Teoría de Grafos y Árboles

UCR – ECCI

CI-0111 Estructuras Discretas

Prof. Kryscia Daviana Ramírez Benavides

Teoría de Grafos

- Los grafos son estructuras discretas que aparecen ubicuamente en cada disciplina donde se requiere modelar algo.
- En general los grafos son mapas conceptuales que ayudan a representar el conocimiento.
- Los grafos tiene muchas aplicaciones en problemas de ingeniería, computación, biología, física, urbanismo, comunicaciones, economía, redes sociales, entre muchas otras.

Teoría de Grafos (cont.)

- Aplicaciones de los grafos:
  - Internet y los protocolos de comunicaciones (TCP/IP, SMTP, FTP, routers, etc.)
  - Diseño de redes de comunicaciones y transporte (redes computacionales, carreteras, aguas, electricidad, telecomunicaciones, aviación, satélites, aero-espacial, flotas de vehículos, etc.)
  - Estructura de datos
  - Algoritmos computacionales
  - Navegador GPS (Google Maps)

Teoría de Grafos (cont.)

- Aplicaciones de los grafos:
  - Economía (bolsa, transacciones económicas, modelos de mercado, etc.)
  - Empresa (localización, estrategia, teoría de juegos, análisis de la competencia, logística, CRM, ERP, MRP, *scheduling*, esquema organizacional, etc.)
  - Redes sociales (Facebook, Skype, MSN)
  - Política y marketing
  - Seguridad y prevención del terrorismo
  - Inteligencia militar

Teoría de Grafos (cont.)

- Aplicaciones de los grafos:
  - Ocio (video juegos, simulación e IA)
  - Domótica, robótica, automatización, control y gestión de redes, etc.
  - La Web (es un grafo cuya estructura cambia cada segundo)
  - Web semántica
  - Investigación médica, biogenética (secuenciación ADN)
  - Biología, medio-ambiente, cambio climático
  - Árboles genealógicos

Propiedades de los Grafos

- **Adyacencia.** Dos aristas son adyacentes si tienen un vértice en común, y dos vértices son adyacentes si una arista los une.
- **Incidencia.** Una arista es incidente a un vértice si ésta lo une a otro.
- **Ponderación.** Corresponde a una función que asocia a cada arista un valor (costo, peso, longitud, etc.), para aumentar la expresividad del modelo.
- **Etiquetado.** Distinción que se hace a los vértices y/o aristas mediante una marca que los hace unívocamente distinguibles del resto.

¿Qué es un grafo?

- Un **grafo *G*** consiste en un conjunto no vacío de **vértices** o **nodos** V y un conjunto de **arcos** o **aristas** $E \Rightarrow \boldsymbol{G = (V,E)}$.
  - Grafo del griego *grafos*: dibujo, imagen.
- Cada $e \in E$ tiene un par $(v_1,v_2) \in V \times V$ asociado; es decir, ***e*** **conecta** $\boldsymbol{v_1}$ **con** $\boldsymbol{v_2}$.
- El **orden** del grafo G a su número de vértices $\Rightarrow \boldsymbol{G = |V|}$.
- El **grado** de un vértice o nodo $v \in V$ es igual al número de arcos o aristas que lo tienen como extremo.

¿Qué es un grafo? (cont.)

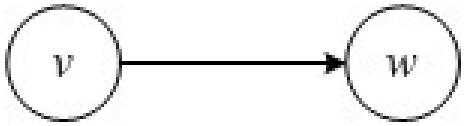
- **Vértices (Nodos).** Elementos (objetos) que forman un grafo. Cada uno lleva asociada un grado característico según la situación, que se corresponde con la cantidad de arcos o aristas que confluyen en dicho vértice.
- **Arcos o Aristas.** Relaciones entre los elementos que forman el grafo. Existen los siguientes tipos:
  - **Ramas.** Arco o arista que unen un vértice con otro.
  - **Paralelas (múltiples).** Arcos o aristas conjuntas si el vértice inicial y final son el mismo.
  - **Cíclicas (lazo, bucle).** Arco o arista que el vértice inicial y final es el mismo.

¿Qué es un grafo? (cont.)

- Un **grafo ponderado $G = (V,E,w)$** es un grafo en el que a cada arco o arista $e \in E$ se le asocia un peso $w(e) \in \mathbb{R}$.
- En un grafo dirigido, un vértices es **adyacente** a otro si están conectados por un arco (cola → cabeza), donde el vértice cabeza es adyacente al vértice cola.
  - En tal caso, el vértices cola es **incidente** al vértice cabeza.
- En un grafo no dirigido, dos vértices son **adyacentes** si están conectados por una arista.
  - En tal caso, cada uno de estos vértices es **incidente** a dicha arista.

Grafos Dirigidos

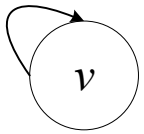
- Un **grafo dirigido** o **digrafo** G consiste en un conjunto no vacío de vértices V y un conjunto de arcos $E \Rightarrow G = (V,E)$.
  - Los **vértices** se denominan también **nodos** o **puntos**.
  - Los **arcos** pueden llamarse **arcos dirigidos** o **líneas dirigidas**.
- Un arco es un par ordenado de vértices (v,w); donde v es la cola (nodo inicial) y w es la cabeza (nodo terminal) del arco.
  - El arco (v,w) se expresa a menudo como $v \rightarrow w$, y se representa como:



  - Se dice que el arco $v \rightarrow w$ va de v a w, y que ***w* es adyacente o vecino a *v***.
  - Se dice que el arco ***v* → *w* es incidente en el vértice *v***.

Grafos Dirigidos (cont.)

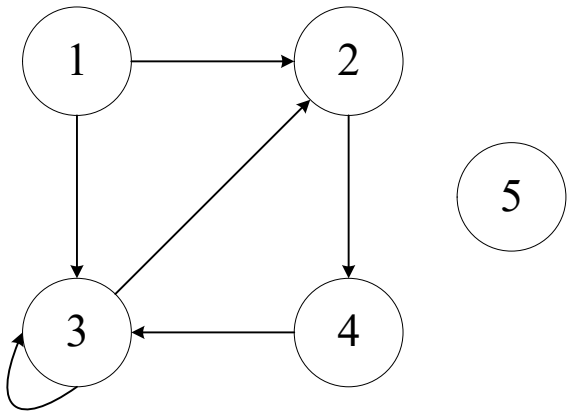
- Cuando un arco es un par ordenado de vértices idénticos se llama **lazo**, **bucle** o **arco cíclico**, arco que une un vértice consigo mismo.
  - El lazo (v,v) se expresa a menudo como $v \rightarrow v$, y se representa como:



  - Se dice que el arco $v \rightarrow v$ va de v a v, y que ***v* es adyacente a *v***.
  - Se dice que el arco ***v* → *v* es incidente en el vértice *v***.

Grafos Dirigidos (cont.)

- El **grado de entrada (interno) de *v*** es el número de arcos que tienen como nodo terminal a *v* y se denota por $g^-(v)$.
- El **grado de salida (externo) de *v*** es el número de arcos que tienen como nodo inicial a *v* y se denota por $g^+(v)$.
  - Sea $G = (V,E)$ un grafo dirigido, entonces $\sum_{v \epsilon V} g^-(v) = \sum_{v \epsilon V} g^+(v) = |E|$



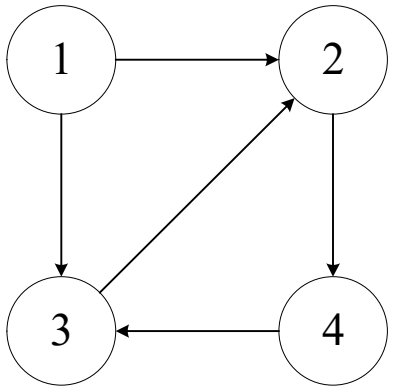
$g^-(1) = 0 \quad g^+(1) = 2$
$g^-(2) = 2 \quad g^+(2) = 1$
$g^-(3) = 3 \quad g^+(3) = 2$
$g^-(4) = 1 \quad g^+(4) = 1$
$g^-(5) = 0 \quad g^+(5) = 0$

Grafos Dirigidos (cont.)

- Los vértices de un grafo dirigido pueden usarse para representar objetos, y los arcos para representar relaciones entre los objetos.
- Los grafos pueden clasificarse en:
  - Grafo simple
  - Multigrafo
  - Pseudografo

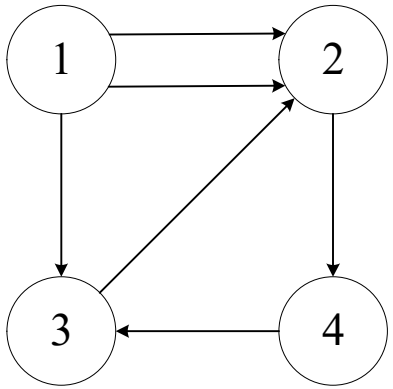
Grafos Dirigidos (cont.)

- Un grafo simple $G = (V,E)$ está compuesto por un conjunto no vacío de vértices V y un conjunto de arcos E, donde se tiene un conjunto de pares de elementos distintos de V.
  - $V = \{1,2,3,4\}$
  - $E = \{(1,2),(1,3),(2,4),(3,2),(4,3)\}$



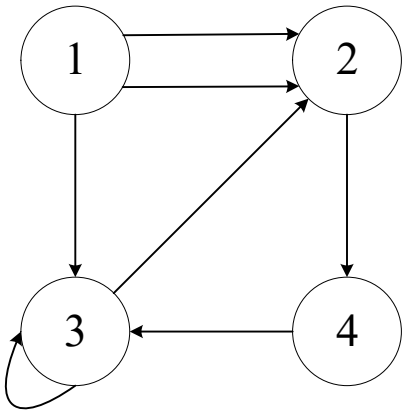
Grafos Dirigidos (cont.)

- Un multigrafo $G = (V,E)$ está compuesto por un conjunto no vacío de vértices V y un conjunto de arcos E en el que se permite que haya arcos múltiples (arcos paralelos, aquellos que conectan el mismo par de vértices).
  - $V = \{1,2,3,4\}$
  - $E = \{(1,2),(1,2),(1,3),(2,4),(3,2),(4,3)\}$



Grafos Dirigidos (cont.)

- Un pseudografo $G = (V,E)$ está compuesto por un conjunto no vacío de vértices V y un conjunto de arcos E en el que se permite que haya arcos múltiples (arcos paralelos) y lazos (arcos cíclicos).
  - $V = \{1,2,3,4\}$
  - $E = \{(1,2),(1,2),(1,3),(2,4),(3,2),(3,3),(4,3)\}$



Grafos Dirigidos (cont.)

- Un grafo simple $G = (V,E)$ es un pseudografo en el que no se permite que haya arcos múltiples ni lazos.
- Un multigrafo $G = (V,E)$ es un pseudografo en el que se permite que haya arcos múltiples.

Grafos Dirigidos (cont.)

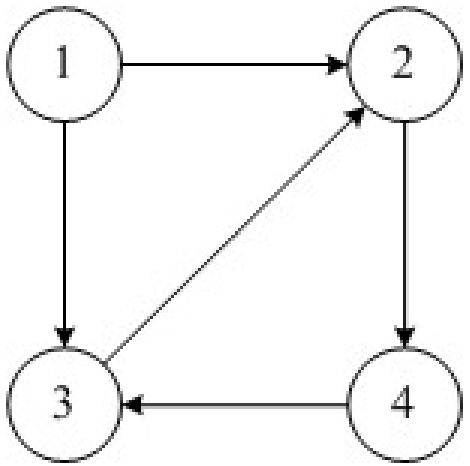
- Un **camino** o **cadena** en un grafo dirigido es una secuencia de vértices $v_1, v_2, \ldots, v_n$, tal que $v_1 \rightarrow v_2, v_2 \rightarrow v_3, \ldots, v_{n-1} \rightarrow v_n$ son arcos.
  - Este camino va del vértice v_1 al vértice v_n, pasando por los vértices $v_2, v_3, \ldots, v_{n-1}$.
  - La **longitud** de un camino es el número de arcos en ese camino, en este caso, $n - 1$.
  - Se pueden repetir vértices y arcos.
  - Como caso especial, un vértice sencillo v, por sí mismo denota un camino de longitud 0 de v a v si no tiene lazos.

Grafos Dirigidos (cont.)

- Un **camino simple** es un camino en donde todos los arcos son distintos (se pueden repetir vértices).
- Un **circuito** es un camino simple cerrado de longitud por lo menos uno, es decir, que empieza y termina en el mismo vértice (se pueden repetir vértices).
- Un **camino elemental** es un camino simple en donde todos los vértices son distintos, excepto tal vez el primero y el último.
- Un **ciclo** es un camino elemental cerrado de longitud por lo menos uno, es decir, que empieza y termina en el mismo vértice.

Grafos Dirigidos (cont.)

- En el grafo $G = (V,E)$, donde $V = \{1,2,3,4\}$ y $E = \{(1,2),(1,3),(2,4),(3,2),(4,3)\}$.

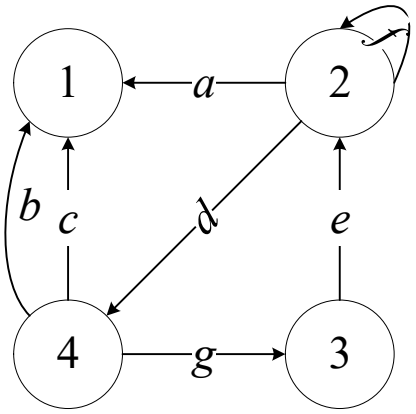


Camino = (1,3,2,4,3,2), longitud 5
Camino simple = (1,3,2,4,3)
Circuito = (3,2,4,3)
Camino elemental = (3,2,4)
Ciclo = (3,2,4,3)

Representación de Grafos Dirigidos

- Una representación común para un grafo dirigido $G = (V,E)$ es la **matriz de adyacencia**.
- La matriz de adyacencia para G es una matriz A de dimensión n x n, donde las entradas $A[i,j]$ cuentan el números de arcos que van del vértice i (nodo inicial) al j (nodo terminal).
  - A_G no es simétrica en general.
  - Si G es un grafo simple $\Longrightarrow$ $A_{ii} = 0$ y $A_{ij} \in \{0,1\}$.
  - Si G es un multigrafo $\Longrightarrow$ $A_{ii} = 0$.

Representación de Grafos Dirigidos (cont.)

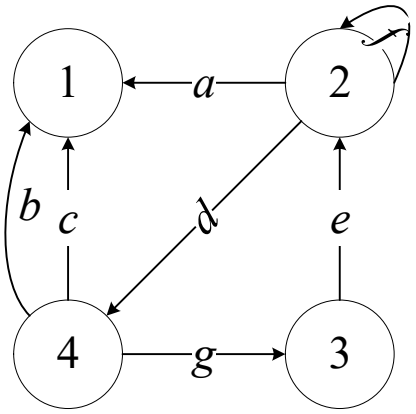


$$A_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Representación de Grafos Dirigidos (cont.)

- Otra representación, relacionada con la anterior, para un grafo dirigido $G = (V,E)$ es la **matriz de adyacencia etiquetada**.
- La matriz de adyacencia etiquetada para G es una matriz A de dimensión n x n, donde $A[i,j]$ es la etiqueta o ponderación del arco que va del vértice i al j.
  - Si no existe un arco de i a j debe emplearse como entrada para $A[i,j]$ un valor que no pueda ser una etiqueta válida, el cual puede ser – (en caso de valores no numéricos) o ∞ (caso de valores numéricos).

Representación de Grafos Dirigidos (cont.)



$$A_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} - & - & - & - \\ a & f & - & d \\ - & e & - & - \\ b/c & - & g & - \end{pmatrix} \end{array}$$

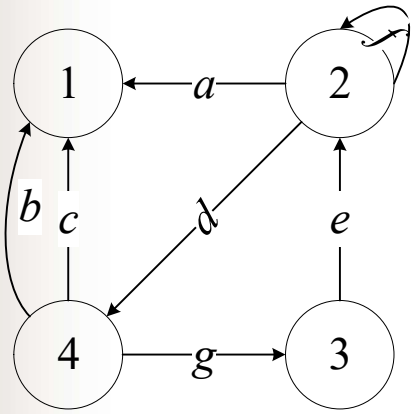
Representación de Grafos Dirigidos (cont.)

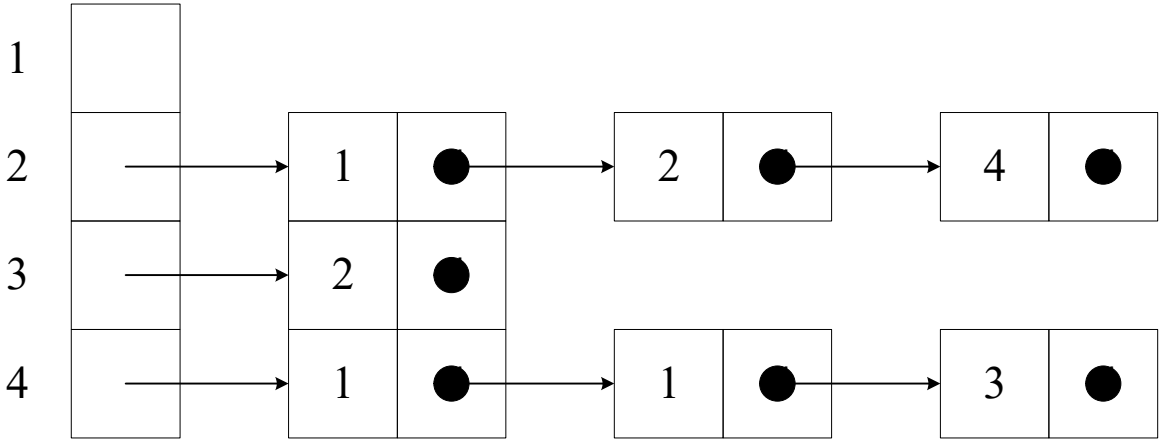
- La ventaja de usar una matriz de adyacencia es que el tiempo de acceso requerido a un elemento es independiente del tamaño de V y A.
- La desventaja de usar una matriz de adyacencia es que requiere un espacio $\Omega(n^2)$ aun si el grafo tiene menos de n^2 arcos.
  - Sólo leer o examinar la matriz puede llevar un tiempo $O(n^2)$.
- Para evitar esta desventaja, se puede utilizar otra representación común para un grafo dirigido $G = (V,E)$ llamada representación con **lista de adyacencia**.

Representación de Grafos Dirigidos (cont.)

- La lista de adyacencia para un vértice i es una lista, en algún orden, de todos los vértices adyacentes a i.
  - Se puede representar G por medio de un arreglo CABEZA, donde CABEZA[i] es un apuntador a la lista de adyacencia del vértice i.
- La representación con lista de adyacencia de un grafo dirigido requiere un espacio proporcional a la suma del número de vértices más el número de arcos.
  - Se usa bastante cuando el número de arcos es mucho menor que n^2.
  - Una desventaja potencial es que puede llevar un tiempo $O(n)$ determinar si existe un arco del vértice i al vértice j.

Representación de Grafos Dirigidos (cont.)



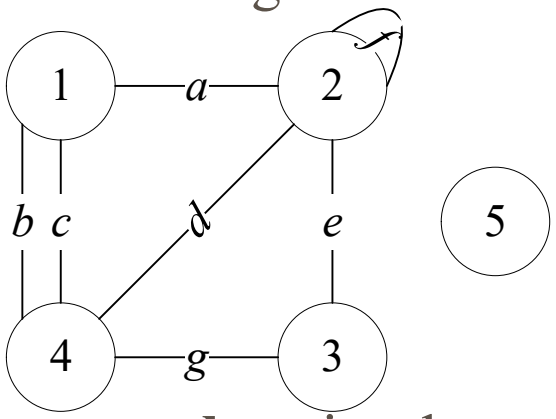


Grafos No Dirigidos

- Parte de la terminología para grafos dirigidos es aplicable a los no dirigidos.
- Un **grafo no dirigido** G consiste en un conjunto no vacío de vértices V y un conjunto de aristas $E \Rightarrow G = (V,E)$.
  - Los **vértices** se denominan también **nodos** o **puntos**.
  - Las **aristas** es un par no ordenado de vértices; la arista $(v,w) = (w,v)$.
- Los vértices v y w son **adyacentes** si es una arista (v,w).
  - Se dice que la arista (v,w) es **incidente** sobre los vértices v y w.

Grafos No Dirigidos (cont.)

- El número de aristas incidentes con un vértice *v* de un grafo *G* se denomina **grado** o **valencia de *v*** y se denota por ***g*(*v*)**.
  - Los lazos dan **dos** unidades al grado del vértice que lo tenga.
  - Los vértices de grado 1 se denominan **terminales**.
  - Los vértices de grado 0 se denominan **aislados**.



$g(1) = 3$
$g(2) = 5$
$g(3) = 2$
$g(4) = 4$
$g(5) = 0$

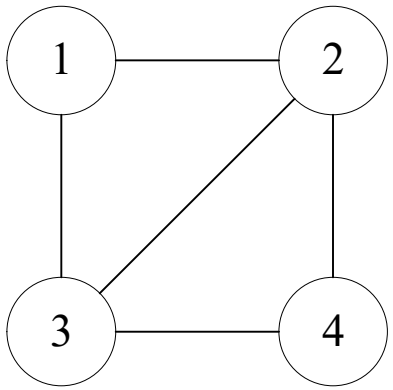
- Un grafo es **regular** si todos sus vértices tienen el mismo grado.

Grafos No Dirigidos (cont.)

- Los vértices de un grafo no dirigido pueden usarse para representar objetos, y las aristas para representar relaciones entre los objetos.
- Los grafos pueden clasificarse en:
  - Grafo simple
  - Multigrafo
  - Pseudografo

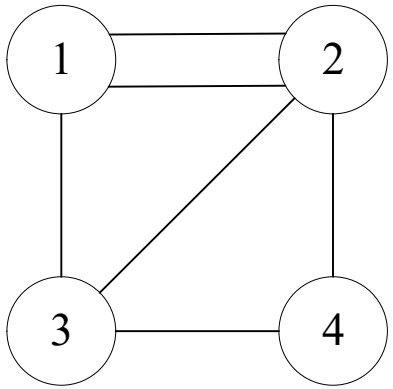
Grafos No Dirigidos (cont.)

