ALGORITMOS Y ESTRUCTURAS DE DATOS
CURSADA 2025

LICENCIATURA EN SISTEMAS - UNRN

Prof. Sebastián N. Valle

snvalle@unrn.edu.ar

Prof. Guillermo A. Difabio

guillermodifabio@unrn.edu.ar

GRAFOS

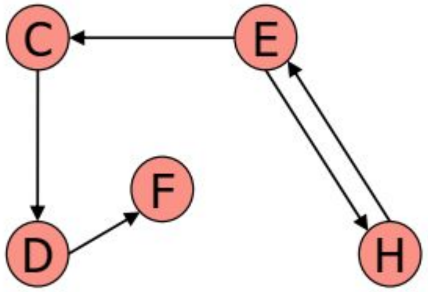
GRAFOS

- Definición
- Terminología
- Representaciones
- Tiempos de Ejecución
- Recorridos

DEFINICIÓN

- Un grafo es un modelo para representar relaciones entre elementos de un conjunto.
- Se lo puede representar como un par ordenado G=(V,E) donde V es un conjunto de vértices o nodos, con una relación entre ellos; E es un conjunto de pares (u,v), u,v ∈ V, llamados aristas o arcos.
- Existen 2 tipos de grafos:
  - Dirigidos: la relación sobre V no es simétrica. Arista es equivalente a par ordenado (u,v)
  - No Dirigidos: la relación sobre V es simétrica. Arista es equivalente a par no ordenado {u,v}, u,v ∈ V y u ≠ v.

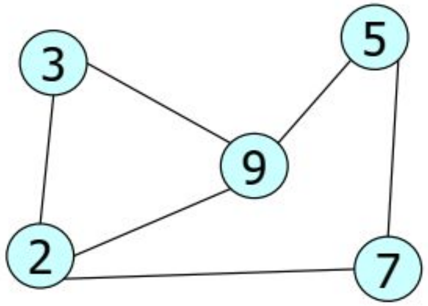
DEFINICIÓN - EJEMPLOS



Grafo dirigido G(V,E).

V = {C,D,E,F,H}

E={(C,D),(D,F),(E,C),(E,H), (H,E)}



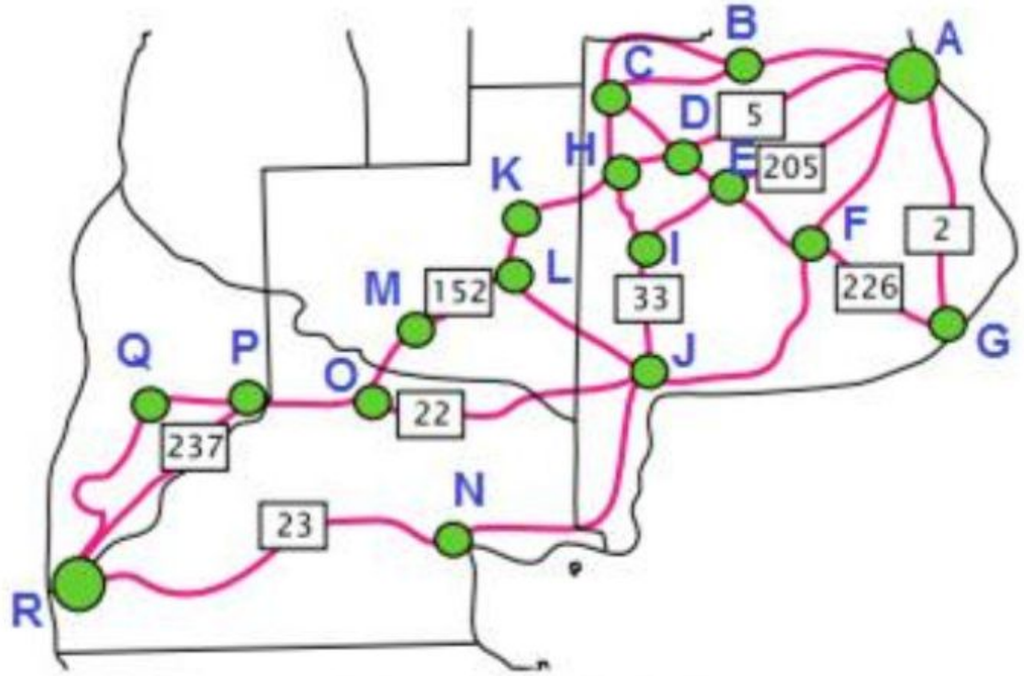
Grafo no dirigido G(V,E).

