    Si w no se ha descubierto:
        `dfs(G, W)`; es decir, explorar vw, visitar w, explorar desde ahí hasta donde sea posible, y retroceder de w a v.
    Si no:
        "Verificar" vw sin visitar w.
Marcar v como "terminado".

`barridoDfs(G)` // *BOSQUEJO*

Asignar a todos los vértices de G el valor inicial "no descubierto".
Para cada vértice $v \in G$, en algún orden:
Si v no se ha descubierto:
`dfs(G, v)`; es decir, realizar una búsqueda primero en profundidad que inicie (y termine) en v; cualesquier vértices descubiertos durante una visita de búsqueda primero en profundidad previa no se vuelven a visitar; todos los vértices visitados durante esta `dfs` se clasifican ahora como "descubiertos".

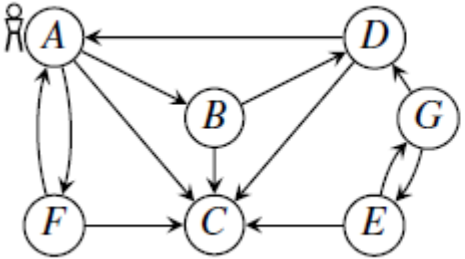
Dada la descripción informal de la búsqueda primero en profundidad, vemos que `barridoDfs` (mediante invocaciones de `dfs`) visita todos los vértices de G exactamente una vez, y recorre todas las aristas de G una vez en la dirección hacia adelante (explorando) y una vez en la dirección hacia atrás (retrocediendo). Sin embargo, cuando la arista conduce a un vértice que ya se descubrió, en lugar de decir que la arista se explora y de inmediato se retrocede por ella, decimos que la arista se *verifica*.

Búsqueda primero en amplitud

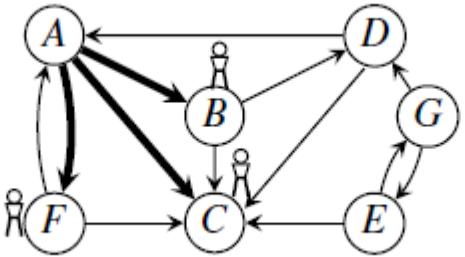
- La búsqueda primero en amplitud es muy diferente de la búsqueda primero en profundidad en términos del orden en el que se descubren los vértices.

- En lugar de un viaje realizado por una persona, la mejor forma de visualizar la búsqueda primero en amplitud es como muchas exploraciones simultáneas (o casi simultáneas) que parten de un mismo punto y se extienden de manera independiente.

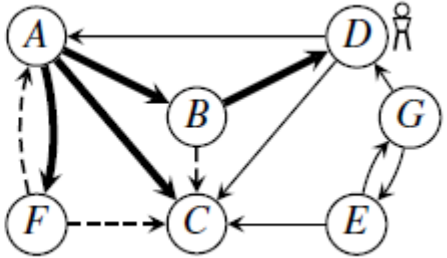
- Veamos como funciona la búsqueda primero en amplitud partiendo desde el vértice A del siguiente grafo.
- En lugar un solo turista, un autobús lleno de turistas deja a sus pasajeros en el punto A, desde donde comienzan a caminar en el diagrama de la izquierda.
- Los turistas se dispersan y exploran en todas las direcciones que permiten las aristas que salen de A.
- Las aristas son como puentes en un solo sentido, también para los peatones.



Todos inician en el nodo A.



Diversos grupos han llegado a B, C y F.



Turistas de B llegan al nodo D.

Las líneas punteadas indican aristas que se exploraron pero conducían a vértices que ya se habían descubierto antes.

En la última fase de la búsqueda se exploran de forma similar las aristas **DA** y **DC**.
En la búsqueda primero en amplitud no hay retrocesos y **E** y **G** son inalcanzables, así que la búsqueda terminará una vez exploradas estas dos últimas aristas.

- Si examinamos el último diagrama, veremos que las aristas dibujadas con líneas gruesas continuas, que condujeron a vértices no descubiertos durante la búsqueda, forman otra vez un árbol, aunque es diferente del que se formó en el recorrido en profundidad.
- Si hay dos o más caminos más cortos a un vértice dado, el empate se romperá de alguna manera y sólo una arista se considerará como “descubridora” del vértice.
- La ganadora depende de los detalles de la implementación y de la estructura de datos al ejecutarse un programa de computadora.

- En la búsqueda primero en amplitud los vértices se visitan en orden de distancia creciente respecto al punto de partida, digamos ***s***.

- La *“distancia”* es simplemente el número de aristas incluidas en un camino más corto.

- En un principio ninguno de los vértices se ha descubierto.