- Un grafo simple $G = (V,E)$ está compuesto por un conjunto no vacío de vértices V y un conjunto de aristas E, donde se tiene un conjunto de pares de elementos distintos de V.
  - $V = \{1,2,3,4\}$
  - $E = \{(1,2),(1,3),(2,3),(2,4),(3,4)\}$



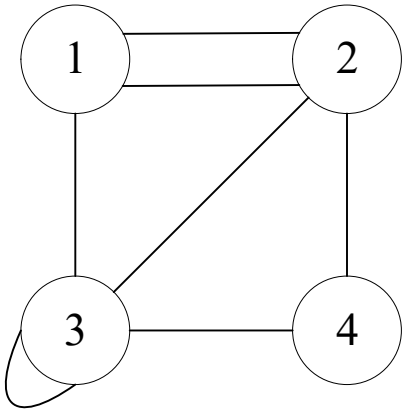
Grafos No Dirigidos (cont.)

- Un multigrafo $G = (V,E)$ está compuesto por un conjunto no vacío de vértices V y un conjunto de aristas E en el que se permite que haya aristas múltiples (aristas paralelas, aquellos que conectan el mismo par de vértices).
  - $V = \{1,2,3,4\}$
  - $E = \{(1,2),(1,2),(1,3),(2,3),(2,4),(3,4)\}$



Grafos No Dirigidos (cont.)

- Un pseudografo $G = (V,E)$ está compuesto por un conjunto no vacío de vértices V y un conjunto de aristas E en el que se permite que haya aristas múltiples (aristas paralelas) y lazos (aristas cíclicas).
  - $V = \{1,2,3,4\}$
  - $E = \{(1,2),(1,2),(1,3),(2,3),(2,4),(3,3),(3,4)\}$



Grafos No Dirigidos (cont.)

- Un grafo simple $G = (V,E)$ es un pseudografo en el que no se permite que haya aristas múltiples ni lazos.
- Un multigrafo $G = (V,E)$ es un pseudografo en el que se permite que haya aristas múltiples.

Grafos No Dirigidos (cont.)

- Un **camino** o **cadena** en un grafo no dirigido es una secuencia de vértices v_1, v_2, …, v_n, tal que (v_1,v_{i+1}) es una arista para $1 \leq i < n$.
  - Este camino va del vértice v_1 al vértice v_n, pasando por los vértices v_2, v_3, …, v_{n-1}.
  - La **longitud** de un camino es el número de aristas en ese camino, en este caso, $n - 1$.
  - Se pueden repetir vértices y aristas.

Grafos No Dirigidos

- Un **camino simple** es un camino en donde todos las aristas son distintas (se pueden repetir vértices).
- Un **circuito** es un camino simple cerrado de longitud por lo menos 1, es decir, que empieza y termina en el mismo vértice (se pueden repetir vértices).
- Un **camino elemental** es un camino simple en donde todos los vértices son distintos, excepto tal vez el primero y el último.
- Un **ciclo** es un camino elemental cerrado de longitud mayor o igual a 3, es decir, que empieza y termina en el mismo vértice.

Grafos No Dirigidos (cont.)

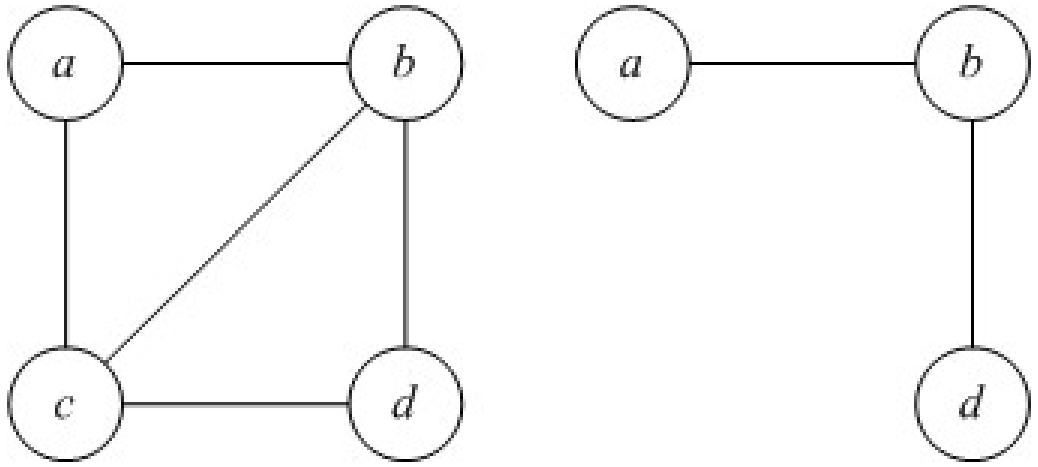
- No se consideran ciclos los caminos de la forma *v* (camino de longitud 0), *v*,*v* (camino de longitud 1), o *v*,*w*,*v* (camino de longitud 2).

Grafos No Dirigidos (cont.)

- Sea $G = (V,E)$ un grafo con conjunto de vértices V y conjunto de aristas E. Un **subgrafo** de G es un grafo $G' = (V',E')$ donde:
  - V' es un subconjunto de V.
  - E' consta de las aristas (v,w) en E tales que v y w están en V'.
- Si E' consta de todas las aristas (v,w) en E, tal que v y w están en V', entonces G' se conoce como un **subgrafo inducido** de G.

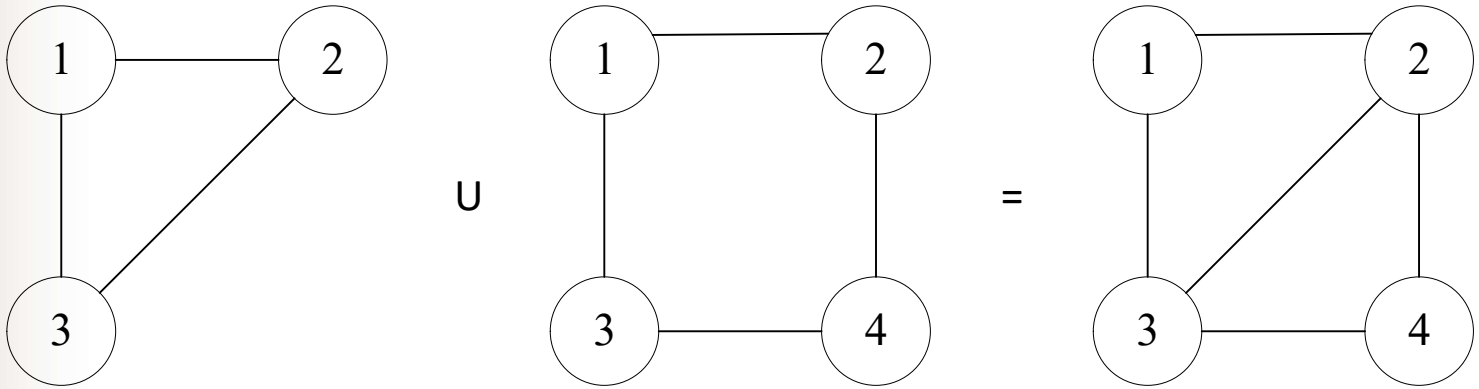
Grafos No Dirigidos (cont.)

- En el grafo $G = (V,E)$, donde $V = \{a,b,c,d\}$ y $E = \{(a,b),(a,c),(b,c),(b,d),(c,d)\}$, y uno de sus subgrafos inducidos G' definido por el conjunto de vértices $V' = \{a,b,c\}$ y $E' = \{(a,b),(b,d)\}$.



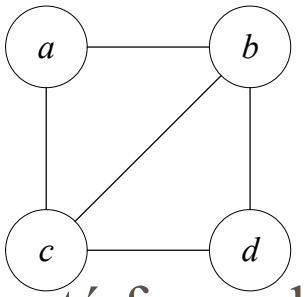
Grafos No Dirigidos (cont.)

- La **unión** de dos grafos simples $G_1 = (V_1,E_1)$ y $G_2 = (V_2,E_2)$ es el grafo simple $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$.



Grafos No Dirigidos

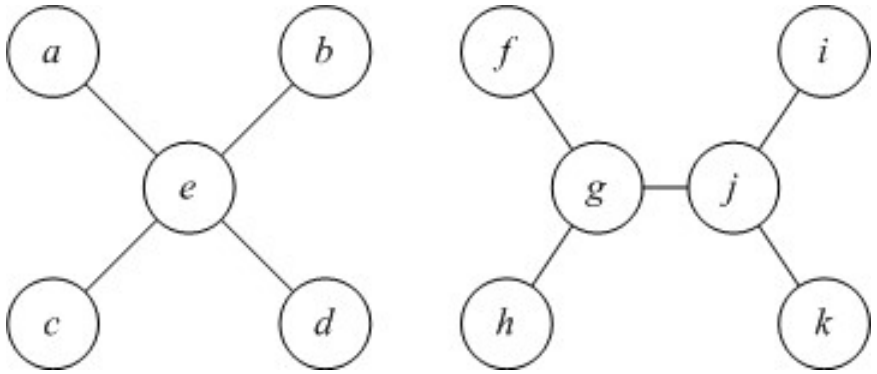
- Un grafo es **conexo** si todos sus pares de vértices están conectados, es decir, si cada par de vértices $v,w \in V$ pueden estar conectados por un camino elemental.
  - Si dos vértices de un grafo se pueden conectar por un camino, entonces existe un camino elemental que los une.
  - Si un grafo dirigido es conexo se llama **fuertemente conexo**.



- Un grafo no conexo está formado por la unión de varios subgrafos conexos y desconectados entre sí que se denominan **componentes conexos** del grafo.

Grafos No Dirigidos (cont.)

- Un **componente conexo** de un grafo *G* es un subgrafo conexo inducido maximal, es decir, un subgrafo conexo inducido que por sí mismo no es un subgrafo propio de ningún otro subgrafo conexo de *G*.
- El grafo no dirigido anterior es un grafo conexo que tiene sólo un componente conexo, y es él mismo.
- El siguiente grafo no dirigido tiene dos componentes conexos.



Grafos No Dirigidos (cont.)

- Un **grafo cíclico** contiene por lo menos un ciclo.
- Un **grafo acíclico** algunas veces se conoce como **árbol libre**.
  - El grafo anterior muestra dos árboles libres.
- Los árboles libres tienen dos propiedades importantes:
  - Todo árbol libre con $n \geq 1$ vértices contiene exactamente $n - 1$ aristas.
  - Si se agrega cualquier arista a un árbol libre, resulta un ciclo.

Representación de Grafos No Dirigidos

- Los métodos de representación de grafos dirigidos se pueden emplear para representar los no dirigidos.
  - Una arista no dirigida entre *v* y *w* se representa simplemente con dos aristas dirigidas, una de *v* a *w*, y otra de *w* a *v*.
- Los métodos son:
  - Una representación con **matriz de adyacencia**. Esta matriz es simétrica.
  - Una representación con **matriz de adyacencia etiquetada**. Esta matriz es simétrica.
  - Una representación con **matriz de incidencia**.
  - Una representación con **lista de adyacencia**.

Representación de Grafos No Dirigidos

- Una representación común para un grafo dirigido $G = (V,E)$ es la **matriz de adyacencia**.
- La matriz de adyacencia para G es una matriz A de dimensión n x n, donde las entradas $A[i,j]$ cuentan el números de aristas que unen los vértices i y j.
  - A_G es simétrica en general.
  - Si G es un grafo simple $\Longrightarrow$ $A_{ii} = 0$ y $A_{ij} \in \{0,1\}$.
  - Si G es un multigrafo $\Longrightarrow$ $A_{ii} = 0$.

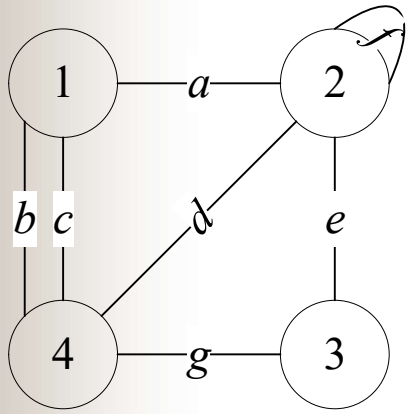
Representación de Grafos No Dirigidos (cont.)

- Otra representación, relacionada con la anterior, para un grafo dirigido $G = (V,E)$ es la **matriz de adyacencia etiquetada**.
- La matriz de adyacencia etiquetada para G es una matriz A de dimensión n x n, donde $A[i,j]$ es la etiqueta o ponderación de la arista que une los vértices i y j.
  - Si no existe un arco de i a j debe emplearse como entrada para $A[i,j]$ un valor que no pueda ser una etiqueta válida, el cual puede ser – (en caso de valores no numéricos) o ∞ (caso de valores numéricos).

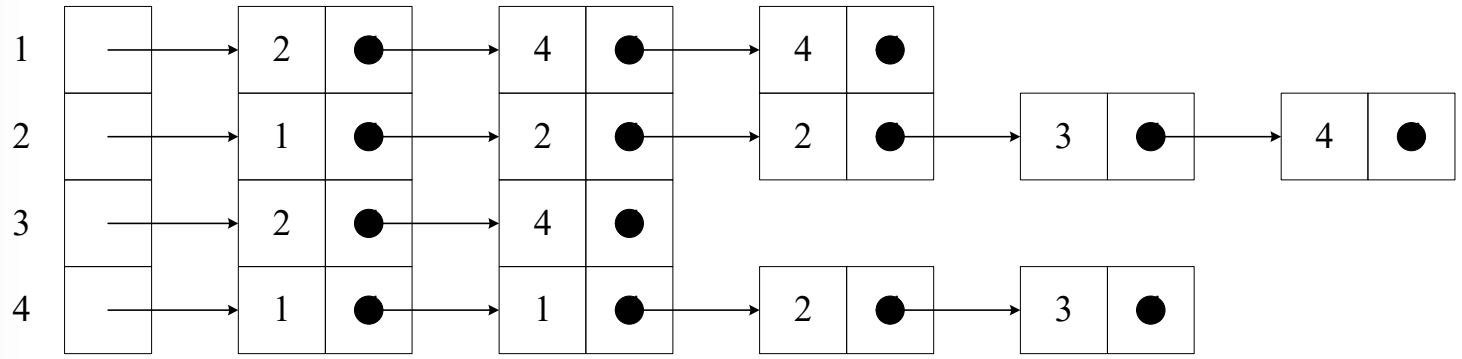
Representación de Grafos No Dirigidos (cont.)

- La lista de adyacencia para un vértice i es una lista, en algún orden, de todos los vértices adyacentes a i.
  - Se puede representar G por medio de un arreglo CABEZA, donde CABEZA[i] es un apuntador a la lista de adyacencia del vértice i.
- La representación con lista de adyacencia de un grafo dirigido requiere un espacio proporcional a la suma del número de vértices más el número de arcos.
  - Se usa bastante cuando el número de arcos es mucho menor que n^2.
  - Una desventaja potencial es que puede llevar un tiempo $O(n)$ determinar si existe un arco del vértice i al vértice j.

Representación de Grafos No Dirigidos (cont.)



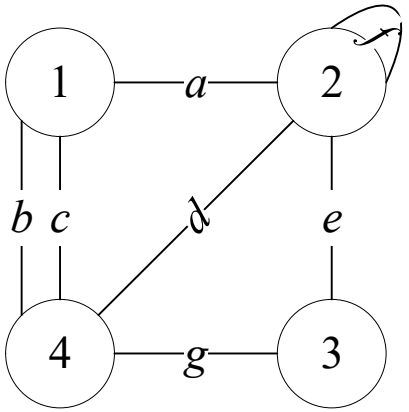
$$A_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 2 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{pmatrix} \end{array} \qquad A_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} - & a & - & b/c \\ a & f & e & d \\ - & e & - & g \\ b/c & d & g & - \end{pmatrix} \end{array}$$



Representación de Grafos No Dirigidos (cont.)

- La matriz de incidencia es otra forma de representar grafos.
- Sea $G = (V,E)$ un grafo no dirigido, donde v_1, v_2, ..., v_n son los vértices y e_1, e_2, ..., e_m son las aristas. La matriz de incidencia I_G es de orden $|V| \times |E|$, donde:
  - $I_{G\,ij} = \begin{cases} 0 & \text{si } e_j \text{ no es incidente con } v_i \\ 1 & \text{si } e_j \text{ es incidente con } v_i \end{cases}$

Representación de Grafos No Dirigidos (cont.)



$$I_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

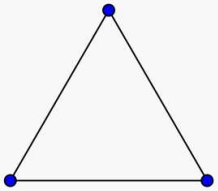
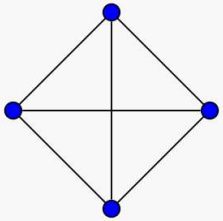
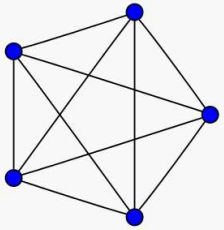
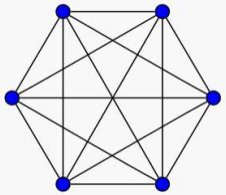
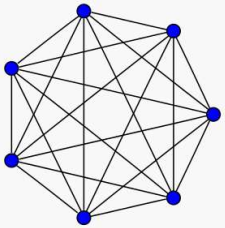
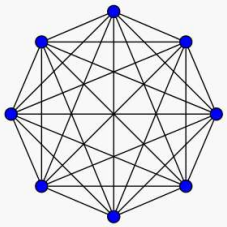
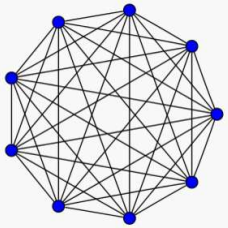
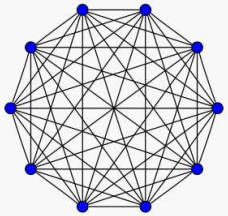
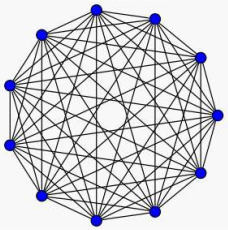
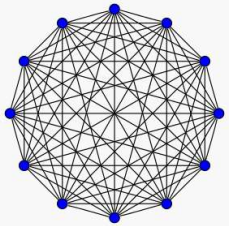
Grafos Particulares

- **Grafo nulo o vacío.** Grafo que no tiene vértices ni aristas.
- **Grafo trivial.** Grafo que tiene un vértice y ninguna arista.
- **Grafo regular.** Grafo donde todos sus vértices tienen el mismo grado.
- **Grafo simple.** Grafo que no permite bucles ni aristas múltiples (paralelas).
- **Multigrafo.** Grafo que permite aristas múltiples (paralelas).
- **Pseudografo.** Grafo que permite bucles y aristas múltiples (paralelas).
- **Árbol.** Grafo conexo sin ciclos.

Grafo Completo

- Un **grafo completo** K_n es un grafo simple donde cada par de vértices está conectado por una arista.
- Un grafo completo de n vértices tiene $n(n-1)/2$ aristas.
- Es un grafo regular con todos sus vértices de grado $n-1$.
- La única forma de hacer que un grafo completo se torne disconexo a través de la eliminación de vértices, sería eliminándolos todos.
- Un grafo simple que al menos un par de vértices no están conectados por una arista se llama **no completo**.

Grafo Completo

K_1: 0	K_2: 1	K_3: 3	K_4: 6
			
K_5: 10	K_6: 15	K_7: 21	K_8: 28
			
K_9: 36	K_{10}: 45	K_{11}: 55	K_{12}: 66
			

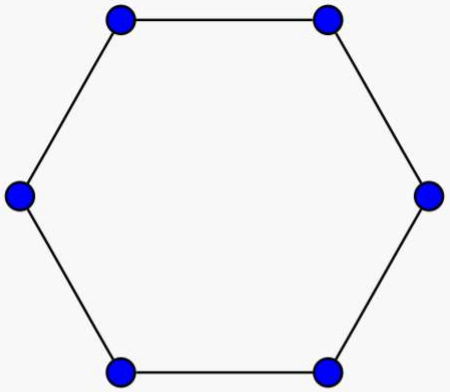
Grafo Ciclo

- Un **grafo ciclo** o simplemente **ciclo** C_n es un grafo simple que se asemeja a un polígono de n lados, con n vértices.
- Consiste en un camino cerrado en el que no se repite ningún vértice a excepción del primero que aparece dos veces como principio y fin del camino.
- El número de vértices en un grafo C_n es igual al número de aristas, y cada vértice tiene grado par, por lo tanto cada vértice tiene dos aristas incidentes.

Grafo Ciclo

- Propiedades:
- https://es.wikipedia.org/wiki/Grafo_ciclo

Grafo ciclo C_n



ciclo C_6

Vértices	n
Aristas	n
Cintura (girth)	n
Automorfismos	$2n$ (D_n)
Número cromático	• 2 si *n* es par • 3 si *n* es impar
Índice cromático	• 2 si *n* es par • 3 si *n* es impar
Propiedades	• 2-conexo por vértices • 2-conexo por aristas • 2-regular • Euleriano • Hamiltoniano • orientable

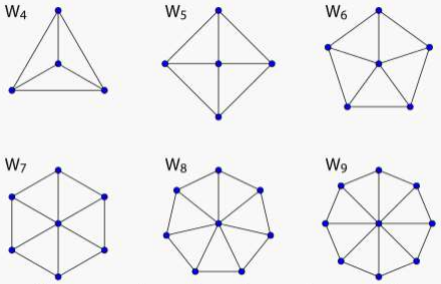
[editar datos en Wikidata]

Grafo Rueda

- Un **grafo rueda** W_n, o simplemente **rueda**, es un grafo simple que se obtiene desde un grafo C_n *agregando un nuevo vértice y uniendo mediante aristas este nuevo vértice a cada uno de los n* vértices de C_n.
- Los grafos rueda son grafos planos, y como tales pueden ser "incrustado" en un plano.
- Todo gráfico rueda es un grafo de Halin.
  - Son auto-duales: el dual de cualquier grafo rueda es un grafo isomórfico.
- En un grafo rueda siempre hay un ciclo hamiltoniano, habiendo n^2-$3n$+3 ciclos en W_n.