V = {2,3,5,7,9}

E= {{2,3},{2,7},{2,9},{3,9}, {5,7},{5,9}}

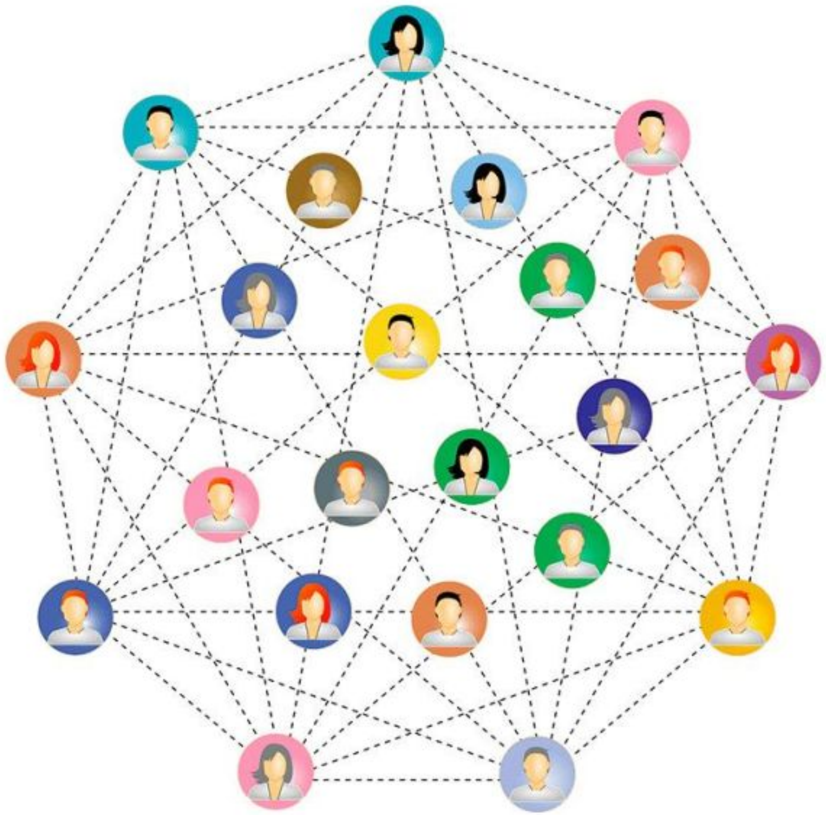
EJEMPLOS

Ciudades **conectadas por**
Rutas

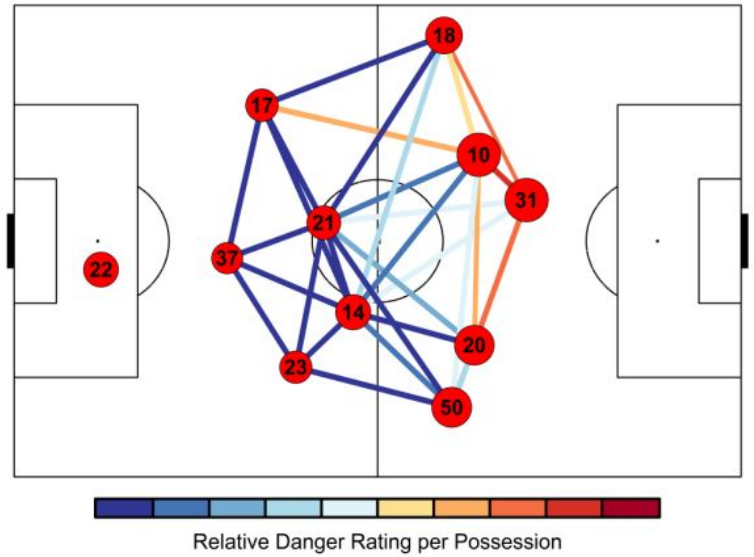


EJEMPLOS

Personas **conectadas en una red social**

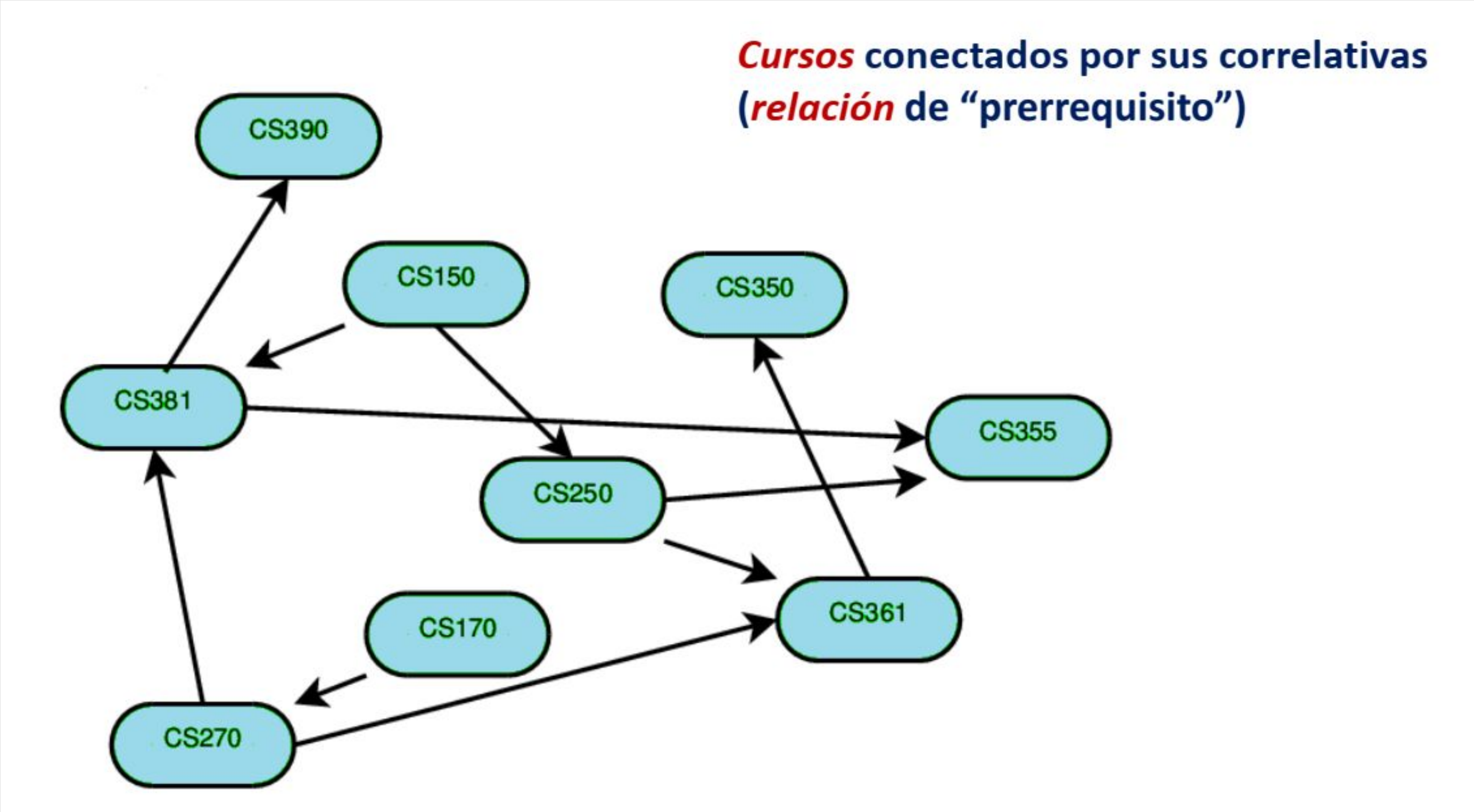


EJEMPLOS



Red de pases para el Barcelona y el AC Milan de un partido de Liga de Campeones.
Las flechas más oscuras y gruesas indican más pases entre cada jugador.

EJEMPLOS


Cursos conectados por sus correlativas
(*relación* de "prerrequisito")
CS390
CS150
CS350
CS381
CS355
CS250
CS361
CS170
CS270

TERMINOLOGÍA

- **v es adyacente a u** si existe una arista (u,v) ∈ E.
  - en un grafo no dirigido, (u,v) ∈ E incide en los nodos u, v.
  - en un grafo dirigido, (u,v) ∈ E incide en v, y parte de u.
- En grafos no dirigidos:
  - **El grado de un nodo**: número de arcos que inciden en él.
- En grafos dirigidos:
  - existen el **grado de salida (grado_out)** y el **grado de entrada (grado_in)**.
    - el grado_out es el número de arcos que parten de él y
    - el grado_in es el número de arcos que inciden en él.
  - El grado del vértice será la suma de los grados de entrada y de salida.
- **Grado de un grafo**: máximo grado de sus vértices.

TERMINOLOGÍA (2)

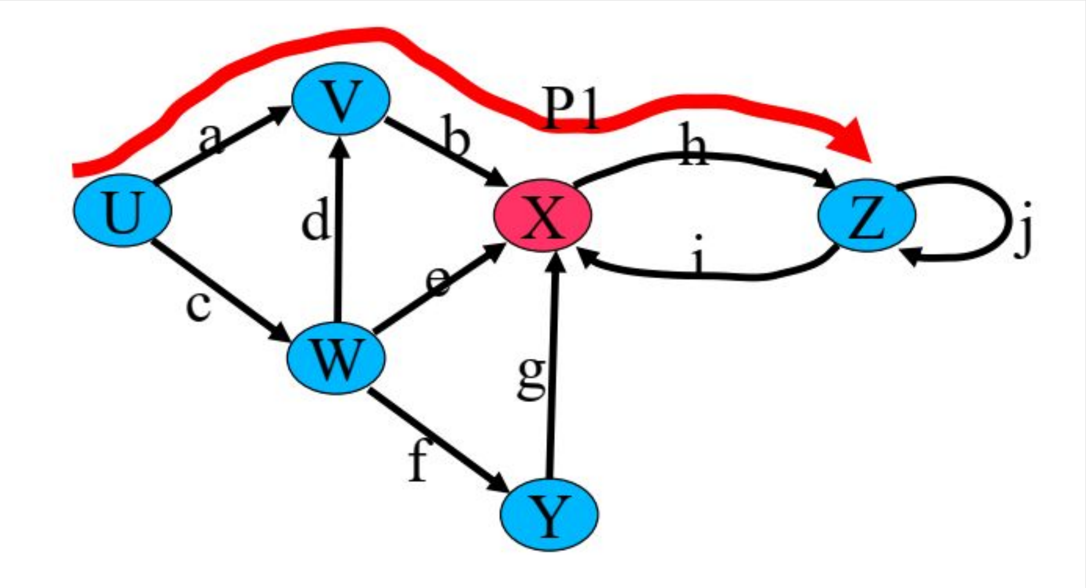
- **Camino** desde u ∈ V a v ∈ V : secuencia $v_1, v_2, \ldots, v_k$ tal que $u=v_1$, $v=v_k$, y $(v_{i-1},v_i) \in E$, para $i = 2, \ldots, k$.
- Ejemplo: camino desde **a** a **d** → <a,b,e,c,d>

- **Longitud de un camino:** número de arcos del camino.
  - Ejemplos: long. del camino desde **a** a **d** → <a,b,e,c,d> es 4.

    long. del camino desde **a** a **d** → <a,b,e,f,b,e,c,d> es 7.

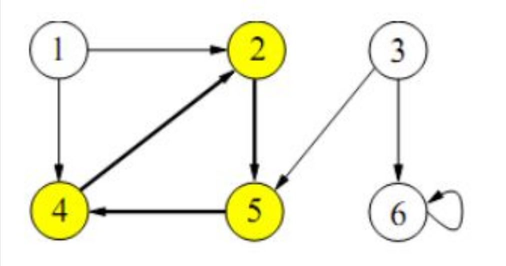
TERMINOLOGÍA (3)

- **Camino simple:** camino en el que todos sus vértices, excepto, tal vez, el primero y el último, son distintos. P1 es un camino simple desde U a Z.