- El paso central de la búsqueda primero en amplitud, que parte de **d = 0** y se repite hasta que dejan de hallarse vértices nuevos, consiste en considerar por turno cada vértice **v** que está a una distancia **d** de **s** y examinar todas las aristas que van desde **v** hacia vértices adyacentes.

- Para cada arista **vw**, si **w** no ha sido descubierto, se añade **w** al conjunto de vértices que están a una distancia **d + 1** del punto de partida **s**; en caso contrario, **w** estará más cerca, y ya se conocerá su distancia.

- Una vez procesados de esta manera todos los vértices que están a la distancia d, se procesan los vértices que están a la distancia **d+1** y así sucesivamente.

- La búsqueda termina cuando se llega a una distancia a la que ya no hay vértices.

- Para la búsqueda primero en amplitud del ejemplo anterior, las distancias son **0** para **A**, **1** para **B**, **C** y **F**, y **2** para **D**.

- Un árbol abarcante primero en amplitud contiene un vértice de árbol por cada vértice de grafo al que se puede llegar partiendo de **s**, de ahí el nombre “abarcante”.
- Además, el camino en el árbol que va de **s** a cualquier vértice contiene el número mínimo posible de aristas; por ello, la profundidad de en este árbol es su distancia mínima en aristas respecto a **s**.
- En la parte final del ejemplo anterior, las aristas gruesas continuas constituyen un árbol abarcante primero en amplitud.

- El algoritmo siguiente pone en práctica la búsqueda primero en amplitud y halla un **árbol abarcante primero en amplitud** cuya raíz es un vértice inicial dado, **s.**
- El árbol se almacena como árbol adentro en el arreglo padre.
- Como siempre sucede en los árboles adentro, al camino que va del vértice inicial **s** a cualquier vértice se puede descubrir en orden inverso siguiendo los elementos de padre de hasta **s.**
- Se asigna el valor **-1** al padre de **s** para indicar que **s** es la raíz.

Algoritmo Búsqueda Primero en Amplitud

- **Entradas:** G = (V, E), un grafo representado por una estructura de listas de adyacencia (*verticesAdya*) donde V = {1, ..., n}.
  s ∈ **V**, el vértice en el que se inicia la búsqueda.

- **Salidas:** Un árbol abarcante primero en amplitud, almacenado en el arreglo padre. Ese arreglo se pasa como parámetro y el algoritmo se encarga de llenarlo.

- ***Comentarios:*** Para una cola **Q**, suponemos que se usan operaciones del tipo de datos abstracto Cola.

- El arreglo *color[1], . . . , color[n]* denota la situación actual de todos los vértices respecto a la búsqueda.

- Los vértices no descubiertos son blancos.
- Los que ya se descubrieron pero todavía no se procesan (en la cola) son grises.
- Los que ya se procesaron son negros.

```
void busquedaPrimeroEnAmplitud(ListaInt[] verticesAdya, int n, int, s,
int[] padre)
    int[] color = new int[n+1];
    Cola pendiente = crear(n);
    Inicializar color[1], ..., color[n] con blanco.

    padre[s] = -1;
    color[s] = gris;
    encolar(pendiente, s);
    while (pendiente no esté vacío)
       v = frente(pendiente);
       desencolar(pendiente);
       Por cada vértice w de la lista verticesAdya[v];
          if (color[w] == blanco)
              color[w] = gris;
              encolar(pendiente, w);
              padre[w] = v; // Procesar la arista vw del árbol.
          // Seguir con la lista.
       // Procesar el vértice v aquí.
       color[v] = negro;
    return;
```

- El algoritmo anterior sirve como esqueleto para todas las aplicaciones de *búsqueda primero en amplitud*.

- Los *comentarios* indican dónde se insertaría el código para procesar vértices y aristas de árbol (aristas a vértices no descubiertos, que constituyen el árbol abarcante primero en amplitud.

- No es forzoso que se pueda llegar a todos los vértices desde un vértice de partida dado.

Análisis de búsqueda primero en amplitud

- Suponemos que **G** tiene **n** vértices y **m** aristas, y que la búsqueda llega a todo **G**.
- Además, suponemos que cada operación de cola tarda un tiempo constante.
- Por último, suponemos que el procesamiento que la aplicación efectúa con vértices y aristas individuales tarda un tiempo constante con cada uno; de lo contrario, sería necesario multiplicar los costos apropiados por el tiempo que tarda el procesamiento de cada operación.

- Cada arista se procesa una vez en el ciclo *while* para dar un costo de $\mathbf{\Theta(m)}$.

- Cada vértice se coloca en la cola, se saca de ella y se procesa una vez, para dar un costo de $\mathbf{\Theta(n)}$.

- Se usa espacio extra para el arreglo color y la cola, y dicho espacio está en $\mathbf{\Theta(n)}$.