Grafo Rueda

Grafo rueda



Algunos ejemplos de grafos rueda

Vértices	n
Aristas	$2(n-1)$
Diámetro	2 si $n>4$ 1 si $n=4$
Cintura (girth)	3
Número cromático	3 si n es impar 4 si n es par
Propiedades	Hamiltoniano, Auto-dual, Planar

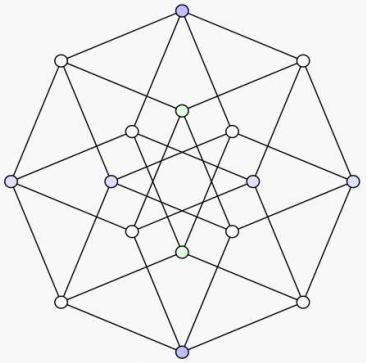
[editar datos en Wikidata]

Grafo Hipercubo (n-cubo)

- El **grafo hipercubo** Q_n es un grafo regular con 2^n vértices, que corresponden a los subconjuntos de un conjunto de n elementos.
- Cada vértice de Q_n es incidente a exactamente n aristas (por lo tanto, el grafo es n-regular) y por eso el número total de aristas es $2^{n-1}n$.

Grafo Hipercubo (*n*-cubo)

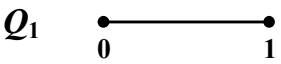
Grafo hipercubo Q_n



El grafo hipercubo Q_4

Nombre en honor a	Hipercubo
Vértices	2^n
Aristas	$2^{n-1}n$
Diámetro	n
Cintura (girth)	4, si $n \geq 2$
Número cromático	2
Propiedades	Simétrico Distancia regular Distancia unitaria Camino hamiltoniano

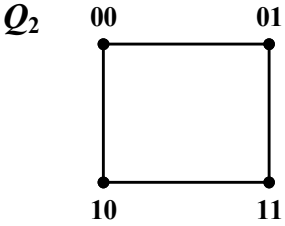
[editar datos en Wikidata]



$2^n = 2^1 = 2$ vértices

$2^{n-1}n = 2^0 * 1 = 1$ arista

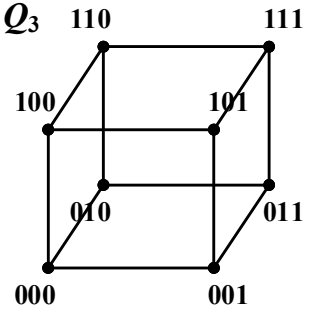
$n = 1$ arista incidente en cada vértice



$2^n = 2^2 = 4$ vértices

$2^{n-1}n = 2^1 * 2 = 4$ aristas

$n = 2$ aristas incidentes en cada vértice



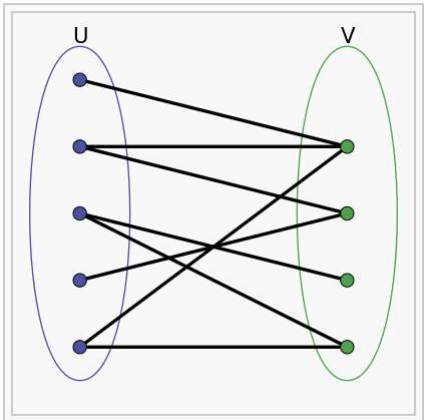
$2^n = 2^3 = 8$ vértices

$2^{n-1}n = 2^2 * 3 = 12$ aristas

$n = 3$ aristas incidentes en cada vértice

Grafo Bipartito

- Un **grafo bipartito (o bipartido)** es un grafo simple $G = (N,E)$ cuyos vértices se pueden separar en dos conjuntos disjuntos U y V, de manera que las aristas sólo pueden conectar vértices de un conjunto con vértices del otro.
- Los grafos bipartitos suelen representarse gráficamente con dos columnas (o filas) de vértices y las aristas uniendo vértices de columnas (o filas) diferentes.



Ejemplo de grafo bipartito.

Grafo Bipartito

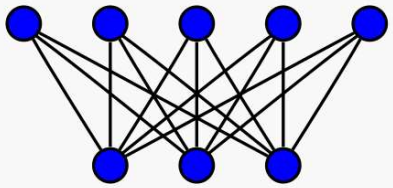
- Los dos conjuntos U y V pueden ser pensados como un coloreo del grafo con dos colores.
  - Un grafo simple es bipartito si y solo si es posible asignar uno de dos colores diferentes a cada vértice del grafo, de tal forma que ningún par (u,v) de vértices adyacentes recibe el mismo color.
- Un grafo bipartito con la partición de los vértices en U y V suele denotarse $G = (U, V, E)$.
  - Si $|U| = |V|$, esto es, si los dos subconjuntos tiene la misma cantidad de elementos o cardinalidad, decimos que el grafo bipartito G es **balanceado**.

Grafo Bipartito Completo

- Un **grafo bipartito (o bipartido) completo** es aquel grafo bipartito en el que todos los vértices de la partición V_1 están conectados a todos los vértices de la partición V_2 y viceversa
- El grafo completo bipartito con particiones de tamaño $V_1 = m$ y $V_2 = n$, es denotado como $K_{m,n}$
- Propiedades:
  - $|E| = |V_1| \cdot |V_2| = m \cdot n$=m
  - $K_{m,n}$ posee $m^{n-1} \cdot n^{m-1}$ árboles de expansión

Grafo Bipartito Completo

Grafo bipartito completo



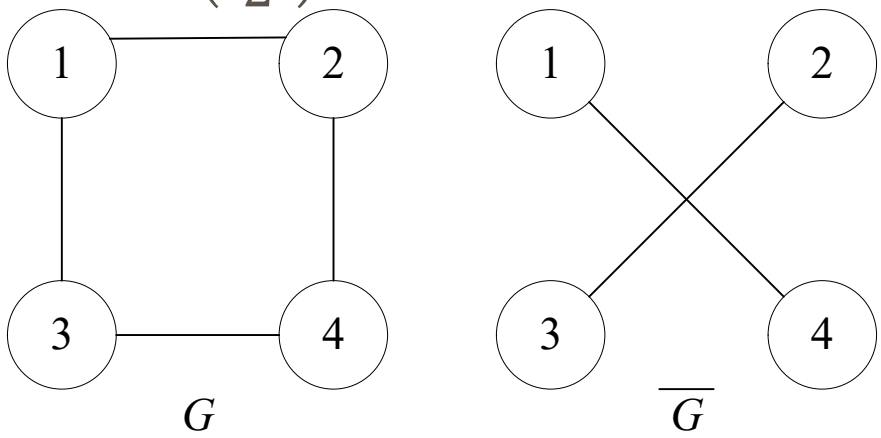
Un grafo bipartito completo con $m = 5$ y $n = 3$

Vértices	$n + m$
Aristas	mn
Radio	$\begin{cases} 1 & m = 1 \vee n = 1 \\ 2 & \text{en otro caso} \end{cases}$
Diámetro	$\begin{cases} 1 & m = n = 1 \\ 2 & \text{en otro caso} \end{cases}$
Cintura (girth)	$\begin{cases} \infty & m = 1 \vee n = 1 \\ 4 & \text{en otro caso} \end{cases}$
Automorfismos	$\begin{cases} 2m!n! & n = m \\ m!n! & \text{en otro caso} \end{cases}$
Número cromático	2
Índice cromático	max{m, n}

[editar datos en Wikidata]

Grafos Complementarios

- Un **grafo complementario** $\overline{G} = (V, \overline{E})$ de un grafo simple $G = (V,E)$ es aquel formando por el mismo conjunto de vértices y tal que dos vértices son adyacentes en $\overline{G}$ si y sólo si no son adyacentes en G; es decir, las aristas del grafo $\overline{G}$ serán las aristas que le falta al grafo G para ser un grafo completo.
  - $|E| + |\overline{E}| = \binom{|V|}{2}$



Grafos Isomorfos

- Un **isomorfismo** entre dos grafos simples $G = (V,E)$ y $G' = (V',E')$ es una biyección f entre los conjuntos de sus vértices $f: V \rightarrow V'$ que preserva la relación de adyacencia.
  - Es decir, cualquier par de vértices u y v de G son adyacentes si y solo si lo son sus imágenes, $f(u)$ y $f(v)$, en G'.
  - $(v_i, v_j) \in E \Leftrightarrow \left(f(v_i), f\left(v_j\right)\right) \in E'$
  - La función f es un isomorfismo entre G y G'.
- Esto implica que todos los vértices de G tienen un vértice equivalente en G' y que todas las aristas de G tienen una arista equivalente en G'.

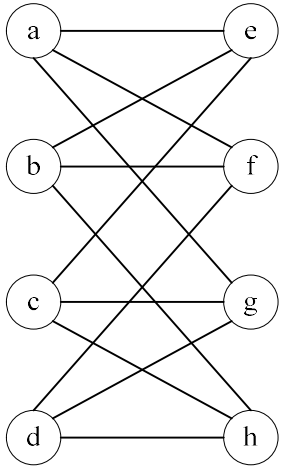
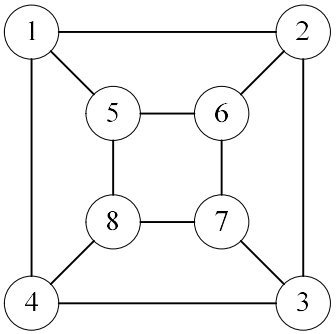
Grafos Isomorfos

- **Propiedades principales.** Dos **grafos** son **isomorfos** si tienen el mismo número de:
  - Vértices y aristas.
  - Componentes conexos.
  - Sucesión de sus grados igual.
  - Ciclos de longitud *n*.
  - Circuitos de Euler.
- Dos grafos son isomorfos si y sólo si, para alguna ordenación de vértices y aristas, sus matrices de adyacencia e incidencia son iguales.

Grafos Isomorfos

- Se debe analizar las matrices de adyacencia de los dos grafos.
  - https://www.youtube.com/watch?v=ybwRY2zzTvY
- Si dos grafos son isomorfos sus complementarios también lo son.
- Cuando el número de vértices y aristas es muy grande se complica demostrar que dos grafos son isomorfos mediante el análisis de las matrices de adyacencia e incidencia.
  - Por lo que se trata de ver si las propiedades principales coinciden todas en ambos grafos.

Grafos Isomorfos

Grafo G	Grafo H	Isomorfismo entre G y H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(e) = 5$ $f(f) = 2$ $f(g) = 4$ $f(h) = 7$

Grafos Isomorfos

	a	b	c	d	e	f	g	h
a	0	0	0	0	1	1	1	0
b	0	0	0	0	1	1	0	1
c	0	0	0	0	1	0	1	1
d	0	0	0	0	0	1	1	1
e	1	1	1	0	0	0	0	0
f	1	1	0	1	0	0	0	0
g	1	0	1	1	0	0	0	0
h	0	1	1	1	0	0	0	0

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

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

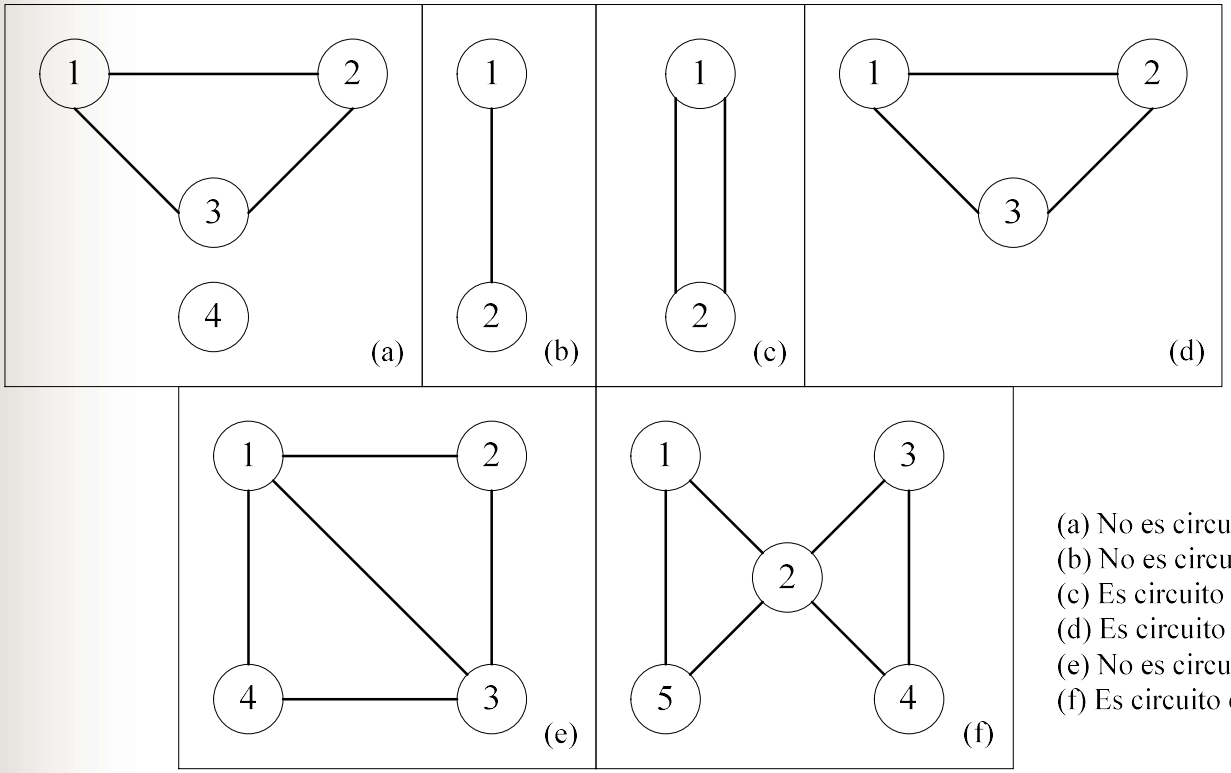
Circuito de Euler o Euleriano

- Un circuito que contiene todas las aristas de *G* recibe el nombre de **circuito de Euler o euleriano**.
  - Trayectoria o camino que empieza y termina en el mismo vértice y recorre cada arista exactamente una vez.
  - Un grafo que admite un circuito euleriano se denomina **grafo euleriano**.
  - Un grafo no euleriano que admite un camino euleriano se denomina **grafo semi-euleriano**.
- **Teorema.** El grafo *G* contiene un circuito euleriano sí y solo sí:
  - *G* es conexo.
  - Todos los vértices de *G* tienen grado par.
  - Un grafo dirigido conexo es euleriano si y solo si para cada vértice el grado interno coincide con el grado externo.

Trayectoria de Euler

- Una **trayectoria o camino de Euler o euleriana** en un grafo *G* es un camino simple y abierto que recorre todas las aristas del grafo una sola vez.
- **Teorema.** Sea *G* un grafo conexo cuya suma de los grados de todos sus vértices es par y hay un número par de vértices impares.
  - Si el número de vértices impares es mayor que dos, el grafo no se puede recorrer (pasando dos veces por alguna arista).
  - Si el número de vértices impares es cero, el grafo se puede recorrer. Podemos además elegir por qué vértice empezar, y el camino siempre será cerrado (termina donde empezó).
  - Si el número de vértices impares es dos, el grafo se puede recorrer, pero el camino ha de empezar en uno de los dos vértices impares y terminar en el otro.

Euler



(a) No es circuito euleriano, no hay trayectoria euleriana
(b) No es circuito euleriano, hay trayectoria euleriana
(c) Es circuito euleriano, hay trayectoria euleriana
(d) Es circuito euleriano, hay trayectoria euleriana
(e) No es circuito euleriano, hay trayectoria euleriana
(f) Es circuito euleriano, hay trayectoria euleriana

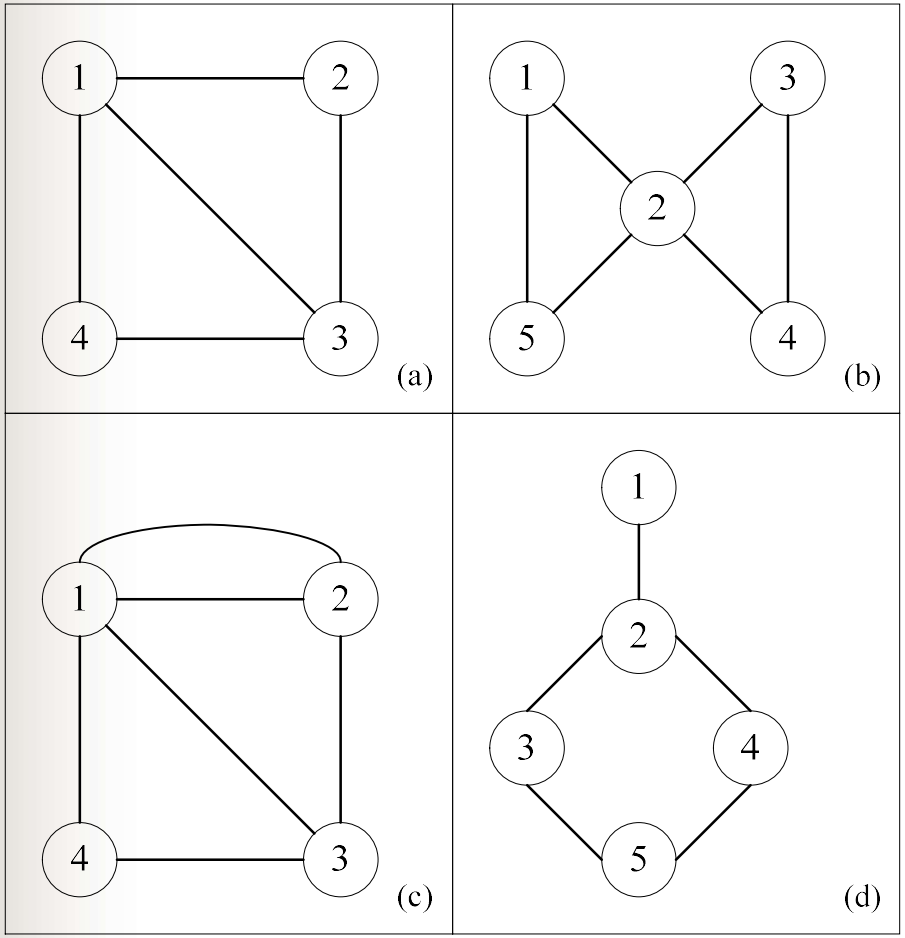
Ciclo de Hamilton o Hamiltoniano

- Un ciclo de Hamilton o hamiltoniano es un ciclo que contiene todos los vértices del grafo G y recorre cada vértice una sola vez.
  - Trayectoria o camino que empieza y termina en el mismo vértice y pasa por cada vértice exactamente una vez.
  - Un grafo que admite un circuito euleriano se denomina **grafo hamiltoniano**.
  - Un grafo no hamiltoniano que admite un camino hamiltoniano se denomina **grafo semi- hamiltoniano**.
- **Teorema.** El grafo G contiene un ciclo hamiltoniano sí y solo sí:
  - G es conexo con n vértices y m aristas, donde cada vértice tiene un grado $\geq n/2$.
  - $m \geq 1/2\ (n^2 - 3n + 6)$

Trayectoria de Hamilton o Hamiltoniana

- Una **trayectoria o camino de Hamilton o hamiltoniana** en un grafo *G* es un camino elemental y abierto que contiene todos los vértices del grafo y los recorre exactamente una vez.

Hamilton



(a) Circuito hamiltoniano, hay trayectoria hamiltoniana
(b) No es circuito hamiltoniano, hay trayectoria hamiltoniana
(c) Es circuito hamiltoniano, hay trayectoria hamiltoniana
(d) No es circuito hamiltoniano, hay trayectoria hamiltoniana

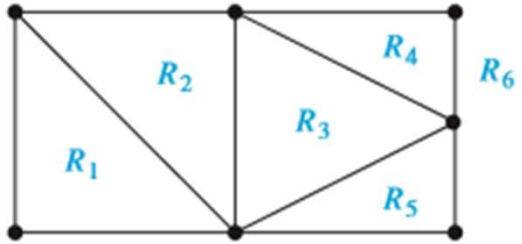
Grafos planares o planos

- Un grafo planar o plano, es una representación de un grafo en la cual ninguna de sus aristas interseca con otra (no se cruzan).

- Aplicaciones:
  - Diseño de circuitos, para que ninguna conexión choque.
  - Diseño de caminos, para conectar ciudades sin que choquen caminos.

Grafos planares o planos (cont.)

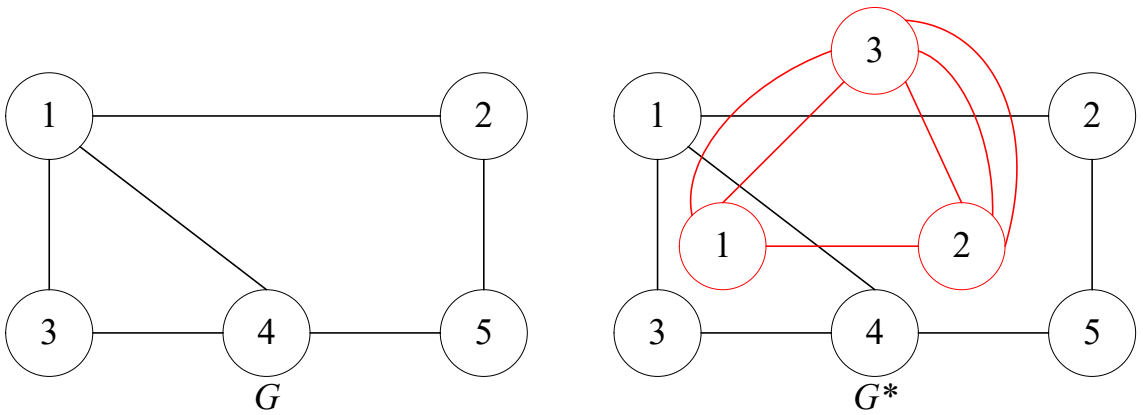
- G es un grafo planar simple conexo con:
  - E aristas
  - V vértices
  - R regiones



- Cumple $R = |E| - |V| + 2$ (Teorema de Euler, 1752).
- Corolarios
  - Si $V \geq 3$, entonces $E \leq 3V - 6$.
  - El grado de vértice de G no es mayor que 5.
  - Si $V \geq 3$ y no hay circuitos de 3, entonces $E \leq 2V - 4$.

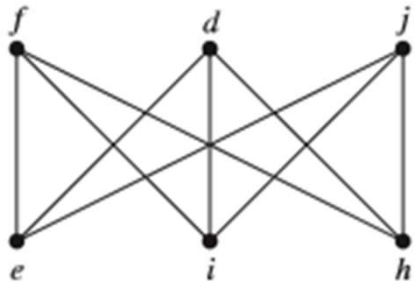
Grafos planares o planos (cont.)