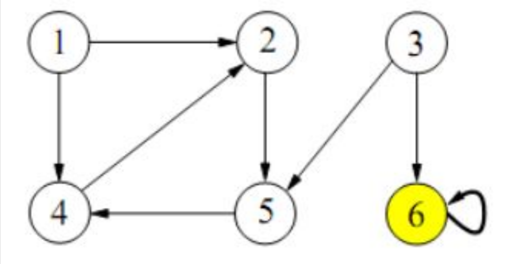


TERMINOLOGÍA (4)

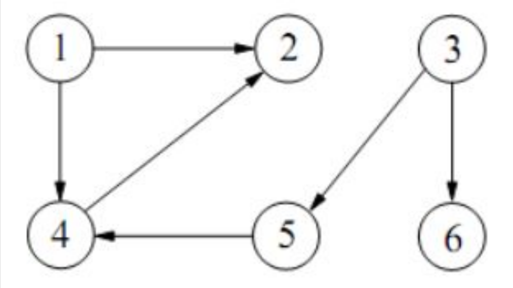
- **Ciclo:** camino desde $v_1, v_2, \ldots, v_k$ tal que $v_1 = v_k$ Ejemplo: <2,5,4,2> es un ciclo de longitud 3. El ciclo es simple si el camino es simple.



- **Bucle:** ciclo de longitud 1.

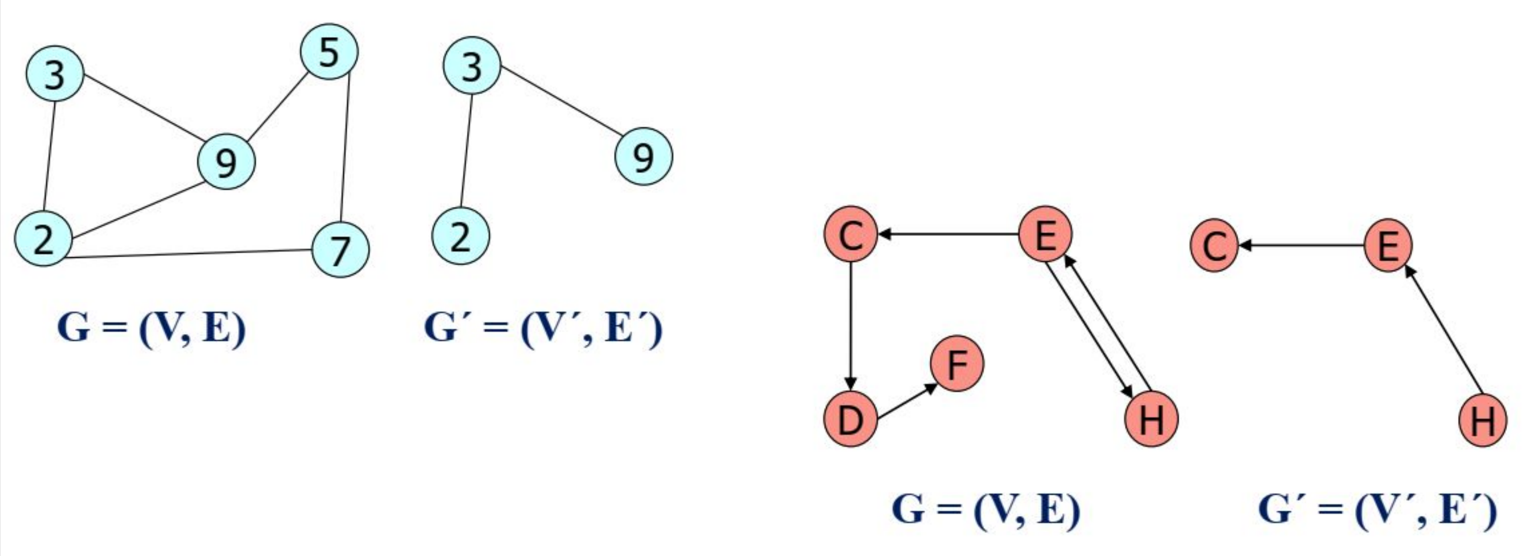


- **Grafo acíclico:** grafo sin ciclos.



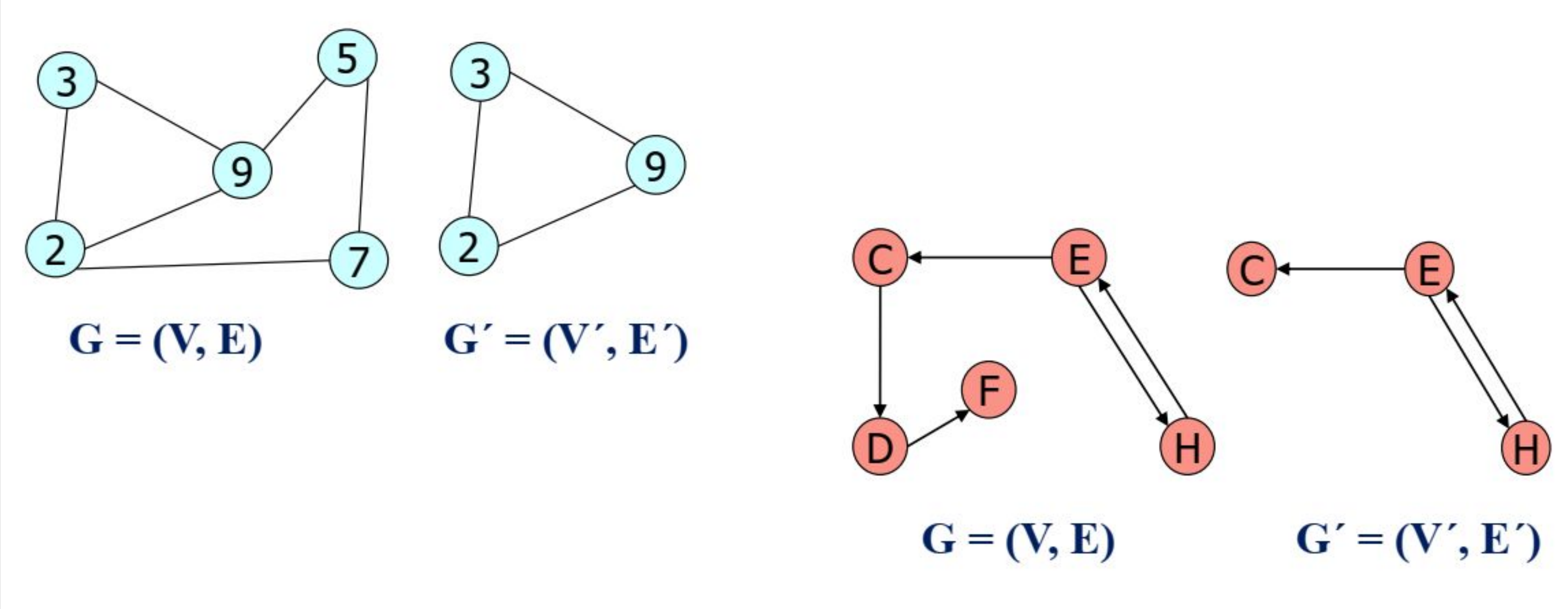
TERMINOLOGÍA (5)

- Dado un grafo G=(V, E), se dice que G´=(V´, E´) es un **subgrafo** de G, si V'esta incluido en V y E' está incluido en E.



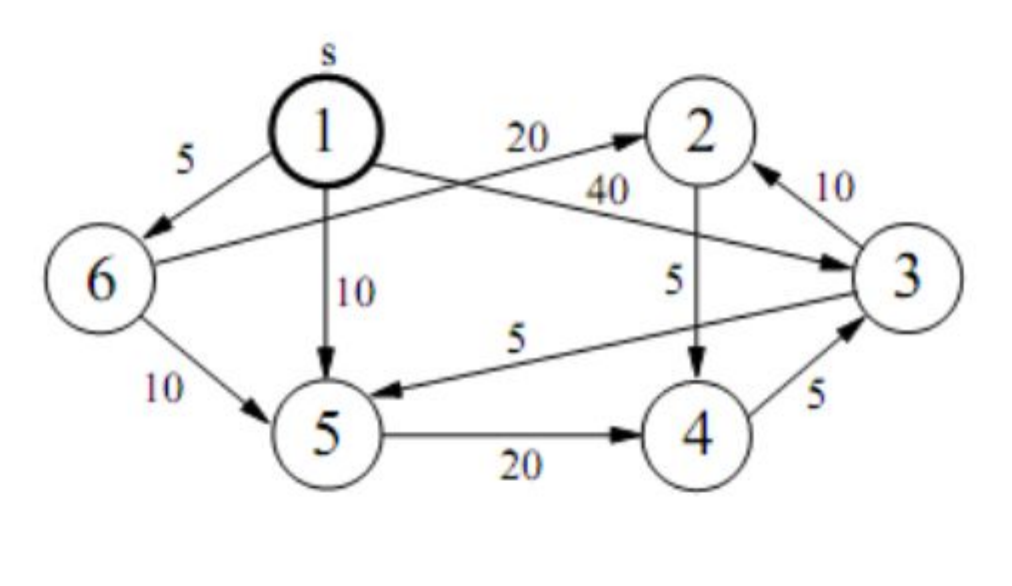
TERMINOLOGÍA (6)

- Un subgrafo inducido por V' está incluido en V : G' = (V',E') tal que E' = {(u,v) pertenecientes a E | u,v pertenecen a V'}.



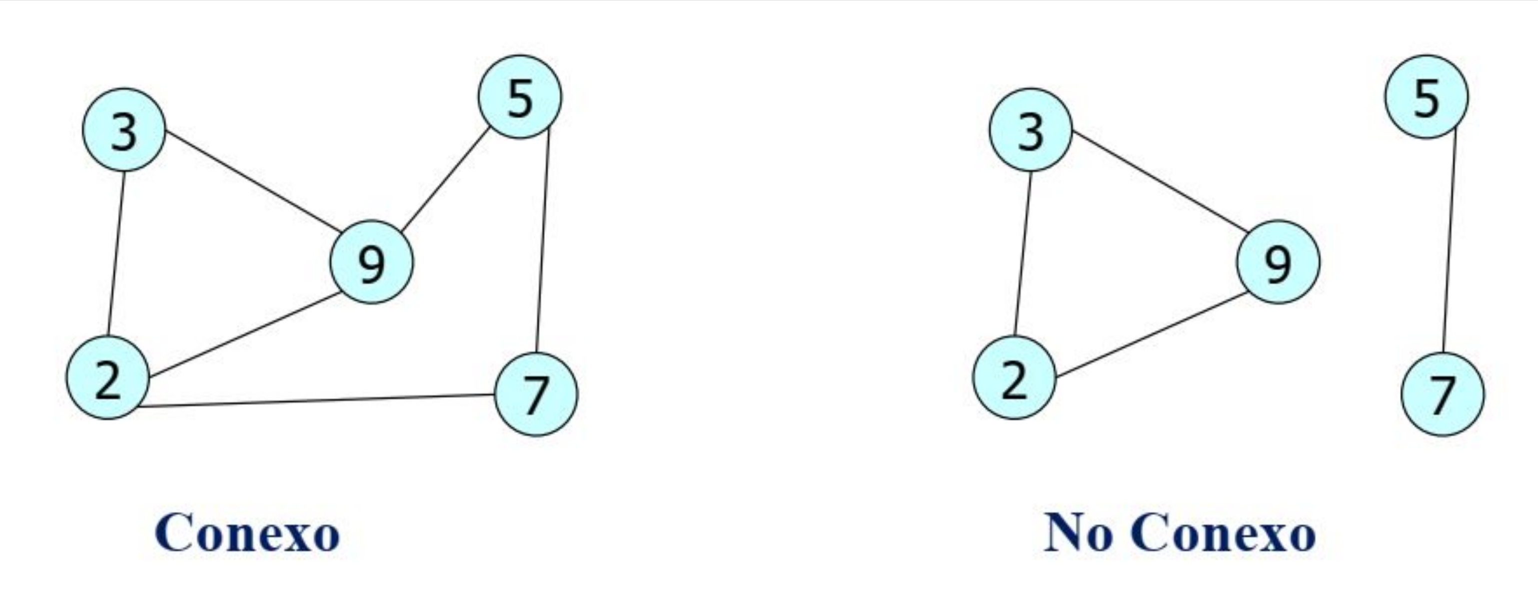
TERMINOLOGÍA (7)

- Un grafo **ponderado, pesado o con costos** es un grafo donde cada arco o arista tiene asociado un valor o etiqueta.



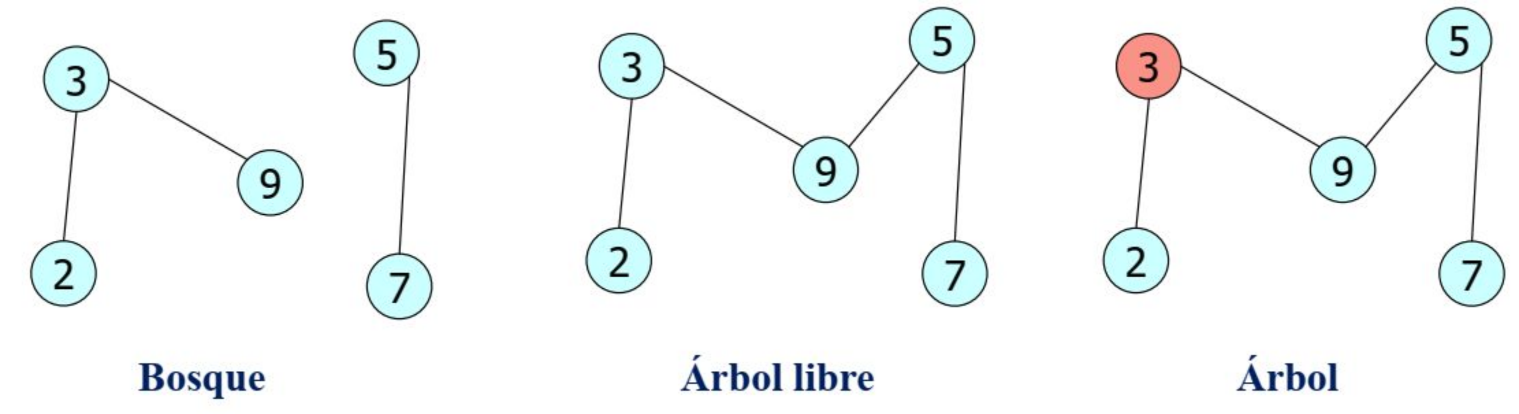
TERMINOLOGÍA (8)

- **Conectividad en grafos no dirigidos**
  - Un grafo no dirigido es conexo si hay un camino entre cada par de vértices.



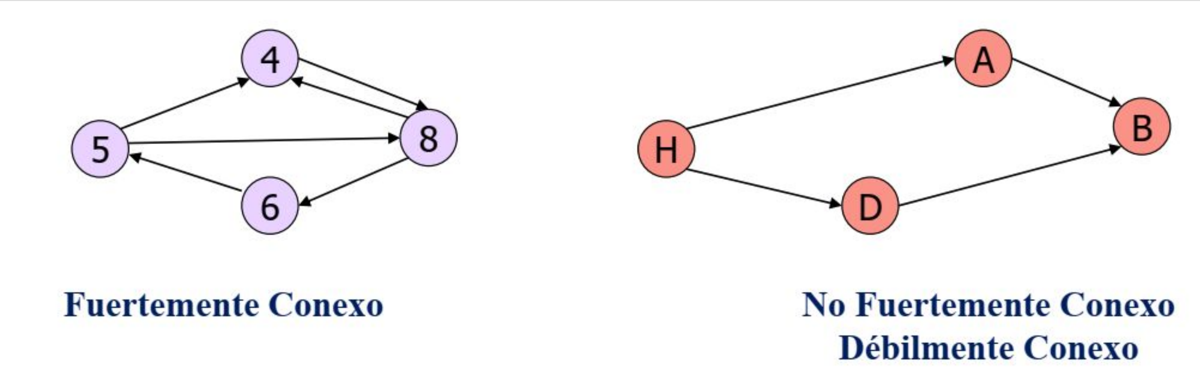
TERMINOLOGÍA (9)

- Conectividad: bosque y árbol
  - Un **bosque** es un grafo sin ciclos.
  - Un **árbol libre** es un bosque conexo.
  - Un **árbol** es un árbol libre en el que un nodo se ha designado como raíz.