- Un **grafo dual** $G^* = (V^*,E^*)$ se puede construir de un grafo plano G = (V,E) de la siguiente manera:
  - Introducir un vértice al grafo G^* por cada región $r \in R$ del grafo $G \rightarrow r \in V^*$.
  - Dibujar tantas aristas entre los vértices $r_1,r_2 \in V^*$ como aristas que comparten las regiones $r_1,r_2 \in R$ del grafo G.

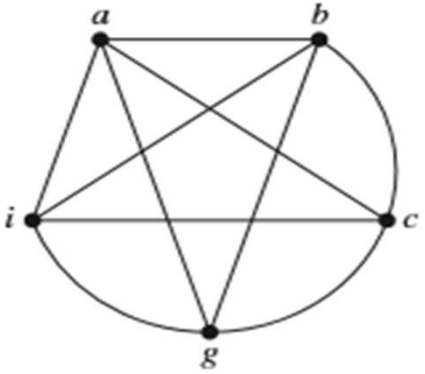


Grafos planares o planos (cont.)

- Teorema de Kuratowski
  - Este teorema dice que un grafo es no planar si contiene un subgrafo homeomorfo a $K_{3,3}$ o K_5.



(c) $K_{3,3}$



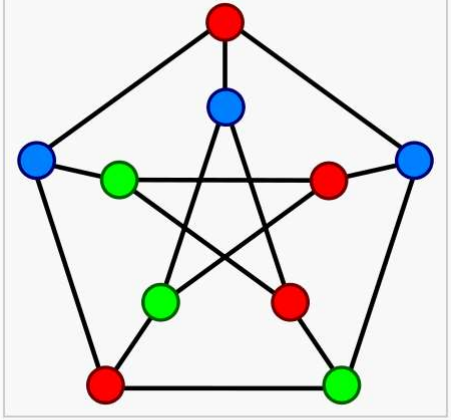
K_5

Grafos Coloreados

- La **coloración de grafos** es un caso especial de etiquetado de grafos; es una asignación de etiquetas llamadas **colores** a elementos del grafo.
  - Una coloración de los vértices de un grafo tal que ningún vértice adyacente comparta el mismo color es llamado vértice coloración.
  - Una coloración de arista asigna colores a cada arista tal que aristas adyacentes no compartan el mismo color.
  - Una coloración de caras de un grafo plano a la asignación de un color a cada cara o región tal que caras que compartan una frontera común tengan colores diferentes.

Grafos Coloreados (cont.)

- La convención de usar colores se origina de la coloración de países de un mapa, donde cada cara es literalmente coloreada.
- En general se puede usar un conjunto finito como conjunto de colores.
  - La naturaleza del problema de coloración depende del número de colores pero no sobre cuales son.



Una coloración de vértices para el grafo de Petersen utilizando tres colores, el número mínimo posible.

Grafos Coloreados (cont.)

- La **vértice coloración** (o simplemente coloración) es la asignación de los vértices de un grafo con colores tal que dos vértices que compartan la misma arista tengan colores diferentes.
  - Un grafo con bucles no puede ser coloreado, y solo se consideran grafos simples.

Grafos Coloreados (cont.)

- Una coloración que usa a lo más *k* colores se llama *k*-coloración (propia).
  - El menor número de colores necesarios para colorear un grafo *G* se llama **número cromático** y se denota como $\chi(G)$.
  - Un grafo que puede ser asignada una *k*-coloración (propia) es *k*-coloreable y es *k*-cromático si su número cromático es exactamente *k*.

Grafos Coloreados (cont.)

- Un subconjunto de vértices asignados con el mismo color se llama una clase de color.
  - Cada clase forma un conjunto independiente.
  - Una *k*-coloración es lo mismo que una partición del conjunto de vértices en *k* conjuntos independientes, y los términos *k*-partito y *k*-coloreable tienen el mismo significado.

Grafos Coloreados (cont.)

- El **polinomio cromático** cuenta el número de maneras en las cuales puede ser coloreado un grafo usando no más que un número de colores dado.
- El polinomio cromático es una función $P(G,t)$ que cuenta el número de t-coloraciones de G.
  - Como el nombre lo indica para un grafo G la función es un polinomio en t.

Polinomios cromáticos de algunos grafos.

Triángulo K_3	$t(t-1)(t-2)$
Grafo completo K_n	$t(t-1)(t-2)\cdots(t-(n-1))$
Árbol con n vértices	$t(t-1)^{n-1}$
Ciclo C_n	$(t-1)^n+(-1)^n(t-1)$
Grafo de Petersen	$t(t-1)(t-2)(t^7-12t^6+67t^5-230t^4+529t^3-814t^2+775t-352)$

Grafos Coloreados (cont.)

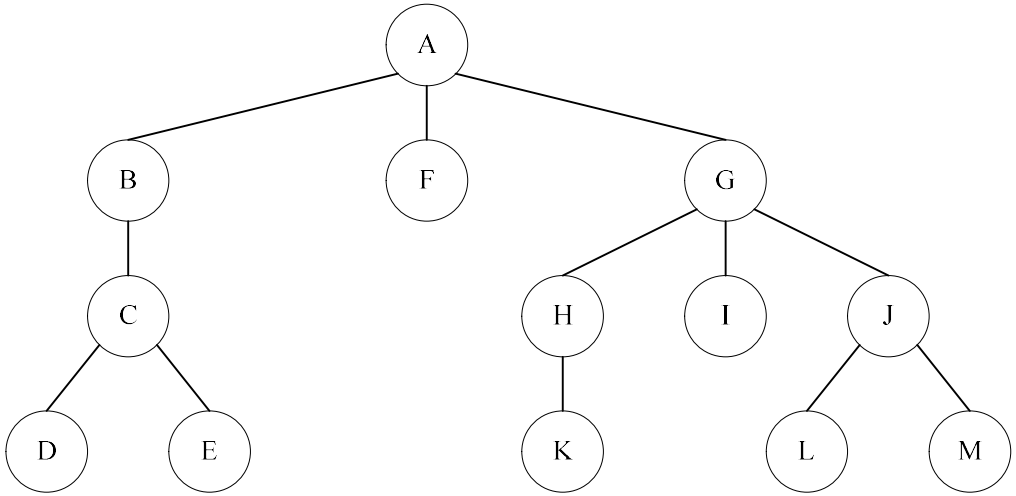
- Una **arista coloración** de un grafo, es una coloración de las aristas, denotada como la asignación de colores a aristas tal que aristas incidentes tengan un color distinto.
  - Una arista coloración con k colores es llamada k-arista-coloración y es equivalente al problema de particionar el conjunto de aristas en k emparejamientos.
- El menor número de colores necesarios para un arista coloración de un grafo G es el índice cromático o número cromático de aristas.
  - Una coloración Tait es una 3-arista-coloración de un grafo cúbico. El teorema de los cuatro colores es equivalente a que cada grafo cúbico sin puentes admite una coloración Tait.

Grafos Coloreados (cont.)

- **Teorema de los cuatro colores**
  - Desarrollado por Appel y Haken en 1976.
  - Basta con cuatro colores para colorear un mapa de tal modo que los países vecinos tengan colores distintos.

Árboles
Terminología

- Padre
- Hijo
- Hermanos
- Ancestro
- Descendiente
- Hoja
- Interno
- Subárbol



Árboles

- Un **árbol** es un grafo simple, conexo y acíclico (sin ciclos) en el que todo par de vértices están unidos por un único camino simple.
- Un **bosque** es un grafo acíclico, es decir, es una unión disjunta de árboles.
- Una **hoja** en un grafo es un vértice de grado 1.
- Un **nivel de un nodo del árbol (altura)** es el nivel de su padre más uno. Por definición, la raíz del árbol tiene nivel 0.
- Un **árbol generador** de un grafo G es un subgrafo generador de *G* que es un árbol.

Árboles (cont.)

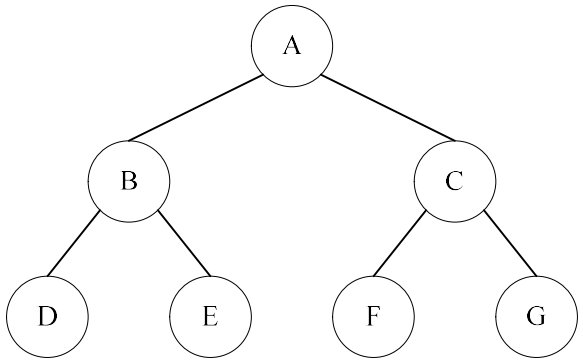
- Si el grafo G es un **árbol** se cumple:
  - Todo par de vértices de G está unido por un único camino simple.
  - Tiene un total de $n - 1$ aristas.
  - G es conexo y si se le quita alguna arista deja de ser conexo.
  - G es acíclico (no tiene ciclos) y si se añade alguna arista se forma un ciclo.
- Las condiciones anteriores son todas equivalentes, es decir, si se cumple una de ellas otras también se cumplen.

Árboles (cont.)

- Propiedades de los árboles:
  - Todo árbol es a su vez un grafo con un conjunto numerable de vértices es un grafo plano.
  - Todo grafo conexo G admite un árbol de expansión, que es un árbol que contiene cada vértice de G y cuyas aristas son aristas de G.
  - Todo árbol k-ario completo de altura h tiene k^h hojas.
- Las condiciones anteriores son todas equivalentes, es decir, si se cumple una de ellas otras también se cumplen.

Árboles (cont.)

- Un árbol raíz es un árbol *n*-ario si cada vértice interno no tiene más de *n* hijos.
- Un árbol es llamado *n*-ario completo si cada vértice interno tiene exactamente *n* hijos.
  - Ejemplo: Un árbol *n*-ario con $n = 2$ es un árbol binario.



Árboles (cont.)

- Contar los vértices en un árbol *n*-ario lleno.
  - La *n* significa que esa es la cantidad máxima de hijos que cada nodo padre puede tener en un árbol enraizado.
  - Que sea lleno significa que cada nodo es una hoja o posee *n* hijos.
- Un árbol con *n* vértices tiene *n*-1 aristas.
  - Es un grafo conexo, por ende todos sus vértices estarán unidos, de ahí que hayan *n*-1 aristas.
- Un árbol *n*-ario lleno con *i* vértices internos contiene $n = m*i + 1$ vértices.
  - Cada vértice, excepto la raíz, es el hijo de un vértice interno. Cada uno de los *i* vértices internos tiene *m* hijos, hay $m*i$ vértices en el árbol excluyendo la raíz.
  - Por ende, el árbol contiene $n = mi + 1$ vértices, siendo *n* el numero total de vértices en el árbol.

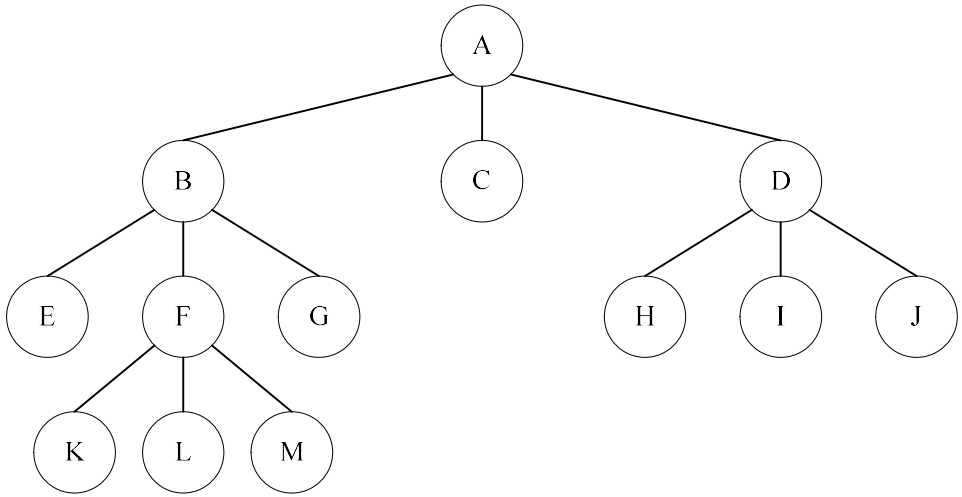
Árboles (cont.)

- Un árbol n-ario lleno con:
  - n vértices tiene $i = (n - 1)/m$ vértices internos y $l = [(m - 1)n + 1]/m$ hojas.
  - i vértices internos tiene $n = mi + 1$ vértices y $l = (m - 1)i + 1$ hojas.
  - l hojas tiene $n = (ml - 1)/(m - 1)$ vértices y $i = (l - 1)/(m - 1)$ vértices internos.

- Donde n representa la cantidad de vértices, i es la cantidad de vértices internos y l el número de hojas en el árbol.
  - Una vez que uno de n, i o l es conocido, se aplica la propiedad anterior para hallar los otros dos mediante la cantidad ya encontrada.

Árboles (cont.)

- Un árbol 3-ario lleno con:
  - $m = 3$ hijos por nodo y $n = 13$ vértices en total
  - $i = (13 – 1)/3 = 4$ vértices internos
  - $l = [(3 – 1)13 + 1]/3 = 9$ hojas



Árboles (cont.)

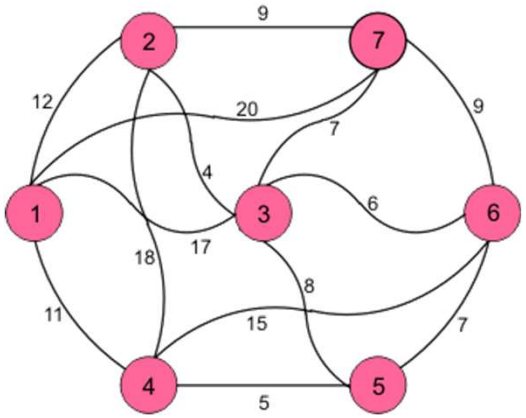
- Un árbol *n*-ario balanceado:
  - Es deseable usualmente usar árboles que estén balanceados, para que así los subárboles de cada vértice contengan aproximadamente el mismo tamaño.
  - El nivel de un vértice *v* en un árbol enraizado es el tamaño del camino único desde la raíz hasta este vértice.
  - El nivel de una raíz es 0.
  - La altura de un árbol enraizado es el máximo de niveles de los vértices, o sea el camino más largo desde la raíz hasta un vértice.

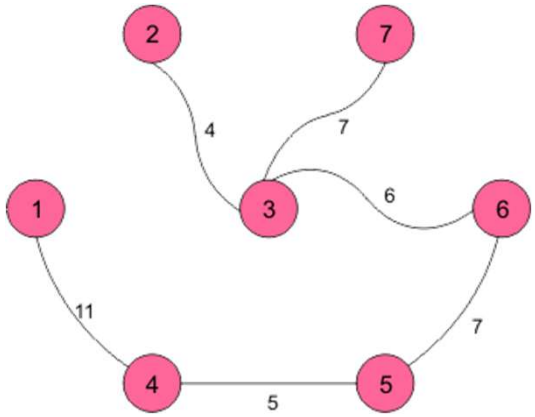
Árboles (cont.)

- Si un árbol n-ario de altura h tiene l hojas, entonces $h \geq \lceil \log_n l \rceil$.
- Si el árbol n-ario es lleno y balanceado, entonces $h = \lceil \log_n l \rceil$.
- Hay como máximo n^h hojas en un árbol n-ario alto h.

Árbol de Expansión

- Dado un grafo *G*, el árbol de expansión es un subgrafo de *G* que conecta todos los nodos de este mismo sin formar un ciclo.
  - Los árboles de expansión pueden ser árboles de profundidad o anchura, y pueden obtenerse por la búsqueda en profundidad o anchura, respectivamente.





Algoritmos de Grafos Dirigidos

- Algoritmos de determinación de los caminos más cortos:
  - Algoritmo de Dijkstra.
  - Algoritmo de Floyd-Warshall.
- Algoritmos de recorrido o búsqueda:
  - Algoritmo de búsqueda en anchura o amplitud.
  - Algoritmo de búsqueda en profundidad.
  - Bosques abarcadores.

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra

- El algoritmo de Dijkstra, también llamado **algoritmo de caminos mínimos**, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo dirigido y con pesos en cada arco.
- Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)

- La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices.
- Cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene.
- El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de costo negativo.

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)

- Descripción detallada:
  - Sea $G=(V,E)$ un grafo dirigido y etiquetado.
  - Sean los vértices $a \in V$ y $z \in V$; a es el vértice de origen y z el vértice de destino.
  - Sea un conjunto $C \subset V$, que contiene los vértices de V cuyo camino más corto desde a todavía no se conoce.
  - Sea un vector D, con tantas dimensiones como elementos tiene V, y que "guarda" las distancias entre a y cada uno de los vértices de V.
  - Sea, finalmente, otro vector, P, con las mismas dimensiones que D, y que conserva la información sobre qué vértice precede a cada uno de los vértices en el camino.

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)

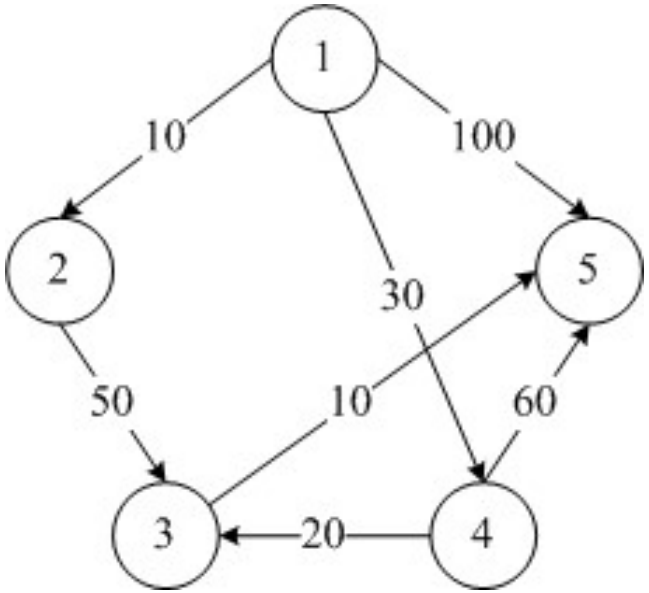
- Descripción detallada:
  - El algoritmo para determinar el camino de longitud mínima entre los vértices a y z es: $C \leftarrow V$.
    1. Para todo vértice $i \in C$, $i \neq a$, se establece $D_i \leftarrow \infty$; $D_a \leftarrow 0$.
    2. Para todo vértice $i \in C$ se establece $P_i = a$.
    3. Se obtiene el vértice $s \in C$ tal que no existe otro vértice $w \in C$ tal que $D_w < D_s$.
       - Si $s = z$ entonces se ha terminado el algoritmo.
    4. Se elimina de C el vértice s: $C \leftarrow C - \{s\}$.
    5. Para cada arista $e \in E$ de longitud l, que une el vértice s con algún otro vértice $t \in C$,
       - Si $l + D_s < D_t$, entonces:
         - Se establece $D_t \leftarrow l + D_s$.
         - Se establece $P_t \leftarrow s$.
    6. Se regresa al paso 4.

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)

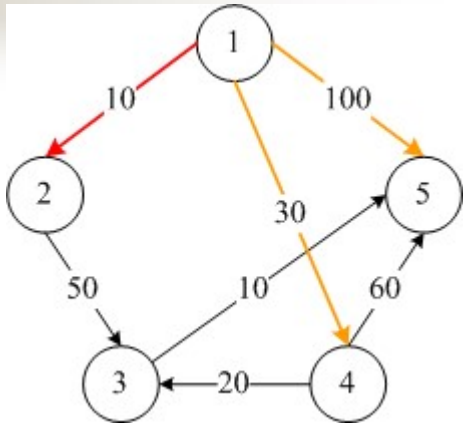
- Al terminar este algoritmo, en D_z estará guardada la distancia mínima entre *a* y *z*.
- Por otro lado, mediante el vector *P* se puede obtener el camino mínimo: en P_z estará *y*, el vértice que precede a *z* en el camino mínimo; en P_y estará el que precede a *y*, y así sucesivamente, hasta llegar a *ESTADO DE ENLACE*.
- Aplicación Web del algoritmo:
  - http://neo.lcc.uma.es/evirtual/cdd/applets/distancia%20corta/Example2.html.

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)

- Ejemplo:
  - Encontrar los caminos más cortos entre el vértice 1 y todos los demás del siguiente grafo dirigido.

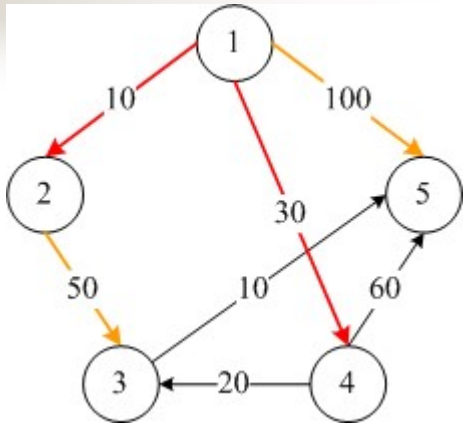


Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)



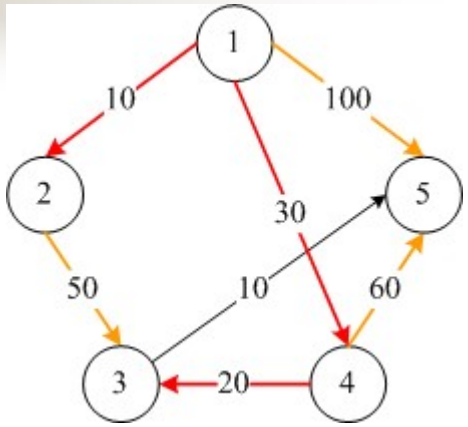
Iteración	***S***	***w***	***D*[2]**	***D*[3]**	***D*[4]**	***D*[5]**
Inicial	{1}	---	10	∞	30	100

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)



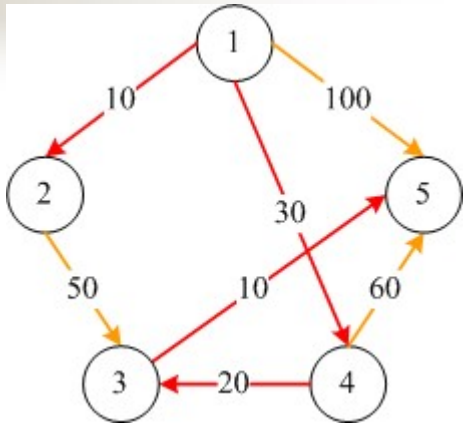
Iteración	*S*	*w*	*D*[2]	*D*[3]	*D*[4]	*D*[5]
Inicial	{1}	---	10	∞	30	100
1	{1,2}	2	10	60	30	100

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)



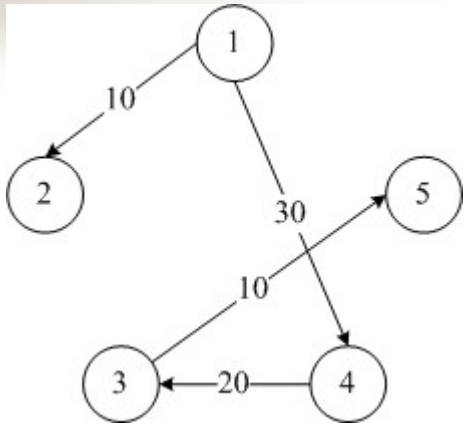
Iteración	***S***	***w***	***D*[2]**	***D*[3]**	***D*[4]**	***D*[5]**
Inicial	{1}	---	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)



Iteración	***S***	***w***	***D*[2]**	***D*[3]**	***D*[4]**	***D*[5]**
Inicial	{1}	---	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)



Iteración	*S*	*w*	*D*[2]	*D*[3]	*D*[4]	*D*[5]
Inicial	{1}	---	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

Algoritmos de Grafos Dirigidos – Algoritmo de Dijkstra (cont.)