TERMINOLOGÍA (10)

- **Conectividad en grafos dirigidos**
- v **es alcanzable desde** u, si existe un camino de u a v.
- Un grafo dirigido se denomina **fuertemente conexo** si existe un camino desde cualquier vértice a cualquier otro vértice.
- Si un grafo dirigido no es fuertemente conexo, pero el grafo subyacente (sin sentido en los arcos) es conexo, el grafo es **débilmente conexo**.



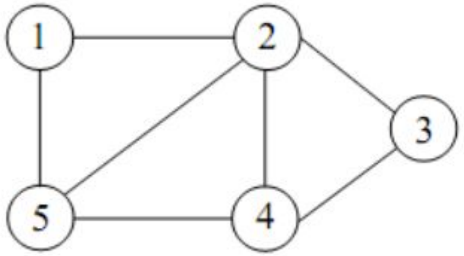
REPRESENTACIONES

- Matriz de Adyacencias
- Lista de Adyacencias

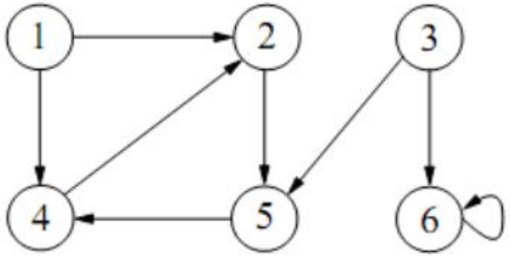
MATRIZ DE ADYACENCIAS

- G =(V,E): matriz A de dimensión |V| × |V|.
- Valor a_{ij} de la matriz:

$$a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{en cualquier otro caso} \end{cases}$$



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



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

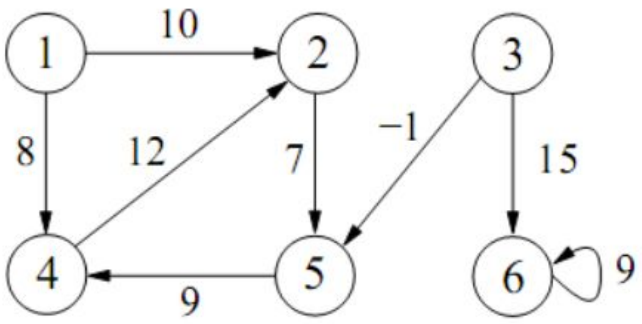
MATRIZ DE ADYACENCIAS (2)

- **Ventajas**
  - Representación útil para grafos con número de vértices pequeño, o grafos densos ($|E| \approx |V| \times |V|$)
  - Comprobar si una arista (u,v) pertenece a E → consultar posición A(u,v)
  - Costo de tiempo $T(|V|,|E|) = O(1)$
- **Desventajas**
  - Costo espacial: $O(|V|^2)$

MATRIZ DE ADYACENCIAS (3)

- Representación aplicada a Grafos pesados
- El **peso de (i,j)** se almacena en **A(i, j)**

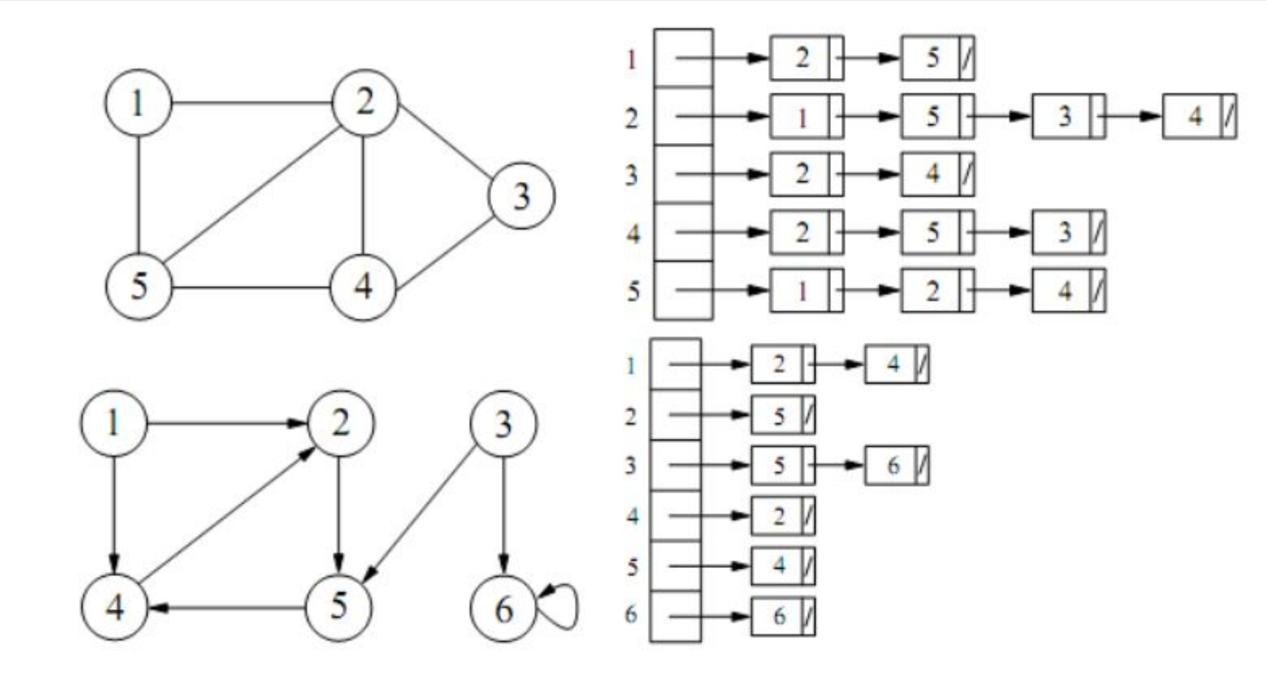
$$a_{ij} = \begin{cases} w(i,j) & \text{si } (i,j) \in E \\ 0 \ o \ \infty & \text{en cualquier otro caso} \end{cases}$$



	1	2	3	4	5	6
1	0	10	0	8	0	0
2	0	0	0	0	7	0
3	0	0	0	0	−1	15
4	0	12	0	0	0	0
5	0	0	0	9	0	0
6	0	0	0	0	0	9

LISTA DE ADYACENCIAS

- G =(V ,E): vector de tamaño |V|.
- Posición **i** → puntero a una lista enlazada de elementos (lista de adyacencia).
- Los elementos de la lista son los vértices adyacentes a **i**