- Pseudocódigo del algoritmo:

```
DIJKSTRA (Grafo G, nodo_fuente s)
    // inicializamos todos los nodos del grafo. La distancia de cada nodo es infinita
    // y los padres son NULL
    for u ∈ V[G] do
        distancia[u] = INFINITO
        padre[u] = NULL
    distancia[s] = 0
    //encolamos todos los nodos del grafo
    Encolar (cola, V[G])
    mientras cola != 0 do
       // OJO: Se extrae el nodo que tiene distancia mínima y se conserva la condición
       // de Cola de prioridad
        u = extraer_minimo(cola)
        for v ∈ adyacencia[u] do
            if distancia[v] > distancia[u] + peso(u,v) do
                distancia[v] = distancia[u] + peso(u,v)
                padre[v] = u
```

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd

- El **algoritmo de Floyd-Warshall** intenta resolver el problema de encontrar el camino más corto entre todos los pares de nodos o vértices de un grafo.
- Esto es similar a construir una tabla con todas las distancias mínimas entre pares de ciudades de un mapa, indicando la ruta a seguir para ir de la primera ciudad a la segunda.

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- Esto puede verse de la siguiente manera:
  - Sea $G=(V,E)$ un grafo en el cual cada arco tiene asociado un costo no negativo. El problema es hallar para cualquier par de vértices (v,w) el camino más corto de v a w.
  - $G=(V,E)$, $V=\{1,...,n\}$ y $C[i,j]$ es el costo del arco que va de i a j.
  - El algoritmo calcula la serie de matrices

$$A_0[i,j] = \begin{cases} 0 & \text{si } i = j \\ C[i,j] & \text{si } i \neq j \end{cases}$$

$$A_k[i,j] = min(A_{k-1}[i,j], A_{k-1}[i,k] + A_{k-1}[k,j])$$

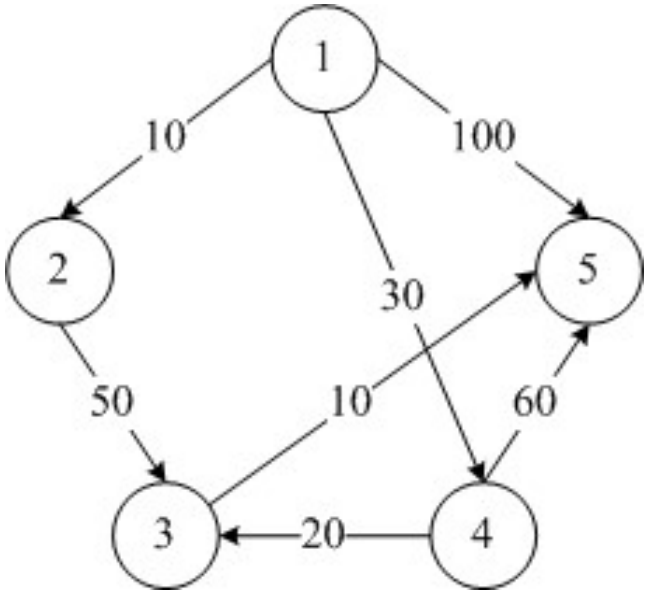
  - $A_k[i,j]$ significa el costo del camino más corto que va de i a j y que no pasa por algún vértice mayor que k.
  - El objetivo es calcular $A_n[i,j]$.

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- El algoritmo se modifica para agregar una matriz que guarde los caminos más económicos entre los vértices.
- Al algoritmo se le agrega una matriz P, donde $P[i,j]$ tiene el vértice k que permitió encontrar el valor más pequeño de $A[i,j]$.
- Si $P[i,j] = 0$, e camino más corto de i a j es directo, siguiendo el arco entre ambos.
- La versión modificada del algoritmo almacenará los vértices intermedios apropiados en P.

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- Ejemplo:
  - Encontrar los caminos más cortos entre todos los vértices del siguiente grafo dirigido.



Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 10 & \infty & 30 & 100 \\
\infty & 0 & 50 & \infty & \infty \\
\infty & \infty & 0 & \infty & 10 \\
\infty & \infty & 20 & 0 & 60 \\
\infty & \infty & \infty & \infty & 0
\end{bmatrix}
\end{array}
$$

$A_0[i,j]$

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

P

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

$$A_0[i,j]:\quad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 10 & \infty & 30 & 100 \\ \infty & 0 & 50 & \infty & \infty \\ \infty & \infty & 0 & \infty & 10 \\ \infty & \infty & 20 & 0 & 60 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$A_1[i,j]:\quad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 10 & \infty & 30 & 100 \\ \infty & 0 & 50 & \infty & \infty \\ \infty & \infty & 0 & \infty & 10 \\ \infty & \infty & 20 & 0 & 60 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$P:\quad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 10 & \infty & 30 & 100 \\ \infty & 0 & 50 & \infty & \infty \\ \infty & \infty & 0 & \infty & 10 \\ \infty & \infty & 20 & 0 & 60 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}
\end{array}
\qquad A_1[i,j]
$$

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 10 & 60 & 30 & 100 \\ \infty & 0 & 50 & \infty & \infty \\ \infty & \infty & 0 & \infty & 10 \\ \infty & \infty & 20 & 0 & 60 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}
\end{array}
\qquad A_2[i,j]
$$

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\end{array}
\qquad P
$$

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

$A_2[i,j]$

	1	2	3	4	5
1	0	10	60	30	100
2	∞	0	50	∞	∞
3	∞	∞	0	∞	10
4	∞	∞	20	0	60
5	∞	∞	∞	∞	0

$A_3[i,j]$

	1	2	3	4	5
1	0	10	60	30	70
2	∞	0	50	∞	60
3	∞	∞	0	∞	10
4	∞	∞	20	0	30
5	∞	∞	∞	∞	0

P

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

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

$A_3[i,j]$

	1	2	3	4	5
1	0	10	60	30	70
2	∞	0	50	∞	60
3	∞	∞	0	∞	10
4	∞	∞	20	0	30
5	∞	∞	∞	∞	0

$A_4[i,j]$

	1	2	3	4	5
1	0	10	50	30	60
2	∞	0	50	∞	60
3	∞	∞	0	∞	10
4	∞	∞	20	0	30
5	∞	∞	∞	∞	0

P

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

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

$A_4[i,j]$

	1	2	3	4	5
1	0	10	50	30	60
2	∞	0	50	∞	60
3	∞	∞	0	∞	10
4	∞	∞	20	0	30
5	∞	∞	∞	∞	0

$A_5[i,j]$

	1	2	3	4	5
1	0	10	50	30	60
2	∞	0	50	∞	60
3	∞	∞	0	∞	10
4	∞	∞	20	0	30
5	∞	∞	∞	∞	0

P

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

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 10 & 50 & 30 & 60 \\
\infty & 0 & 50 & \infty & 60 \\
\infty & \infty & 0 & \infty & 10 \\
\infty & \infty & 20 & 0 & 30 \\
\infty & \infty & \infty & \infty & 0
\end{bmatrix}
\end{array}
$$

A

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 4 & 0 & 4 \\
0 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

P

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- Pseudocódigo del algoritmo:

```
FLOYD-WARSHALL (Grafo G)
      for i = 1 to V do
            for j = 1 to V do
                  D[i,j] = G[i,j]
                  P[i,j] = 0
      for i = 1 to V do
            D[i,i] = 0
      for k = 1 to V do
            for i = 1 to V do
                  for j = 1 to V do
                        if D[i,j] > D[i,k]+D[k,j] do
                              D[i,j] = D[i,k]+D[k,j]
                              P[i,j] = k
```

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- Pseudocódigo para imprimir los vértices intermedios del vértice *i* hasta el vértice *j*:

```
CAMINO (nodos i,j)
        k = P[i,j]
        if k = 0 do
                return
        CAMINO(i,k)
        imprimir k
        CAMINO(k,j)
```

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- En algunos casos podría ser importante saber sólo si existe un camino de longitud mayor o igual a 1 que vaya desde el vértice *i* al vértice *j*.
- El algoritmo de Floyd puede especializarse para este problema; el algoritmo resultante, que antecede al de Floyd, se conoce como el algoritmo de Warshall.
- Con el algoritmo de Warshall se desea obtener la matriz *A* tal que $A[i,j] = 1$ si hay un camino de longitud igual o mayor que 1, y 0 en caso contrario.
  - Esta matriz *A* se conoce como **cerradura transitiva** de la matriz de adyacencia.

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

Cerradura Transitiva

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- Pseudocódigo del algoritmo de Warshall:

```
WARSHALL (Matriz de adyacencia C)
        for i = 1 to V do
                for j = 1 to V do
                        A[i,j] = C[i,j]
        for k = 1 to V do
                for i = 1 to V do
                        for j = 1 to V do
                                if A[i,j] = 0 do
                                        A[i,j] = A[i,k] and A[k,j]
```

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

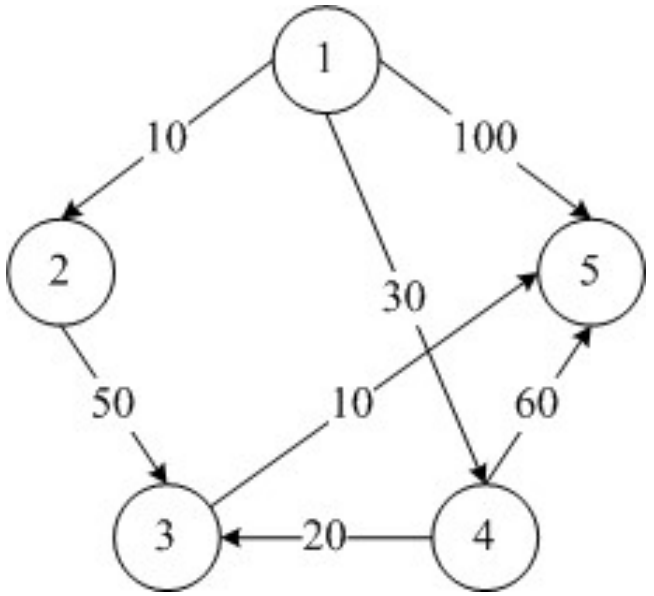
- El algoritmo de Floyd se utiliza, aparte de hallar los caminos más cortos entre todos los vértices, para determinar el vértice más central de un grafo dirigido.
- Para encontrar el centro de un grafo dirigido *G* se necesita:
  - La **excentricidad** de *v*, la cual es el valor máximo de las longitudes de los caminos más cortos de *w* a *v*.
- Después de encontrar la excentricidad de cada vértice, se obtiene el centro de *G*, el cual es el vértice de mínima excentricidad.
- Así, el centro de un grafo dirigido es un vértice más cercano al vértice más distante.

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

- Para encontrar el centro de un grafo dirigido *G* se hace lo siguiente:
  - Aplicar el algoritmo de Floyd para obtener la matriz de los caminos más cortos entre todos los pares.
  - Encontrar el costo máximo de cada columna *i*, esto da la excentricidad del vértice *i*.
  - Encontrar el vértice con excentricidad mínima, este es el centro del grafo *G*.

Algoritmos de Grafos Dirigidos – Algoritmo de Floyd (cont.)

	1	2	3	4	5
1	0	10	50	30	60
2	∞	0	50	∞	60
3	∞	∞	0	∞	10
4	∞	∞	20	0	30
5	∞	∞	∞	∞	0
max	∞	∞	∞	∞	60



Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura o Amplitud

- **Búsqueda en anchura o amplitud** (*BFS* o *Breadth-first search* en inglés) es un algoritmo para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles).
  - Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo.
  - A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.
- Su nombre se debe a que expande uniformemente la frontera entre lo descubierto y lo no descubierto. Llega a los nodos de distancia *k*, sólo tras haber llegado a todos los nodos a distancia *k*-1.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

- Formalmente, *BFS* es un algoritmo de **búsqueda sin información**, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución.
- El algoritmo no usa ninguna estrategia heurística.
- El peso de los arcos para ejecutar **BFS** debe de ser de IGUAL costo.
- Si las aristas tienen pesos negativos se aplica el algoritmo de Bellman-Ford en alguna de sus dos versiones.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

- Descripción detallada:
  - Dado un vértice fuente ***s***, se explora los vértices de *G* para "descubrir" todos los vértices alcanzables desde ***s***.
  - Se busca desde ***s*** a todos los vértices alcanzables.
  - Después produce un árbol **BF** con raíz en ***s*** y que contiene a todos los vértices alcanzables.
  - El camino desde ***s*** a cada vértice en este recorrido contiene el mínimo número de vértices. Es el camino más corto medido en número de vértices.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

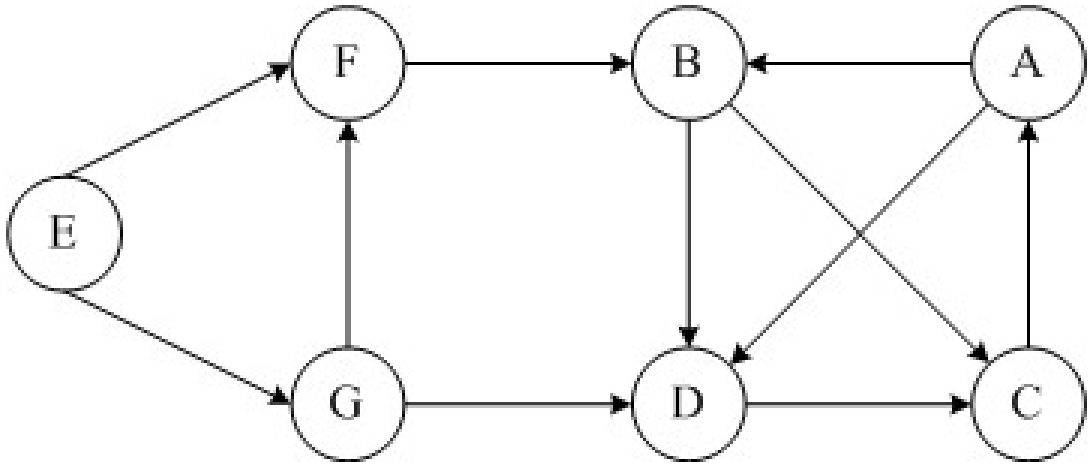
- Durante un recorrido en anchura, cuando se recorren ciertos arcos, llevan a vértices sin visitar.
- Los arcos que llevan a vértices nuevos se conocen como **arcos de árbol** y forman un **árbol abarcador en anchura** para el grafo dirigido dado.
  - **Arco de árbol.** Es el arco que forma parte del árbol.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

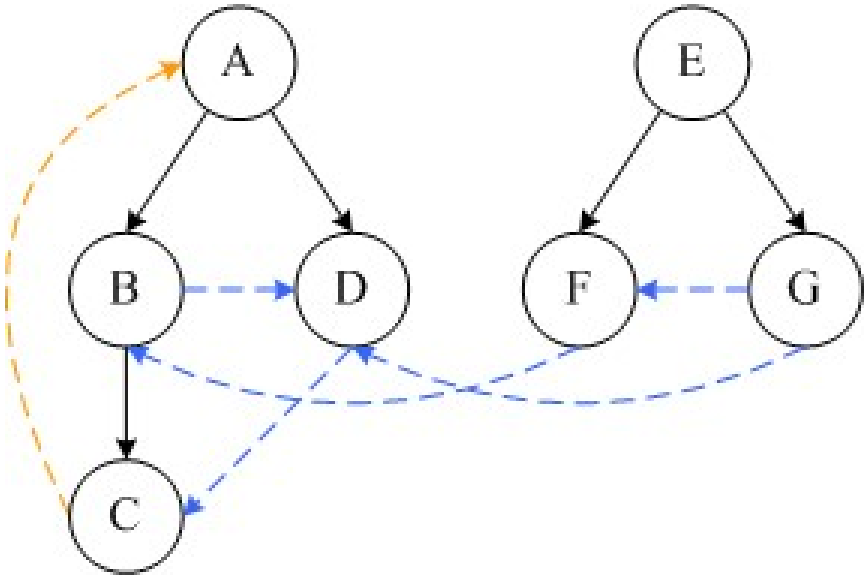
- Además de los arcos de árbol, existen dos tipos de arcos definidos por una búsqueda en anchura de un grafo dirigido, que se conocen como:
  - **Arco de retroceso:** Es el arco que va de un vértice *v* a un vértice *w* que es antecesor de *v* en el árbol abarcador. Un arco que va de un vértice hacia si mismo se considera un arco de retroceso.
  - **Arco cruzado:** Es el arco que va de un vértice *v* a un vértice *w* que no es ancestro ni descendiente.
- Todos los arcos forman un **bosque abarcador en anchura** para el grafo dirigido dado.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

- Ejemplo:
  - Realizar el recorrido en anchura (siga el orden alfabético) y encontrar el bosque abarcador del siguiente grafo dirigido.



Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)



Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

- Pseudocódigo del algoritmo:

```
BFS(grafo G, nodo_fuente s)
{
   // recorremos todos los vértices del grafo inicializándolos a NO_VISITADO,
   // distancia INFINITA y padre de cada nodo NULL
   for u ∈ V[G] do
   {
      estado[u] = NO_VISITADO;
      distancia[u] = INFINITO; /* distancia infinita si el nodo no es alcanzable */
      padre[u] = NULL;
   }
   estado[s] = VISITADO;
   distancia[s] = 0;
   Encolar(Q, s);
   while Q != 0 do
   {
      // extraemos el nodo u de la cola Q y exploramos todos sus nodos adyacentes
      u = extraer(Q);
      for  v ∈ adyacencia[u]  do
      {
         if estado[v] == NO_VISITADO then
         {
               estado[v] = VISITADO;
               distancia[v] = distancia[u] + 1;
               padre[v] = u;
               Encolar(Q, v);
         }
      }
   }
}
```

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Profundidad

- Un **recorrido en profundidad** (en inglés **DFS** - *Depth First Search*) es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme.
- Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto.
- Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

- Arcos DF:
  - Si en tiempo de descubrimiento de *u* tenemos el arco **(u,v)**:
    - i. Si el estado de *v* es NO_VISITADO, entonces *(u,v)* ∈ DF.
    - ii. Si el estado de *v* es VISITADO, entonces *(u,v)* es un **arco hacia atrás**.
    - iii. Si el estado de *v* es TERMINADO, entonces *(u,v)* es un **arco de cruce** o **arco hacia delante**. Será de cruce si $d[v]<d[u]$; y será hacia delante si $d[v]>d[v]$.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

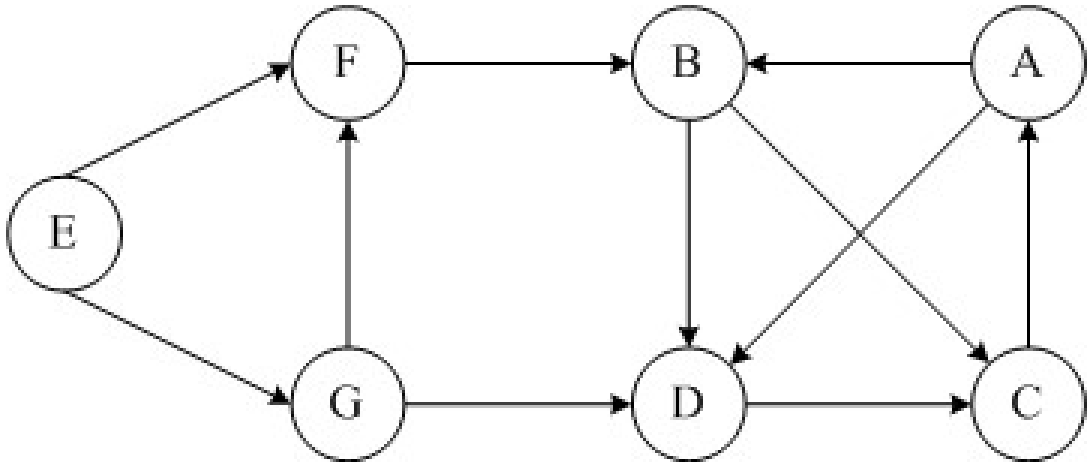
- Durante un recorrido en profundidad, cuando se recorren ciertos arcos, llevan a vértices sin visitar.
- Los arcos que llevan a vértices nuevos se conocen como **arcos de árbol** y forman un **árbol abarcador en profundidad** para el grafo dirigido dado.
  - **Arco de árbol.** Es el arco que forma parte del árbol.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

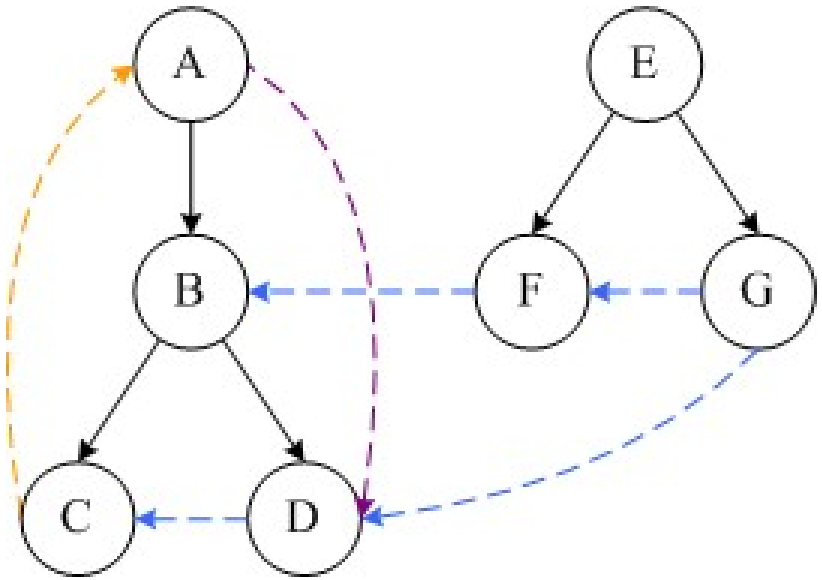
- Además de los arcos de árbol, existen tres tipos de arcos definidos por una búsqueda en profundidad de un grafo dirigido, que se conocen como:
  - **Arco de retroceso:** Es el arco que va de un vértice *v* a un vértice *w* que es antecesor de *v* en el árbol abarcador. Un arco que va de un vértice hacia si mismo se considera un arco de retroceso.
  - **Arco de avance:** Es el arco que va de un vértice *v* a un vértice *w* que es descendiente de *v* en el árbol abarcador.
  - **Arco cruzado:** Es el arco que va de un vértice *v* a un vértice *w* que no es ancestro ni descendiente.
- Todos los arcos forman un **bosque abarcador en profundidad** para el grafo dirigido dado.

Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

- Ejemplo:
  - Realizar el recorrido en profundidad (siga el orden alfabético) y encontrar el bosque abarcador del siguiente grafo dirigido.



Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)



Algoritmos de Grafos Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

- Pseudocódigo del algoritmo:

```
DFS(grafo G)
for each vertice u ∈ V[G] do
        estado[u]=NO_VISITADO
        padre[u]= NULL
tiempo=0
for each vertice u ∈ V[G] do
        if estado[u] = NO_VISITADO then
                DFS-Visitar(u)
```

```
DFS-Visitar(nodo u)
estado[u]=VISITADO
tiempo=tiempo+1
d[u]=tiempo
for each v ∈ Vecinos[u] do
        if estado[v]=NO_VISITADO then
                padre[v]=u
                DFS-Visitar(v)
estado[u]=TERMINADO
tiempo=tiempo+1
f[u]=tiempo
```

Algoritmos de Grafos No Dirigidos

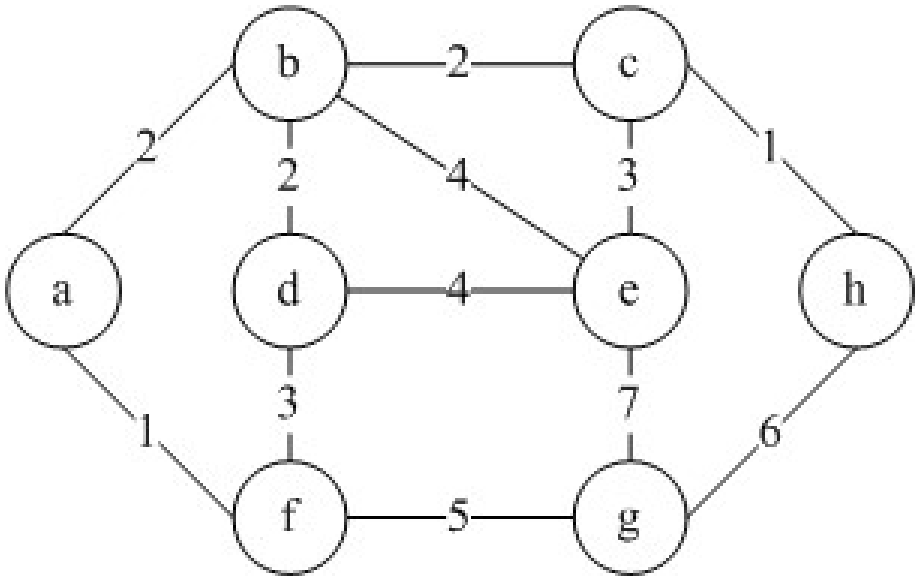
- Algoritmos de determinación de los caminos más cortos:
  - Algoritmo del camino más corto.
  - Algoritmo del vendedor ambulante.
- Algoritmos de árboles abarcadores de costo mínimo (árboles generadores mínimos):
  - Algoritmo de Prim.
  - Algoritmo de Kruskal.
- Algoritmos de recorrido o búsqueda:
  - Algoritmo de recorrido en pre-orden, orden y post-orden
  - Algoritmo de búsqueda en anchura o amplitud.
  - Algoritmo de búsqueda en profundidad.
  - Bosques abarcadores.

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto

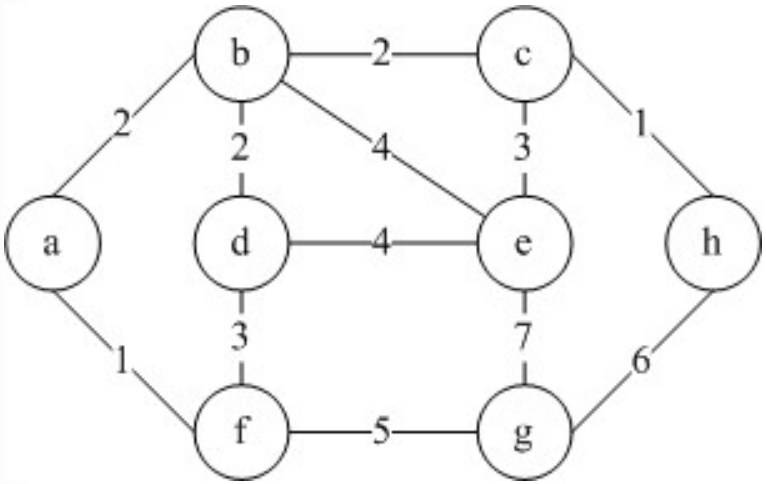
- Este algoritmo busca el camino más corto entre dos vértices.
- Recibe como entrada el grafo no dirigido G, el vértice inicial y el vértice final.
- El algoritmo es el siguiente:
  1. $D[a] = 0$, si $x \neq a \Rightarrow D[x] = \infty$. Se tiene el conjunto de vértices T.
  2. Si $z \notin T \Rightarrow$ terminar y $D[z]$ es la distancia más corta entre a y z.
  3. Escoja $v \in T$ donde $D[v]$ es el valor mínimo. $T = T - \{v\}$.
  4. Si $x \in T$ y es adyacente a $v \Rightarrow D[x] = \min\{D[x], D[v]+c(v,x)\}$.
  5. Pase al paso 2.

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)

- Ejemplo:
  - Encontrar el camino más corto entre los vértices *a* y *h*.



Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$$T = \{a, b, c, d, e, f, g, h\}$$

$$D[a] = 0$$

$$D[b] = \infty$$

$$D[c] = \infty$$

$$D[d] = \infty$$

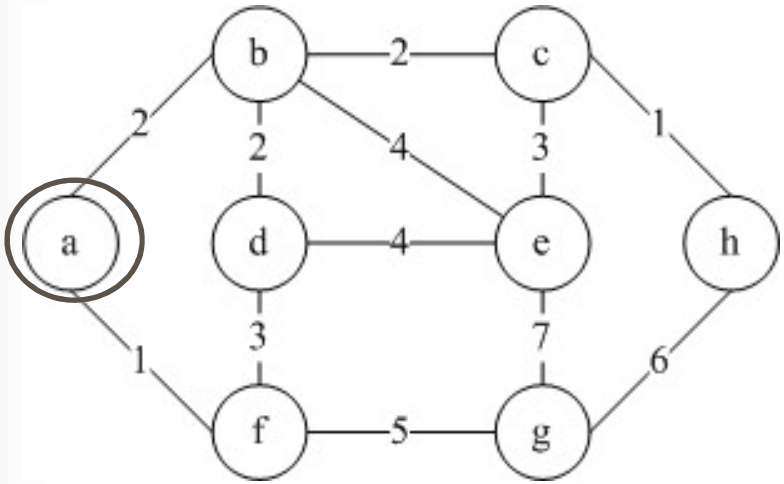
$$D[e] = \infty$$

$$D[f] = \infty$$

$$D[g] = \infty$$

$$D[h] = \infty$$

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$$T = \{b, c, d, e, f, g, h\}$$

$$D[a] = 0$$

$$D[b] = 2$$

$$D[c] = \infty$$

$$D[d] = \infty$$

$$D[e] = \infty$$

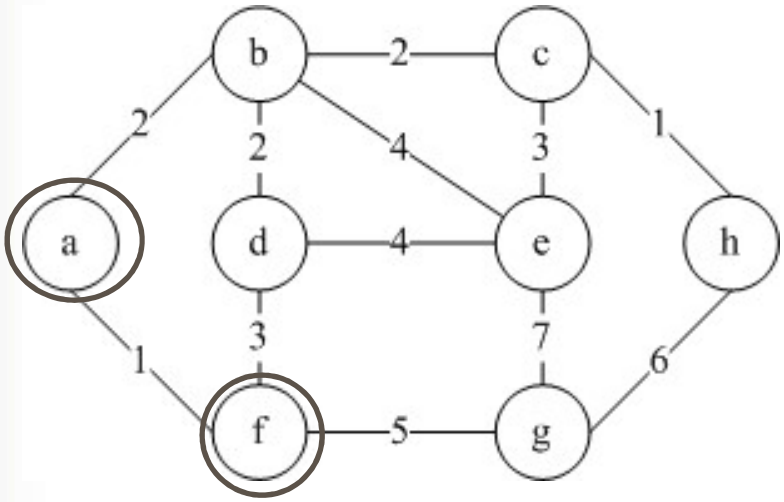
$$D[f] = 1$$

$$D[g] = \infty$$

$$D[h] = \infty$$

$$Adyacente\ \text{a}\ a \begin{cases} D[b] = \min\{\infty, 0+2\} = 2 \\ D[f] = \min\{\infty, 0+1\} = 1 \end{cases}$$

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$$T = \{b, c, d, e, g, h\}$$

$$D[a] = 0$$

$$D[b] = 2$$

$$D[c] = \infty$$

$$D[d] = 4$$

$$D[e] = \infty$$

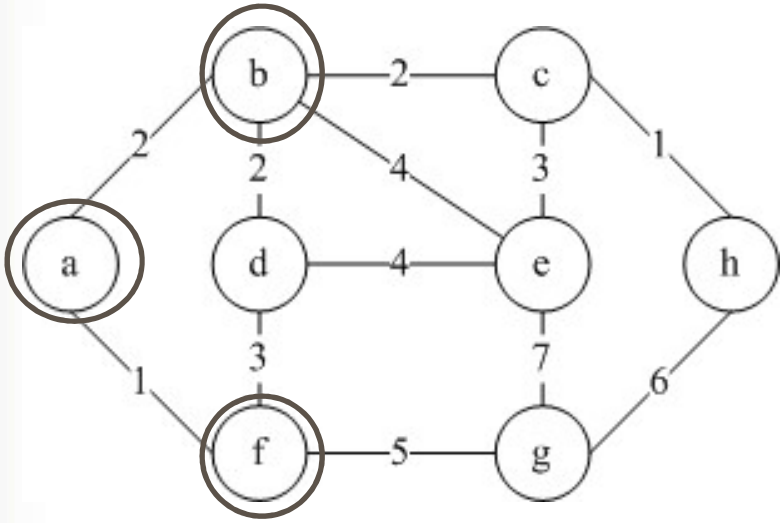
$$D[f] = 1$$

$$D[g] = 6$$

$$D[h] = \infty$$

$$Adyacente\ \text{a}\ f \begin{cases} D[d] = \min\{\infty, 1+3\} = 4 \\ D[g] = \min\{\infty, 1+5\} = 6 \end{cases}$$

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$$T = \{c, d, e, g, h\}$$

$$D[a] = 0$$

$$D[b] = 2$$

$$D[c] = 4$$

$$D[d] = 4$$

$$D[e] = 6$$

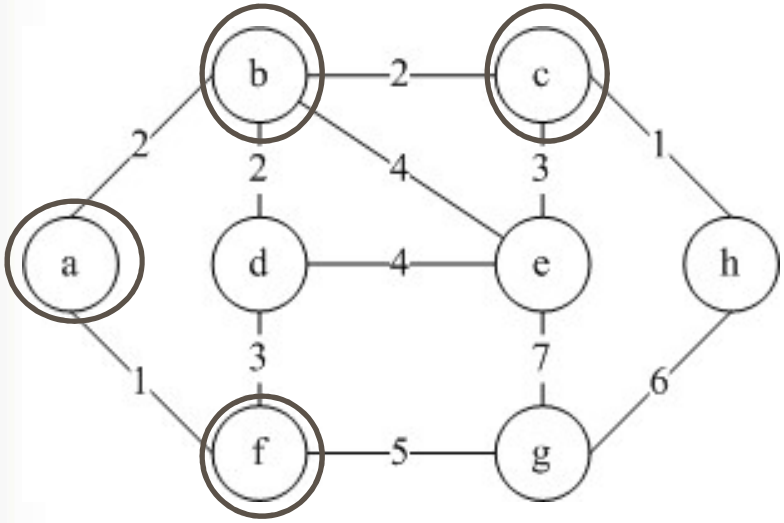
$$D[f] = 1$$

$$D[g] = 6$$

$$D[h] = \infty$$

$$\textit{Adyacente a } b \begin{cases} D[c] = \min\{\infty, 2+2\} = 4 \\ D[d] = \min\{4, 2+2\} = 4 \\ D[e] = \min\{\infty, 2+4\} = 6 \end{cases}$$

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$$T = \{d, e, g, h\}$$

$$D[a] = 0$$

$$D[b] = 2$$

$$D[c] = 4$$

$$D[d] = 4$$

$$D[e] = 6$$

$$D[f] = 1$$

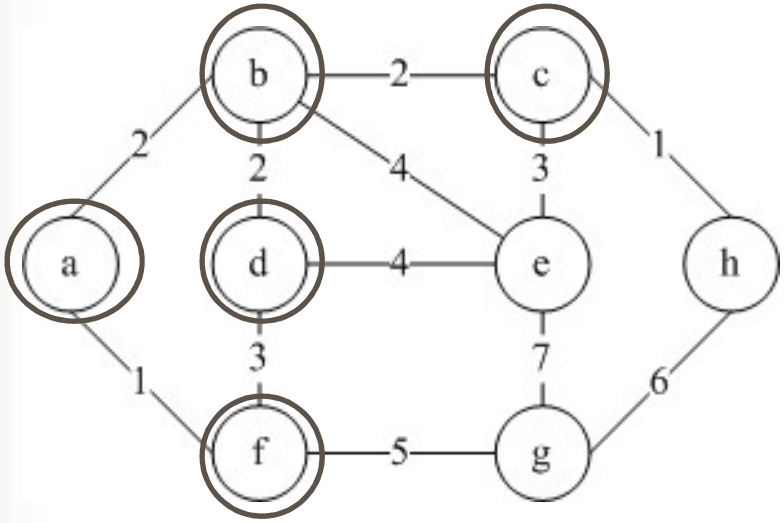
$$D[g] = 6$$

$$D[h] = 5$$

Adyacente a c

$$\begin{cases} D[e] = \min\{6, 4+3\} = 6 \\ D[h] = \min\{\infty, 4+1\} = 5 \end{cases}$$

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$$T = \{e, g, h\}$$

$$D[a] = 0$$

$$D[b] = 2$$

$$D[c] = 4$$

$$D[d] = 4$$

$$D[e] = 6$$

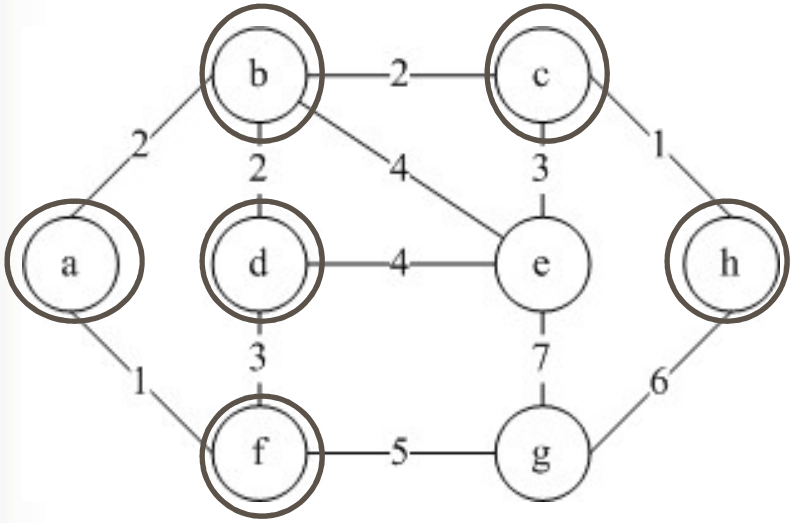
$$D[f] = 1$$

$$D[g] = 6$$

$$D[h] = 5$$

$Adyacente\ \text{a}\ d\ \ \{D[e] = \min\{6, 4+4\} = 6$

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$T = \{e, g\}$

$D[a] = 0$

$D[b] = 2$

$D[c] = 4$

$D[d] = 4$

$D[e] = 6$

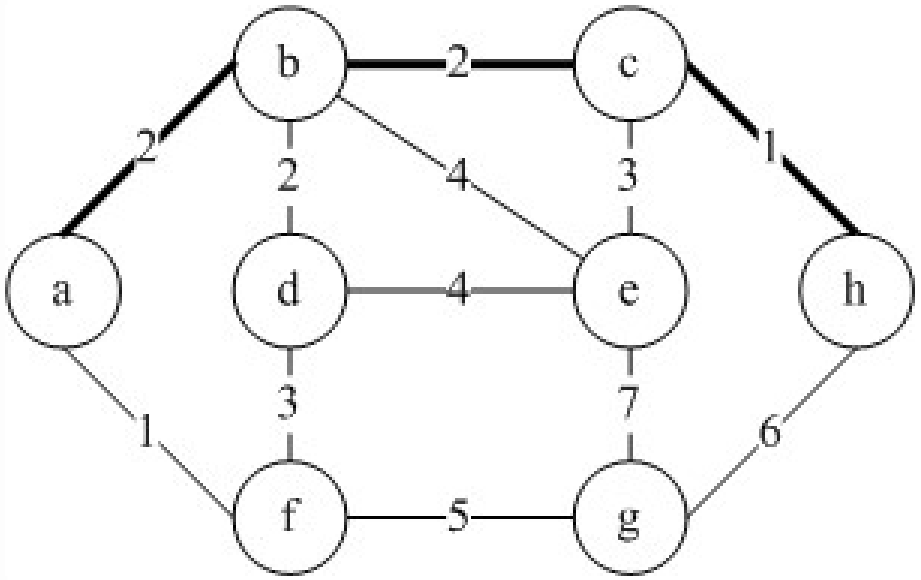
$D[f] = 1$

$D[g] = 6$

$D[h] = 5$

Adyacente a h $\{D[g] = \min\{6, 5+6\} = 6$

Algoritmos de Grafos No Dirigidos – Algoritmo del Camino Más Corto (cont.)



$T = \{e, g\}$

$D[a] = 0$

$D[b] = 2$

$D[c] = 4$

$D[d] = 4$

$D[e] = 6$

$D[f] = 1$

$D[g] = 6$

$D[h] = 5$

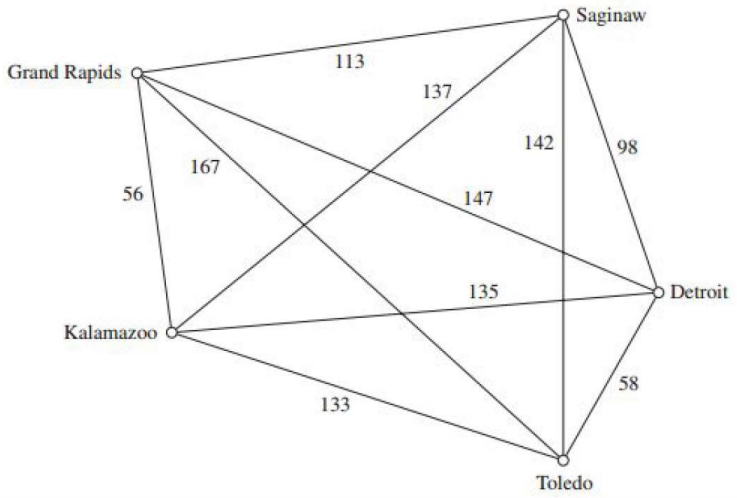
Algoritmos de Grafos No Dirigidos – Algoritmo del Vendedor Ambulante

- Este algoritmo busca pasar por todos los vértices del grafo mediante las aristas con el peso total mínimo, iniciando y terminando en el mismo.
- Igual que Hamilton se pasa una vez por vértice y se devuelve al origen.
- Pasos del algoritmo:
  - Encontrar todos los circuitos de Hamilton
  - Sumar los pesos asociados a las aristas de cada circuito encontrado, ese será el peso total del circuito.
  - Seleccionar aquel circuito con el peso total mínimo.

Algoritmos de Grafos No Dirigidos – Algoritmo del Vendedor Ambulante (cont.)

- Ejemplo:
  - Encontrar el circuito de peso total mínimo.