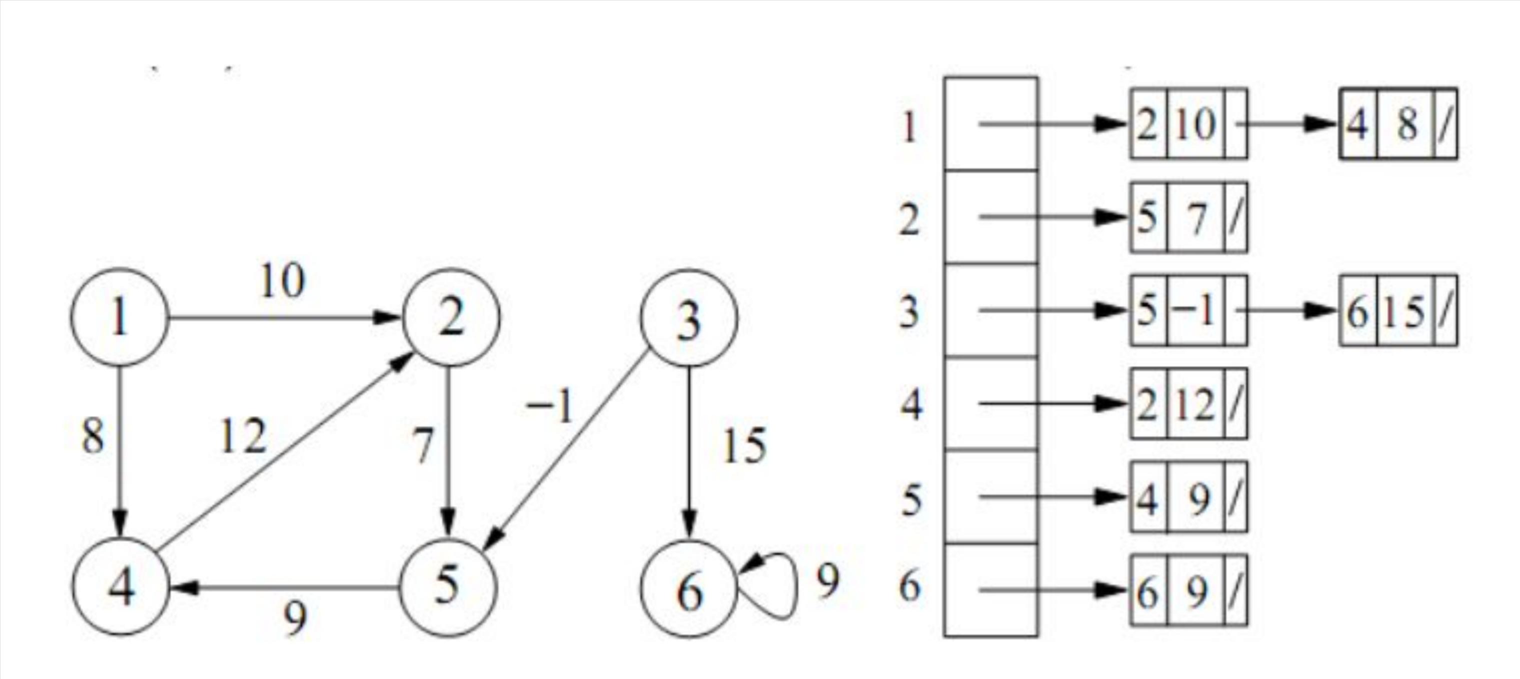
LISTA DE ADYACENCIAS (2)

- **Ventajas**
  - Si G es dirigido, la suma de las longitudes de las listas de adyacencia será |E|.
  - Si G es no dirigido, la suma de las longitudes de las listas de adyacencia será 2|E|.
  - Costo espacial, sea dirigido o no: O (|V|+|E|)
  - Representación apropiada para grafos con |E| menor que $|V|^2$.

- **Desventajas**
- Si se quiere comprobar si una arista (u,v) pertenece a E implica buscar **v** en la lista de adyacencia de u.
- Costo temporal T (|V|,|E|) será O (|V|).

LISTA DE ADYACENCIAS (3)

- Representación aplicada a Grafos pesados
- El **peso de (u,v)** se almacena en el nodo de v de la lista de adyacencia de **u**.



TIEMPO DE EJECUCIÓN

- Para grafos, el tiempo de ejecución se deja de calcular en función del valor de entrada N como hacíamos hasta ahora para listas, árboles, heap.
- En grafos no es suficiente tomar una única variable para evaluar el tiempo de ejecución de las operaciones y los algoritmos, se utiliza entonces una combinación de V y E donde V es el conjunto de vértices del grafo y E es el conjunto de aristas del grafo.
- Los tiempos de ejecución comienzan a ser del tipo
  - $O(V)$, $O(V+E)$, $O(V^2)$, etc.
- Se mostrarán ejemplos de estos usos en los diferentes algoritmos y operaciones que nos restan ver.

RECORRIDOS

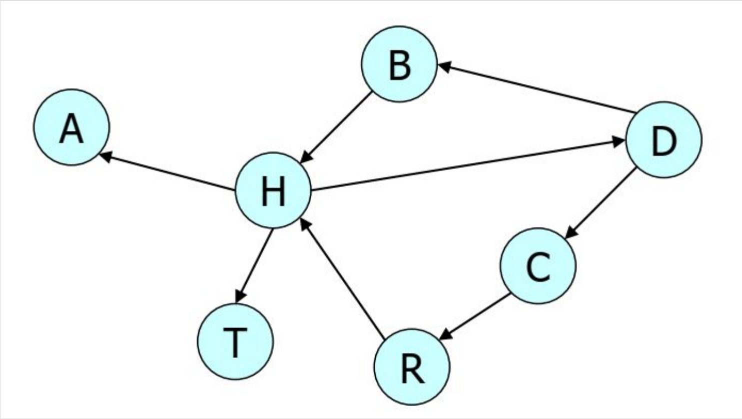
- Existen 2 tipos de recorridos en grafos
  - DFS – Depth First Search – Recorrido en profundidad
  - BFS – Breadth First Search – Recorrido en amplitud

DFS

- Estrategia conceptual:
  - Partir de un vértice determinado v.
  - Cuando se visita un nuevo vértice, explorar cada camino que salga de él.
  - Hasta que no se haya finalizado de explorar uno de los caminos no se comienza con el siguiente.
  - Un camino deja de explorarse cuando se llega a un vértice ya visitado.
  - Si existían vértices no alcanzables desde v el recorrido queda incompleto; entonces, se debe seleccionar algún vértice como nuevo vértice de partida, y repetir el proceso.

DFS (2)

- Es una generalización del recorrido preorden de un árbol.
- Ejemplo: tomando como vértice de partida D



  - Se muestra D C R H T A B

DFS (3)

- Esquema recursivo: dado G = (V, E)

  1. Marcar todos los vértices como no visitados.

  2. Elegir vértice **u** como punto de partida.

  3. Marcar **u** como visitado.

  4. Para todo **v** adyacente a **u,(u,v)** ∈ E, si **v** no ha sido visitado, repetir recursivamente (3) y (4) para **v**.

- Finalizar cuando se hayan visitado todos los nodos alcanzables desde u.
- Si desde u no fueran alcanzables todos los nodos del grafo: volver a (2), elegir un nuevo vértice de partida v no visitado, y repetir el proceso hasta que se hayan recorrido todos los vértices.

DFS (4)

- dfs (v: vértice)

  marca[v]:= visitado;

  para cada nodo **w** adyacente a **v**

    si w no está visitado

      dfs(w);

- main:dfs (grafo)

  inicializar **marca** en false (arreglo de booleanos);

  para cada vértice v del grafo

    si v no está visitado

      dfs(v);

DFS (5) – TIEMPO DE EJECUCIÓN

- G (V , E) se representa mediante listas de adyacencia.
- El método dfs(v) se aplica únicamente sobre vértices no visitados → sólo una vez sobre cada vértice.
- dfs(v) depende del número de vértices adyacentes que tenga (longitud de la lista de adyacencia). → el tiempo de todas las llamadas a dfs(v): O(|E|)
- añadir el tiempo asociado al bucle de main_dfs(grafo): O (|V|)).

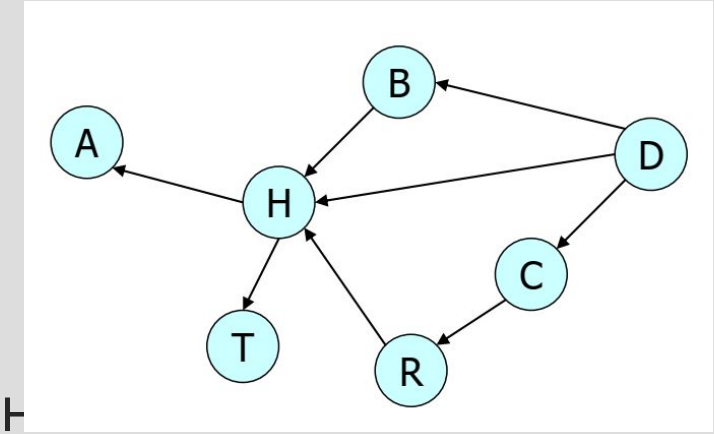
- **Tiempo del recorrido en profundidad es O(|V|+|E|)**

BFS

- Estrategia Conceptual:
  - Partir de algún vértice **u**, visitar **u** y, después, visitar cada uno de los vértices adyacentes a **u**.
  - Repetir el proceso para cada nodo adyacente a **u**, siguiendo el orden en que fueron visitados.

BFS (2)

- Es una generalización del recorrido por niveles de un árbol.
- Ejemplo: tomando como vértice de partida D



  - Se muestra D C H
  - En la cola auxiliar habremos procesado los vértices como D C H B R T A

BFS (3)

- Esquema iterativo: dado G = (V , E)
  1. Encolar el vértice origen **u**.
  2. Marcar el vértice **u** como visitado.
  3. Procesar la cola.
     4. Desencolar **u** de la cola
        5. Para todo adyacente a **u,(u,v)** ∈ E,
           6. si **v** no ha sido visitado
              7. encolar y visitar **v**
- Si desde **u** no fueran alcanzables todos los nodos del grafo: volver a (1), elegir un nuevo vértice de partida no visitado, y repetir el proceso hasta que se hayan recorrido todos los vértices.
- **Tiempo del recorrido en amplitud es O(|V|+|E|)**

RECORRIDOS - CONCLUSIONES

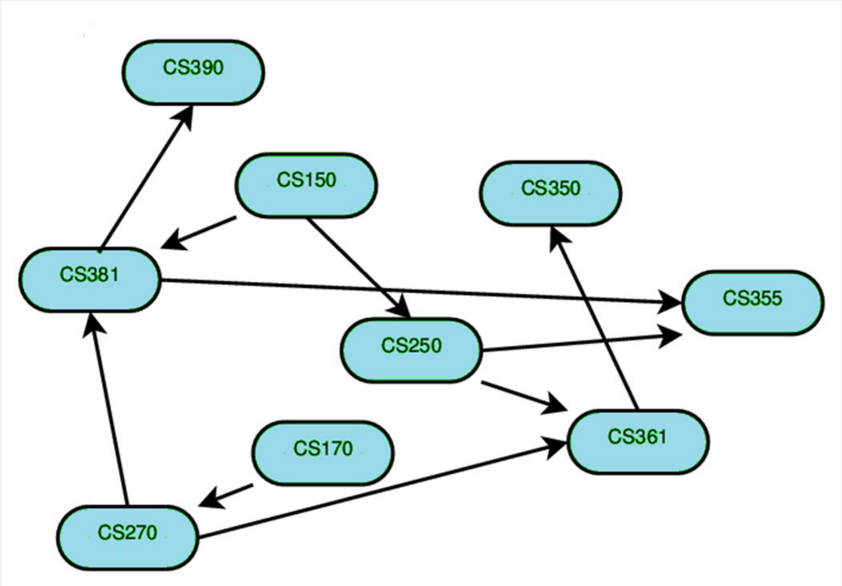
- Mismo recorrido y mismo grafo disparado desde vértices diferentes puede dar resultados de recorrido diferentes.
- DFS: recursivo, no usa estructura auxiliar, control de vértices visitados.
- BFS: iterativo, usa una cola como estructura auxiliar, control de vértices visitados.
- Tanto DFS como BFS aseguran que se procesa al menos 1 vez cada vértice del grafo, tenga o no tenga aristas.
- Tanto DFS como BFS tienen el mismo tiempo de ejecución, no hay uno más eficiente que otro, la clave está en elegir el recorrido adecuado según la situación o problema a resolver.
- No hay ninguna propiedad que indique que un algoritmo se resuelve más fácil o más difícil con BFS y DFS, aunque el nivel de complejidad de implementación puede ser notablemente más alto si se elige el recorrido equivocado.

ORDENACIÓN TOPOLÓGICA

- La ordenación topológica es una permutación: v_1, v_2, v_3, ..., $v_{|V|}$ de los vértices, tal que si $(v_i,v_j) \in E$, $v_i \neq v_j$, entonces v_i precede a v_j en la permutación.
- La ordenación no es posible si G es cíclico.
- La ordenación topológica no es única.
- Una ordenación topológica es como una ordenación de los vértices a lo largo de una línea horizontal, con los arcos de izquierda a derecha

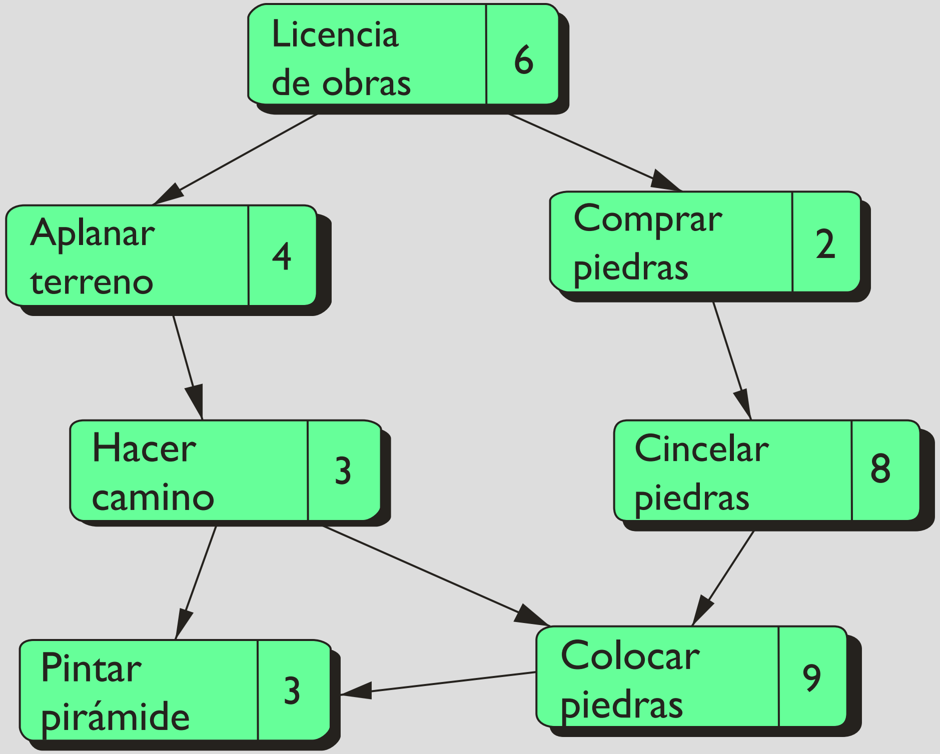
APLICACIÓN

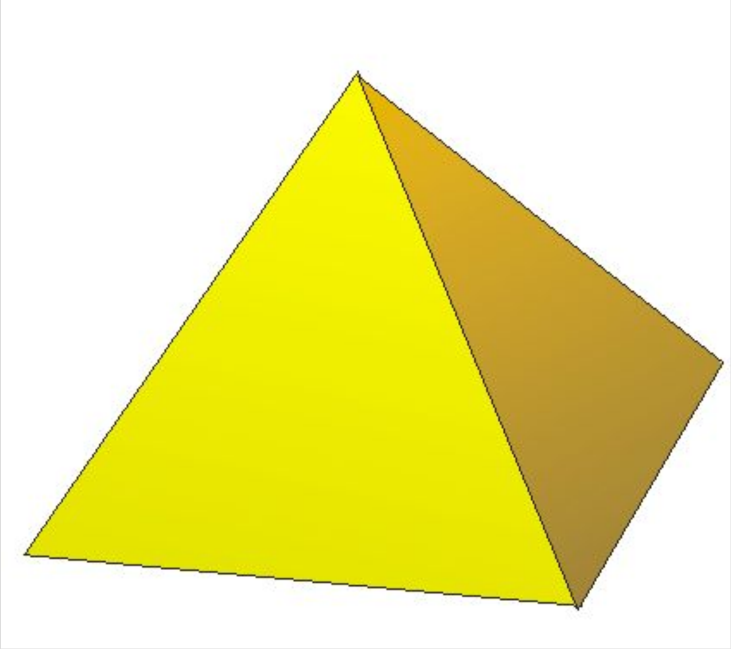
- Para indicar la precedencia entre eventos
  - Cursos conectados por aristas que representan la relación de "prerrequisito"