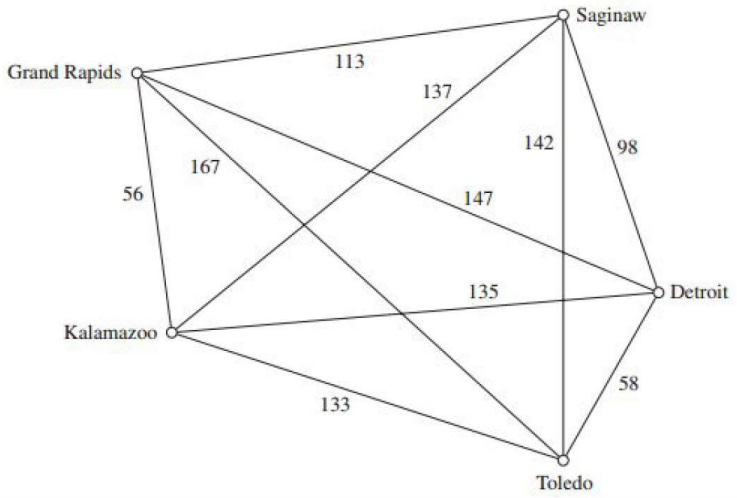
Route	*Total Distance (miles)*
Detroit–Toledo–Grand Rapids–Saginaw–Kalamazoo–Detroit	610
Detroit–Toledo–Grand Rapids–Kalamazoo–Saginaw–Detroit	516
Detroit–Toledo–Kalamazoo–Saginaw–Grand Rapids–Detroit	588
Detroit–Toledo–Kalamazoo–Grand Rapids–Saginaw–Detroit	458
Detroit–Toledo–Saginaw–Kalamazoo–Grand Rapids–Detroit	540
Detroit–Toledo–Saginaw–Grand Rapids–Kalamazoo–Detroit	504
Detroit–Saginaw–Toledo–Grand Rapids–Kalamazoo–Detroit	598
Detroit–Saginaw–Toledo–Kalamazoo–Grand Rapids–Detroit	576
Detroit–Saginaw–Kalamazoo–Toledo–Grand Rapids–Detroit	682
Detroit–Saginaw–Grand Rapids–Toledo–Kalamazoo–Detroit	646
Detroit–Grand Rapids–Saginaw–Toledo–Kalamazoo–Detroit	670
Detroit–Grand Rapids–Toledo–Saginaw–Kalamazoo–Detroit	728



Algoritmos de Grafos No Dirigidos – Algoritmo del Vendedor Ambulante (cont.)

- Ejemplo:
  - Encontrar el circuito de peso total mínimo.

Route	*Total Distance (miles)*
Detroit–Toledo–Grand Rapids–Saginaw–Kalamazoo–Detroit	610
Detroit–Toledo–Grand Rapids–Kalamazoo–Saginaw–Detroit	516
Detroit–Toledo–Kalamazoo–Saginaw–Grand Rapids–Detroit	588
Detroit–Toledo–Kalamazoo–Grand Rapids–Saginaw–Detroit	458
Detroit–Toledo–Saginaw–Kalamazoo–Grand Rapids–Detroit	540
Detroit–Toledo–Saginaw–Grand Rapids–Kalamazoo–Detroit	504
Detroit–Saginaw–Toledo–Grand Rapids–Kalamazoo–Detroit	598
Detroit–Saginaw–Toledo–Kalamazoo–Grand Rapids–Detroit	576
Detroit–Saginaw–Kalamazoo–Toledo–Grand Rapids–Detroit	682
Detroit–Saginaw–Grand Rapids–Toledo–Kalamazoo–Detroit	646
Detroit–Grand Rapids–Saginaw–Toledo–Kalamazoo–Detroit	670
Detroit–Grand Rapids–Toledo–Saginaw–Kalamazoo–Detroit	728



Algoritmos de Grafos No Dirigidos – Árboles Abarcadores de Costo Mínimo

- Sea $G = (V,E)$ un grafo conexo en donde cada arista (u,v) de E tiene un costo asociado $c(u,v)$.
- Un **árbol abarcador o generador** de un grafo conexo G es un árbol libre que contiene todos los vértices de G y es subgrafo de G, su peso es la suma de los costos de las aristas del árbol.
- Un **árbol abarcador de costo mínimo** o **generador mínimo** de un grafo conexo ponderado es un árbol abarcador o generador tal que la suma de los pesos de sus aristas es la más pequeña posible.

Algoritmos de Grafos No Dirigidos – Árboles Abarcadores de Costo Mínimo (cont.)

- Una aplicación típica de los árboles abarcadores de costo mínimo tiene lugar en el diseño de redes de comunicación.
  - Un árbol abarcador de costo mínimo representa una red que comunica todas las ciudades a un costo minimal.
- Hay diferentes maneras de construir un árbol abarcador de costo mínimo.
- Muchos métodos utilizan la propiedad *AAM*.
  - Sea $G = (V,E)$ un grafo conexo con una función de peso definida en las aristas.
  - Sea U algún subconjunto propio del conjunto de vértices V.
  - Si (u,v) es una arista de costo mínimo tal que $u \in U$ y $v \in V\text{-}U$, existe un árbol abarcador de costo mínimo que incluye (u,v) entre sus aristas.

Algoritmos de Grafos No Dirigidos – Algoritmo de Prim

- El **algoritmo de Prim** es un algoritmo de la teoría de los grafos para encontrar un árbol abarcador de costo mínimo en un grafo conexo, no dirigido y cuyas aristas están etiquetadas.
- En otras palabras, el algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el peso total de todas las aristas en el árbol es el mínimo posible.
- Si el grafo no es conexo, entonces el algoritmo encontrará el árbol abarcador de costo mínimo para uno de los componentes conexos que forman dicho grafo no conexo.

Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)

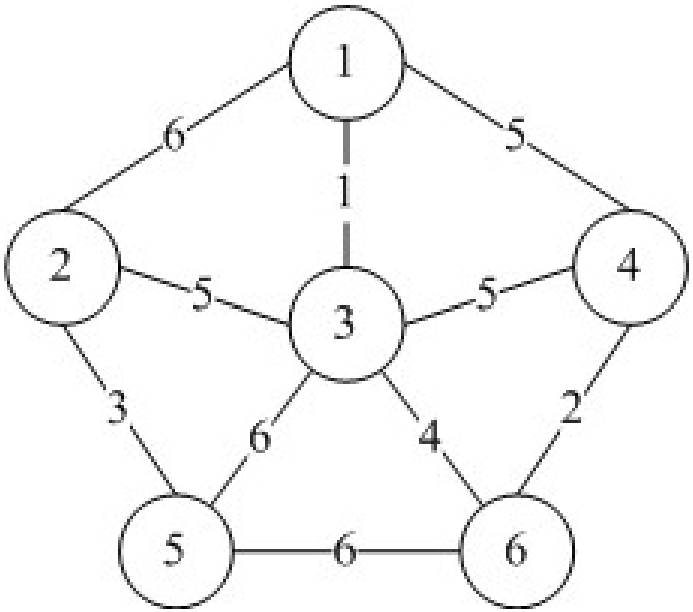
- El algoritmo fue diseñado en 1930 por el matemático Vojtech Jarnik y luego de manera independiente por el científico computacional Robert C. Prim en 1957 y redescubierto por Dijkstra en 1959.
- Por esta razón, el algoritmo es también conocido como **algoritmo DJP** o **algoritmo de Jarnik**.

Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)

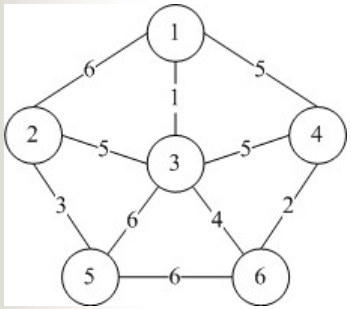
- El algoritmo comienza cuando se asigna a un conjunto U un valor inicial (un vértice del grafo), en el cual "crece" un árbol abarcador, arista por arista.
- En cada paso localiza la arista más corta (u,v) que conecta los vértices, y después agrega u en U. Este paso se repite hasta que $U = V$.
- Ejemplo en el Web:
  - http://www.dma.fi.upm.es/java/matematicadiscreta/Kruskal%5Fprim/applet.htm.
  - http://students.ceid.upatras.gr/~papagel/project/prim.htm.

Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)

- Ejemplo:
  - Encontrar el árbol abarcador de costo mínimo del siguiente grafo no dirigido utilizando el algoritmo de Prim.



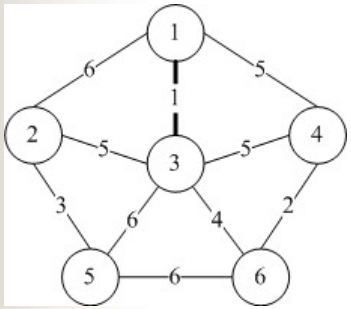
Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)



	1	2	3	4	5	6
1	–	6	1	5	–	–
2	6	–	5	–	3	–
3	1	5	–	5	6	4
4	5	–	5	–	–	2
5	–	3	6	–	–	6
6	–	–	4	2	6	–

T	*v*	*V*	*U*	*V-U*
∅	---	{1,2,3,4,5,6}	{1}	{2,3,4,5,6}

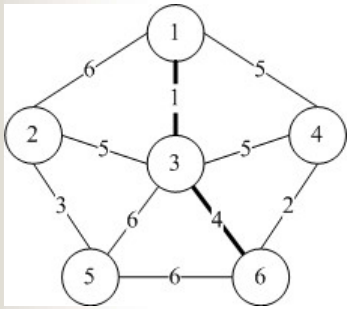
Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)



	1	2	3	4	5	6
1	–	6	1	5	–	–
2	6	–	5	–	3	–
3	1	5	–	5	6	4
4	5	–	5	–	–	2
5	–	3	6	–	–	6
6	–	–	4	2	6	–

T	*v*	*V*	*U*	*V-U*
∅	---	{1,2,3,4,5,6}	{1}	{2,3,4,5,6}
{(1,3)}	3	{1,2,3,4,5,6}	{1,3}	{2,4,5,6}

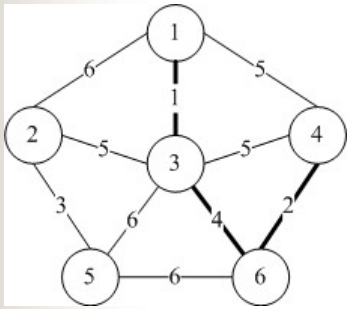
Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)



	1	2	3	4	5	6
1	–	6	1	5	–	–
2	6	–	5	–	3	–
3	1	5	–	5	6	4
4	5	–	5	–	–	2
5	–	3	6	–	–	6
6	–	–	4	2	6	–

T	*v*	*V*	*U*	*V-U*
∅	---	{1,2,3,4,5,6}	{1}	{2,3,4,5,6}
{(1,3)}	3	{1,2,3,4,5,6}	{1,3}	{2,4,5,6}
{(1,3),(3,6)}	6	{1,2,3,4,5,6}	{1,3,6}	{2,4,5}

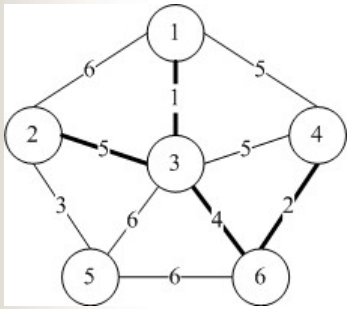
Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)



	1	2	3	4	5	6
1	–	6	1	5	–	–
2	6	–	5	–	3	–
3	1	5	–	5	6	4
4	5	–	5	–	–	2
5	–	3	6	–	–	6
6	–	–	4	2	6	–

T	*v*	*V*	*U*	*V-U*
∅	---	{1,2,3,4,5,6}	{1}	{2,3,4,5,6}
{(1,3)}	3	{1,2,3,4,5,6}	{1,3}	{2,4,5,6}
{(1,3),(3,6)}	6	{1,2,3,4,5,6}	{1,3,6}	{2,4,5}
{(1,3),(3,6),(6,4)}	4	{1,2,3,4,5,6}	{1,3,4,6}	{2,5}

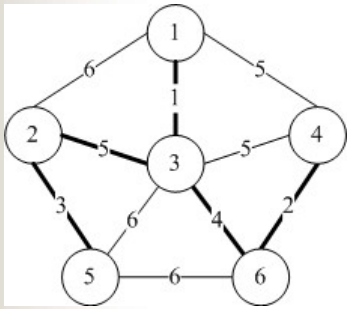
Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)

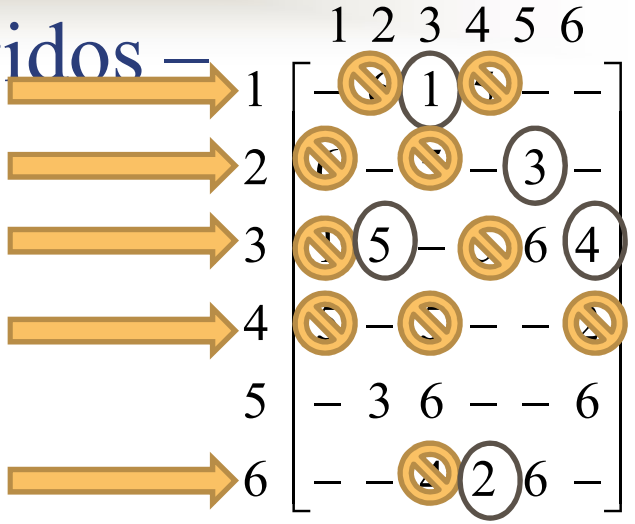


	1	2	3	4	5	6
1	–	6	1	5	–	–
2	6	–	5	–	3	–
3	1	5	–	5	6	4
4	5	–	5	–	–	2
5	–	3	6	–	–	6
6	–	–	4	2	6	–

T	*v*	*V*	*U*	*V-U*
∅	---	{1,2,3,4,5,6}	{1}	{2,3,4,5,6}
{(1,3)}	3	{1,2,3,4,5,6}	{1,3}	{2,4,5,6}
{(1,3),(3,6)}	6	{1,2,3,4,5,6}	{1,3,6}	{2,4,5}
{(1,3),(3,6),(6,4)}	4	{1,2,3,4,5,6}	{1,3,4,6}	{2,5}
{(1,3),(3,6),(6,4),(3,2)}	2	{1,2,3,4,5,6}	{1,2.3,4,6}	{5}

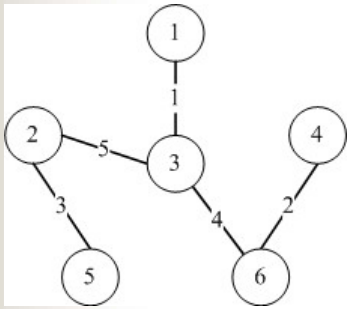
Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)

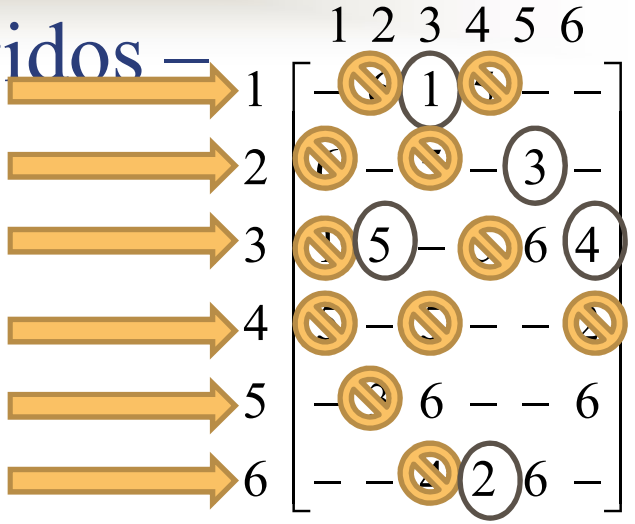




T	*v*	*V*	*U*	*V-U*
∅	---	{1,2,3,4,5,6}	{1}	{2,3,4,5,6}
{(1,3)}	3	{1,2,3,4,5,6}	{1,3}	{2,4,5,6}
{(1,3),(3,6)}	6	{1,2,3,4,5,6}	{1,3,6}	{2,4,5}
{(1,3),(3,6),(6,4)}	4	{1,2,3,4,5,6}	{1,3,4,6}	{2,5}
{(1,3),(3,6),(6,4),(3,2)}	2	{1,2,3,4,5,6}	{1,2.3,4,6}	{5}
{(1,3),(3,6),(6,4),(3,2),(2,5)}	5	{1,2,3,4,5,6}	{1,2,3,4,5,6}	{}

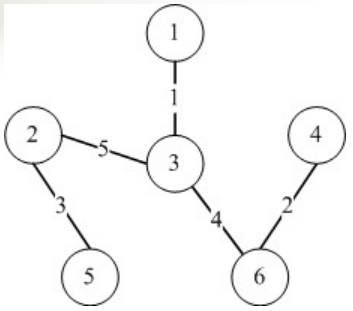
Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)





T	*v*	*V*	*U*	*V-U*
∅	---	{1,2,3,4,5,6}	{1}	{2,3,4,5,6}
{(1,3)}	3	{1,2,3,4,5,6}	{1,3}	{2,4,5,6}
{(1,3),(3,6)}	6	{1,2,3,4,5,6}	{1,3,6}	{2,4,5}
{(1,3),(3,6),(6,4)}	4	{1,2,3,4,5,6}	{1,3,4,6}	{2,5}
{(1,3),(3,6),(6,4),(3,2)}	2	{1,2,3,4,5,6}	{1,2.3,4,6}	{5}
{(1,3),(3,6),(6,4),(3,2),(2,5)}	5	{1,2,3,4,5,6}	{1,2,3,4,5,6}	{}

Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)



$U = \{1\}$

$V - U = \{2,3,4,5,6\}$

	1	2	3	4	5	6
$M[1]$	*					
$D[1]$	0	6	1	5	–	–
$P[1]$	0	1	1	1	1	1

$U = \{1,3\}$

$V - U = \{2,4,5,6\}$

	1	2	3	4	5	6
$M[2]$	*		*			
$D[2]$	0	5	1	5	6	4
$P[2]$	0	3	1	1	3	3

$U = \{1,3,6\}$

$V - U = \{2,4,5\}$

	1	2	3	4	5	6
$M[3]$	*		*			*
$D[3]$	0	5	1	2	6	4
$P[3]$	0	3	1	6	3	3

$U = \{1,3,4,6\}$

$V - U = \{2,5\}$

	1	2	3	4	5	6
$M[4]$	*		*	*		*
$D[4]$	0	5	1	2	6	4
$P[4]$	0	3	1	6	3	3

$U = \{1,2,3,4,6\}$

$V - U = \{5\}$

	1	2	3	4	5	6
$M[5]$	*	*	*	*		*
$D[5]$	0	5	1	2	3	4
$P[5]$	0	3	1	6	2	3

$U = \{1,2,3,4,5,6\}$

$V - U = \{\}$

	1	2	3	4	5	6
$M[6]$	*	*	*	*	*	*
$D[6]$	0	5	1	2	3	4
$P[6]$	0	3	1	6	2	3

Algoritmos de Grafos No Dirigidos – Algoritmo de Prim (cont.)

- Pseudocódigo del algoritmo:

```
PRIM (Grafo G, nodo_fuente s)
   // inicializamos todos los nodos del grafo. La distancia la ponemos a infinito
   // y el padre de cada nodo a NULL
    for each u ∈ V[G] do
        distancia[u] = INFINITO
        padre[u] = NULL
    distancia[s]=0
    //encolamos todos los nodos del grafo
    Encolar(cola, V[G])
    while cola != 0 do
        // OJO: Se extrae el nodo que tiene distancia mínima y se conserva la condición
        // de Cola de prioridad
        u = extraer_minimo(cola)
        for v ∈ adyacencia[u] do
            if ((v ∈ cola) && (distancia[v] > peso(u, v)) do
                padre[v] = u
                distancia[v] = peso(u, v)
```

Algoritmos de Grafos No Dirigidos – Algoritmo de Kruskal (cont.)

- El **algoritmo de Kruskal** es un algoritmo de la teoría de grafos para encontrar un árbol abarcador de costo mínimo en un grafo conexo y ponderado.
- Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor total de todas las aristas del árbol es el mínimo.
- Si el grafo no es conexo, entonces busca un bosque abarcador de costo mínimo (un *árbol abarcador de costo mínimo* para cada componente conexa).
- Este algoritmo fue escrito por Joseph Kruskal.

Algoritmos de Grafos No Dirigidos – Algoritmo de Kruskal (cont.)

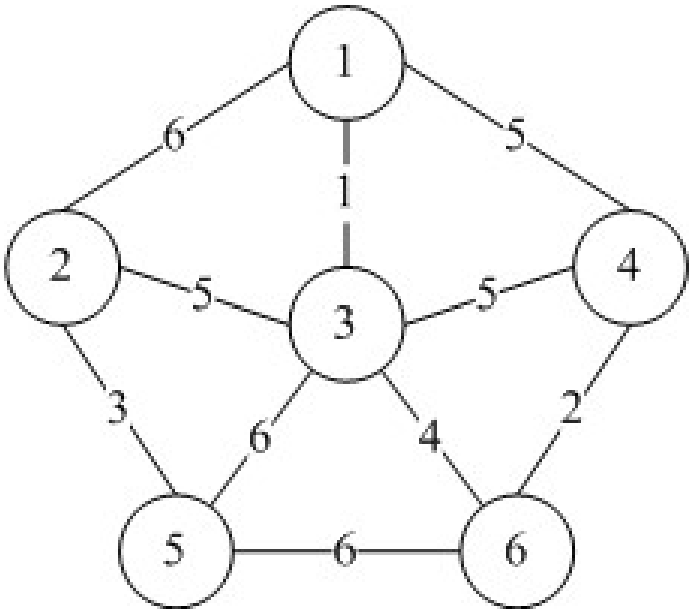
- Funciona de la siguiente manera:
  - Se crea un bosque *B* (un conjunto de árboles), donde cada vértice del grafo es un árbol separado.
  - Se crea un conjunto *C* que contenga a todas las aristas del grafo.
  - Mientras *C* es sea vacío:
    - Eliminar una arista de peso mínimo de *C*.
    - Si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol.
    - En caso contrario, se desecha la arista.

Algoritmos de Grafos No Dirigidos – Algoritmo de Kruskal (cont.)

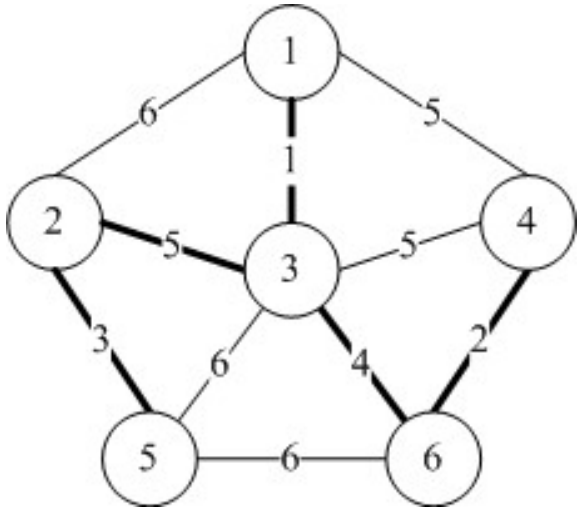
- Al acabar el algoritmo, el bosque tiene una sola componente, la cual forma un árbol abarcador de costo mínimo del grafo.
- Ejemplo en el Web:
  - http://students.ceid.upatras.gr/~papagel/project/kruskal.htm.

Algoritmos de Grafos No Dirigidos – Algoritmo de Kruskal (cont.)

- Ejemplo:
  - Encontrar el árbol abarcador de costo mínimo del siguiente grafo no dirigido utilizando el algoritmo de Kruskal.

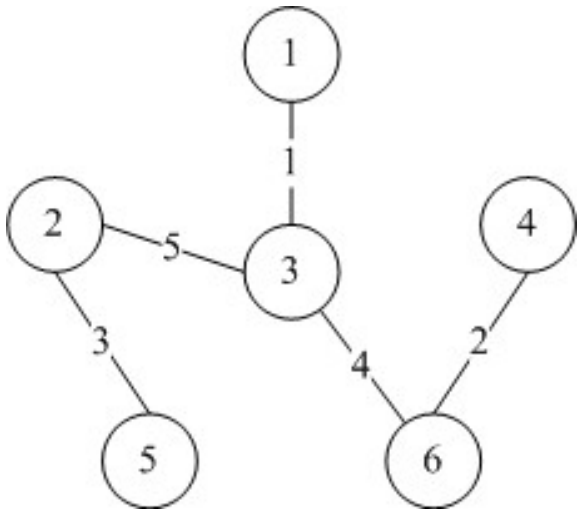


Algoritmos de Grafos No Dirigidos – Algoritmo de Kruskal (cont.)



Costo	Aristas	
1	(1,3)	⟸
2	(4,6)	⟸
3	(2,5)	⟸
4	(3,6)	⟸
5	( ) – (2,3) – (3,4)	⟸
6	(3,5) – (5,6)	

Algoritmos de Grafos No Dirigidos – Algoritmo de Kruskal (cont.)



Costo	Aristas
1	(1,3)
2	(4,6)
3	(2,5)
4	(3,6)
5	(1,2) – (2,3) – (1,5)
6	(2,5) – (3,6)

Algoritmos de Grafos No Dirigidos – Algoritmo de Kruskal (cont.)

- Pseudocódigo del algoritmo:

```
KRUSKAL (Grafo G)
      T = ∅
      for each v ∈ V do
            CreateSet(v)    // Crea un componente conexo que contiene solo a v
            Q = ordenar(A) // Sea Q una ordenación de las aristas de G con relación a su peso
            i = 1
            while |T| < |V|-1 do    // Parar cuando en T hayan n-1 aristas,
                                    // pues un árbol libre con n vértices
                                    // tiene n-1 aristas por la propiedad 1
                  (u,v) = Q[i]
                  Cu = SetOf(u)
                  Cv = SetOf(v)
                  // Vértices en componentes diferentes no forman ciclo
                  if Cu != Cv do
                        T = T ∪ {(u,v)}
                        Merge(Cu,Cv)
                  i = i + 1
```

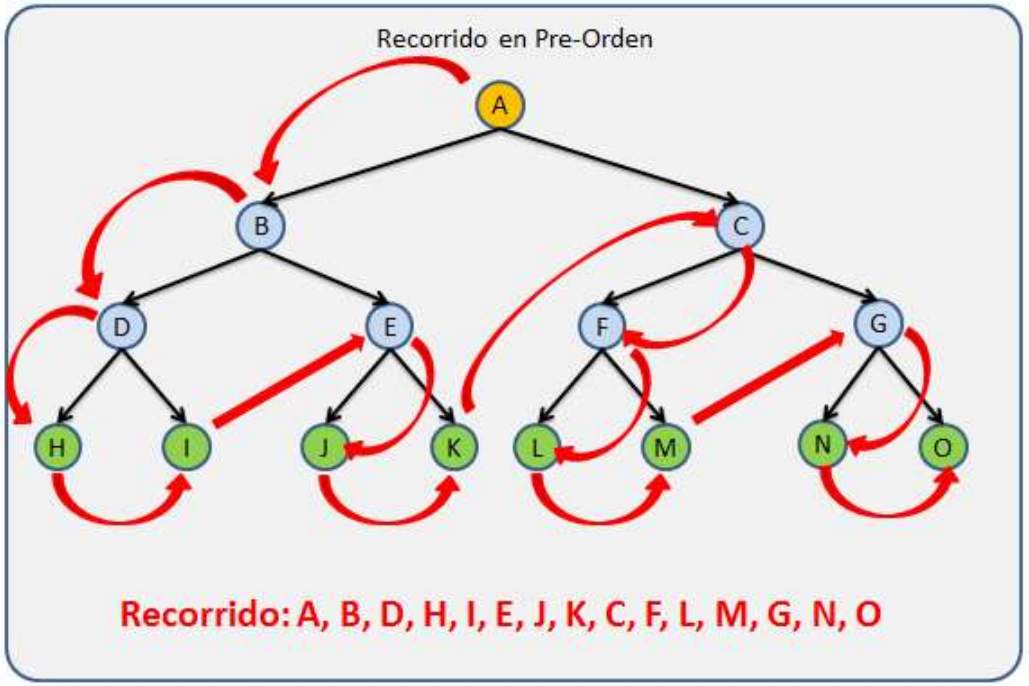
Algoritmos de Grafos No Dirigidos – Algoritmo de Recorrido en Pre-Orden

- Algoritmo que se usa para recorrer sistemáticamente cada vértice de un árbol ordenado en pre-orden.
- El recorrido inicia en la raíz y luego se recorre en pre-orden cada unos de los sub-árboles de izquierda a derecha.
- Considerando un árbol *T* con una raíz *r*.
  - Si *T* únicamente consta de su raíz, entonces *r* es el recorrido en pre-orden de *T*.
  - De otra forma, supongamos que *T*1, *T*2, ..., *Tn* son los sub-árboles de *r* de izquierda a derecha. El recorrido en pre-orden comienza en *r*, continua en *T*1, luego en *T*2 y así sucesivamente hasta *Tn*.

Algoritmos de Grafos No Dirigidos – Algoritmo de Recorrido en Pre-Orden

```
public void preorden(NodoArbol nodo) {
    if (nodo == null)
        return;

    System.out.print("Nodo Value => " + nodo.value);
    preorden(nodo.hijo1);
    preorden(nodo.hijo2);
}
```



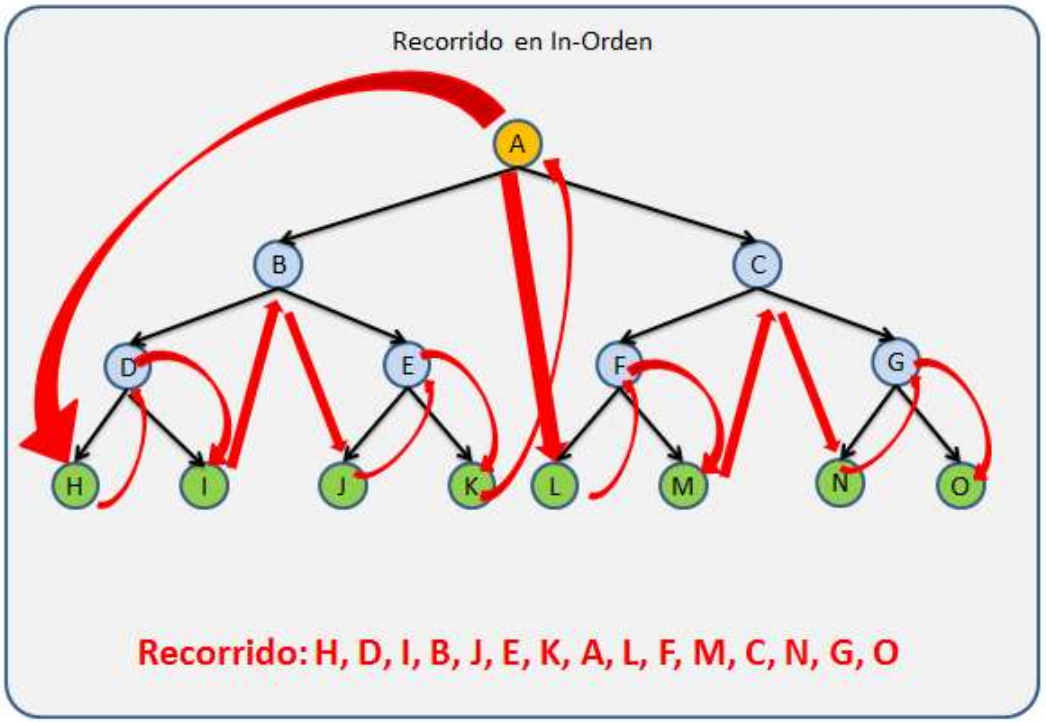
Algoritmos de Grafos No Dirigidos – Algoritmo de Recorrido en Orden

- Algoritmo que se usa para recorrer sistemáticamente cada vértice de un árbol ordenado en orden.
- El recorrido inicia en el primer sub-árbol, luego se recorre la raíz y al final se recorre en orden los demás sub-árboles.
- Considerando un árbol *T* con una raíz *r*.
  - Se recorren los sub-árboles de la izquierda, se continúa con la raíz *r*, y se completa con los sub-árboles de la derecha.

Algoritmos de Grafos No Dirigidos – Algoritmo de Recorrido en Orden

```
public void inorden(NodoArbol nodo) {
    if (nodo == null)
        return;

    inorden(nodo.hijo1);
    System.out.print("Nodo Value => " + nodo.value);
    inorden(nodo.hijo2);
}
```



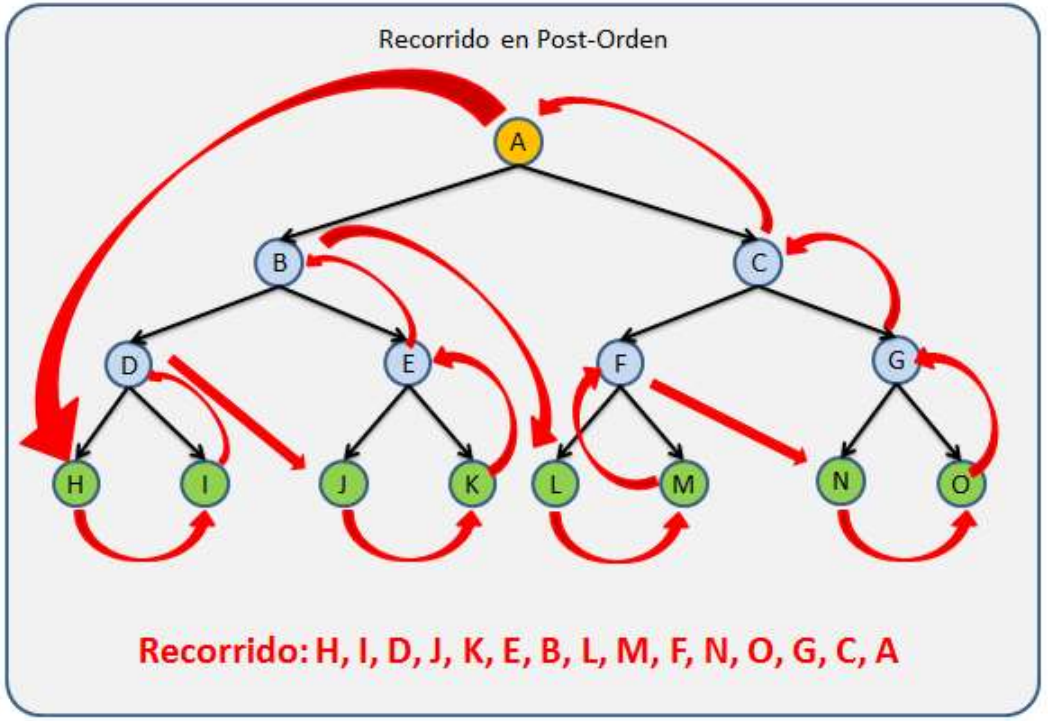
Algoritmos de Grafos No Dirigidos – Algoritmo de Recorrido en Post-Orden

- Algoritmo que se usa para recorrer sistemáticamente cada vértice de un árbol ordenado en post-orden.
- El recorrido inicia en cada uno de los sub-árboles y al final se recorre la raíz.
- Considerando un árbol *T* con una raíz *r*.
  - Si *T* únicamente consta de su raíz, entonces *r* es el recorrido en post-orden de *T*.
  - Se recorren los sub-árboles de la izquierda, se completa con los sub-árboles de la derecha y por último se recorre la raíz *r*.

Algoritmos de Grafos No Dirigidos – Algoritmo de Recorrido en Post-Orden

```
public void posorden(NodoArbol nodo) {
    if (nodo == null)
        return;

    posorden(nodo.hijo1);
    posorden(nodo.hijo2);
    System.out.print("Nodo Value => " + nodo.value);
}
```



Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Anchura o Amplitud

- **Búsqueda en anchura o amplitud** (*BFS* o *Breadth-first search* en inglés) es un algoritmo para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles).
  - Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo.
  - A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.
- Su nombre se debe a que expande uniformemente la frontera entre lo descubierto y lo no descubierto. Llega a los nodos de distancia *k*, sólo tras haber llegado a todos los nodos a distancia *k*-1.

Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

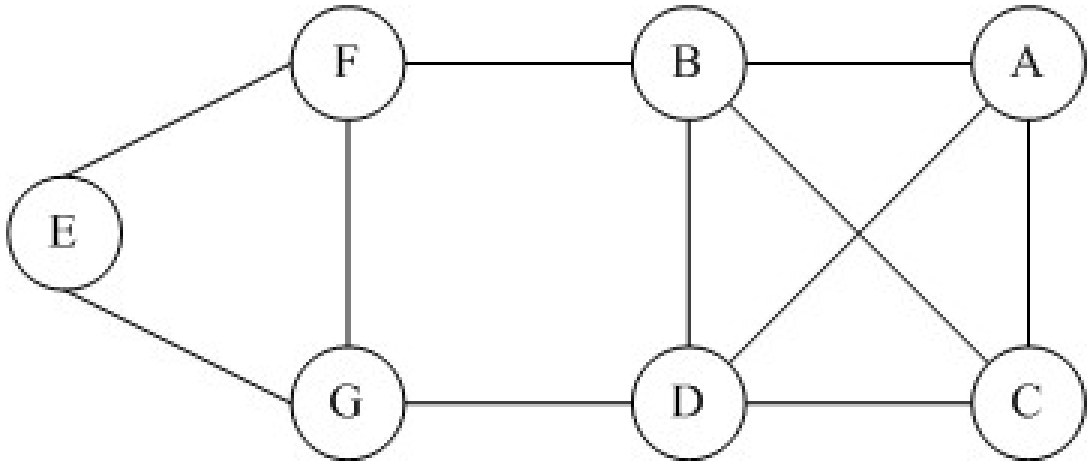
- Formalmente, *BFS* es un algoritmo de **búsqueda sin información**, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución.
- El algoritmo no usa ninguna estrategia heurística.
- El peso de las aristas para ejecutar **BFS** debe de ser de IGUAL costo.
- Si las aristas tienen pesos negativos se aplica el algoritmo de Bellman-Ford en alguna de sus dos versiones.

Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

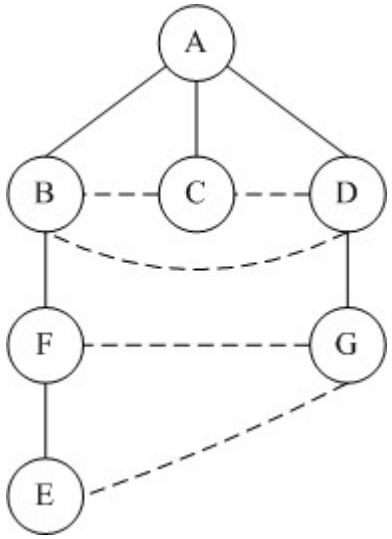
- Durante un recorrido en anchura, cuando se recorren ciertas aristas, llevan a vértices sin visitar.
- Las aristas que llevan a vértices nuevos se conocen como **aristas de árbol** y forman un **árbol abarcador en anchura** para el grafo no dirigido dado.
  - **Arista de árbol.** Es la arista que forma parte del árbol.
- Además, existen un tipo de arista definido por una búsqueda en profundidad de un grafo no dirigido, que se conocen como:
  - **Aristas cruzadas:** Es la arista que existe entre un vértice *v* a un vértice *w* que no es ancestro ni descendiente.
- Todos las aristas forman un **bosque abarcador en profundidad** para el grafo dirigido dado.

Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

- Ejemplo:
  - Realizar el recorrido en anchura (siga el orden alfabético) y encontrar el bosque abarcador del siguiente grafo no dirigido.



Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)



Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Anchura (cont.)

- Pseudocódigo del algoritmo:

```
BFS(grafo G, nodo_fuente s)
{
   // recorremos todos los vértices del grafo inicializándolos a NO_VISITADO,
   // distancia INFINITA y padre de cada nodo NULL
   for u ∈ V[G] do
   {
      estado[u] = NO_VISITADO;
      distancia[u] = INFINITO; /* distancia infinita si el nodo no es alcanzable */
      padre[u] = NULL;
   }
   estado[s] = VISITADO;
   distancia[s] = 0;
   Encolar(Q, s);
   while Q != 0 do
   {
      // extraemos el nodo u de la cola Q y exploramos todos sus nodos adyacentes
      u = extraer(Q);
      for  v ∈ adyacencia[u]  do
      {
         if estado[v] == NO_VISITADO then
         {
               estado[v] = VISITADO;
               distancia[v] = distancia[u] + 1;
               padre[v] = u;
               Encolar(Q, v);
         }
      }
   }
}
```

Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Profundidad

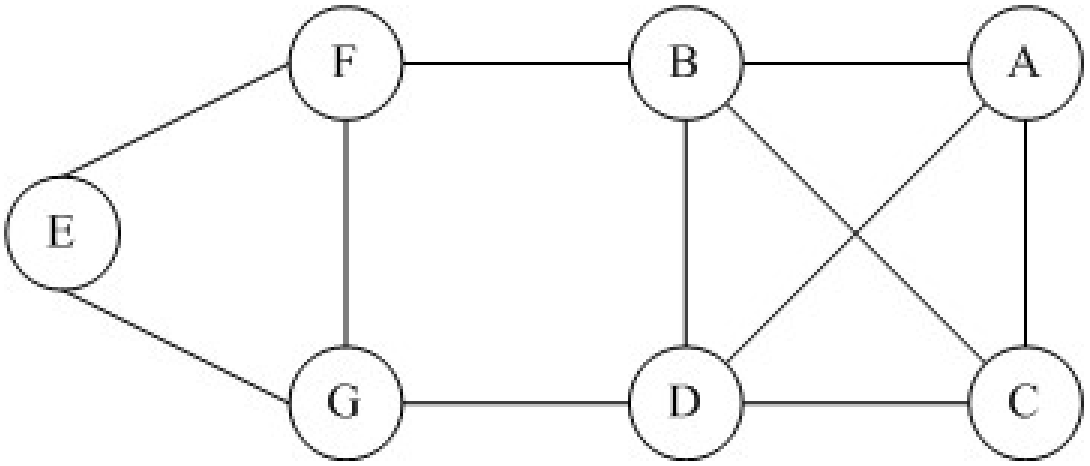
- Un **recorrido en profundidad** (en inglés **DFS** - *Depth First Search*) es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme.
- Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto.
- Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.
- Durante un recorrido en profundidad, cuando se recorren ciertas aristas, llevan a vértices sin visitar.

Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

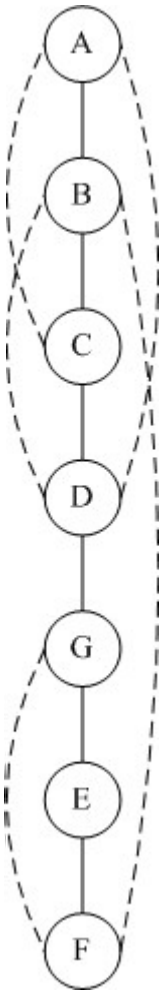
- Las aristas que llevan a vértices nuevos se conocen como **aristas de árbol** y forman un **árbol abarcador en profundidad** para el grafo no dirigido dado.
  - **Arista de árbol.** Es la arista que forma parte del árbol.
- Además, existen un tipo de arista definido por una búsqueda en profundidad de un grafo no dirigido, que se conocen como:
  - **Aristas de retroceso:** Es la arista que va de un vértice *v* a un vértice *w* que es antecesor de *v* en el árbol abarcador. Una arista que va de un vértice hacia si mismo se considera una arista de retroceso.
- Todos las aristas forman un **bosque abarcador en profundidad** para el grafo dirigido dado.

Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

- Ejemplo:
  - Realizar el recorrido en profundidad (siga el orden alfabético) y encontrar el bosque abarcador del siguiente grafo no dirigido.



Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)



Algoritmos de Grafos No Dirigidos – Algoritmo de Búsqueda en Profundidad (cont.)

- Pseudocódigo del algoritmo:

```
DFS(grafo G)
for each vertice u ∈ V[G] do
        estado[u]=NO_VISITADO
        padre[u]= NULL
tiempo=0
for each vertice u ∈ V[G] do
        if estado[u] = NO_VISITADO then
                DFS-Visitar(u)
```

```
DFS-Visitar(nodo u)
estado[u]=VISITADO
tiempo=tiempo+1
d[u]=tiempo
for each v ∈ Vecinos[u] do
        if estado[v]=NO_VISITADO then
                padre[v]=u
                DFS-Visitar(v)
estado[u]=TERMINADO
tiempo=tiempo+1
f[u]=tiempo
```

Referencias Bibliográficas

- Aho, Hopcroft & Ullman. “Estructuras de Datos y Algoritmos”. Pearson – Addison Wesley Longman, Primera Edición, 1998.
- Wikipedia. URL: http://es.wikipedia.org.