APLICACIÓN (2)

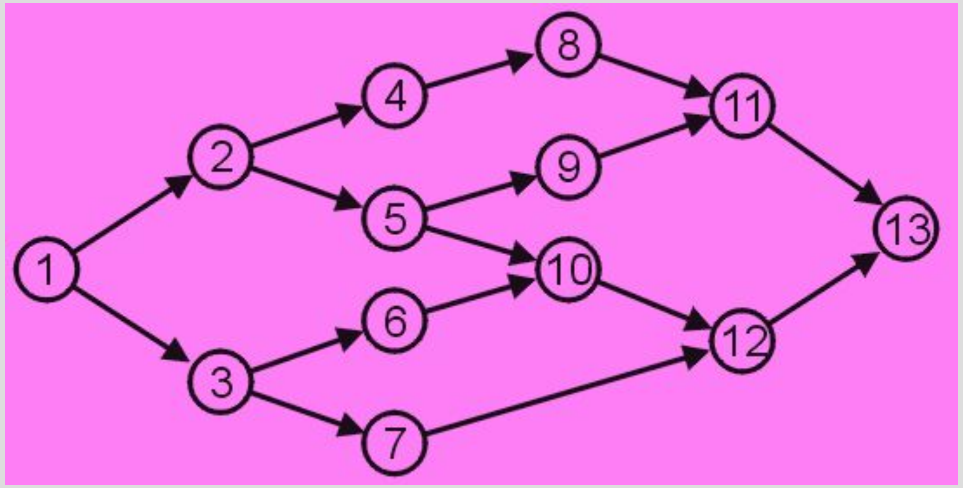
- Planificación de tareas





ALGORITMOS PARA SORT TOPOLÓGICO

- Dos ordenaciones válidas para el siguiente grafo:
- 1, 3, 2, 7, 6, 5, 4, 10, 9, 8, 12, 11, 13
- 1, 2, 4, 8, 5, 9, 11, 3, 6, 10, 7, 12, 13
- Y hay varias más…



SORT TOPOLÓGICO – VERSIÓN 1

- En esta versión el algoritmo utiliza un arreglo Grado_in en el que se almacenan los grados de entradas de los vértices y en cada paso se toma de allí un vértice con grado_in = 0.
- Estrategia general:

1-Seleccionar un vértice v con grado de entrada cero

2-Visitar v

3-"Eliminar" v, junto con sus aristas salientes

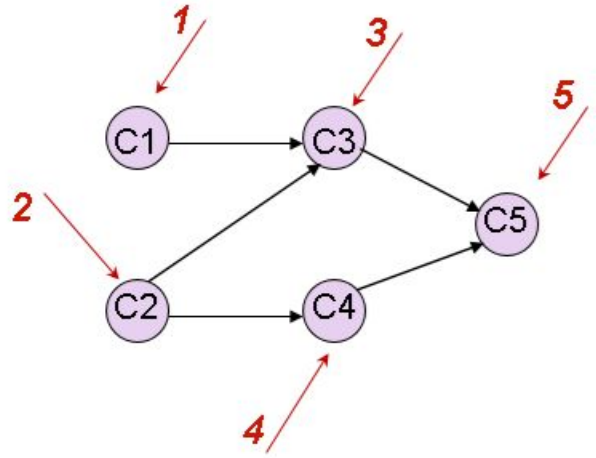
4-Repetir el paso 1 hasta seleccionar todos los vértices

SORT TOPOLÓGICO – VERSIÓN 1 (2)

- Tomando vértice con grado_in = 0 del vector Grado_in

Grado_in

C1	C2	C3	C4	C5
0	0	2	1	2
0	0	1	1	2
0	0	0	0	2
0	0	0	0	1
0	0	0	0	0



Sort Topológico :

C1 C2 C3 C4 C5

SORT TOPOLÓGICO – VERSIÓN 1 (3)

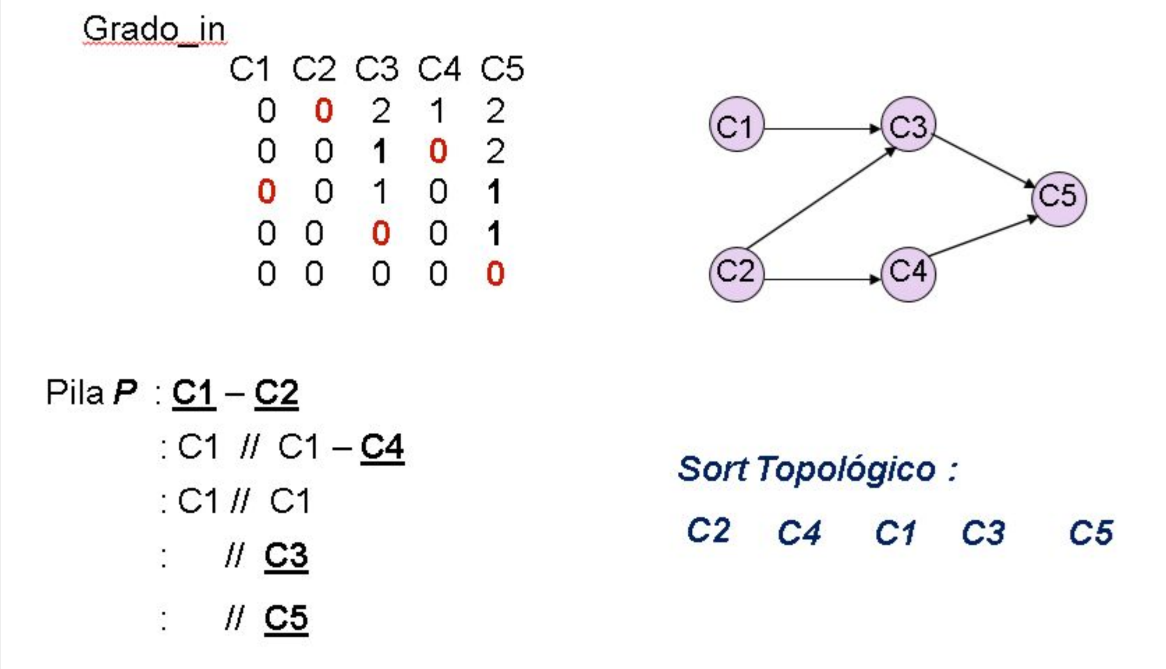
```
int sortTopologico( ){
    int numVerticesVisitados = 0;
    while(haya vertices para visitar){
        if(no existe vertice con grado_in = 0)
             break;
        else{
         seleccionar un vertice v con grado_in = 0;
         visitar v; //mandar a la salida
         numVerticesVisitados++;
         "borrar" v y todas sus aristas salientes;
       }
    }

   return numVerticesVisitados;
}
```

SORT TOPOLÓGICO – VERSIÓN 1 (4)

- En el else:
  - Búsqueda secuencial en el arreglo tiene O(|V|)
  - Decrementar el grado de entrada de los adyacentes a v tiene O (del número de aristas de v).
  - **Por lo tanto $T_{sortTopologico}$(V,E) es $O(|V|^2+|E|)$**

SORT TOPOLÓGICO – VERSIÓN 2

- En esta versión el algoritmo utiliza un arreglo Grado_in en el que se almacenan los grados de entradas de los vértices y una pila P (o una cola Q) en donde se almacenan los vértices con grados de entrada igual a cero.
- Tomando los vértices con grado_in = 0 de una Pila (o Cola)



SORT TOPOLÓGICO – VERSIÓN 2 (2)

```
int sortTopologico( ){
    int numVerticesVisitados = 0;
    while(hay vertices para visitar){
        if(no existe vértice con grado_in = 0)
            break;
        else{
         seleccionar un vértice v con grado_in = 0;
         visitar v; //mandar a la salida
         numVerticesVisitados++;
         “borrar” v y todas sus aristas salientes;
       }
    }

   return numVerticesVisitados;
}
```

SORT TOPOLÓGICO – VERSIÓN 2 (3)

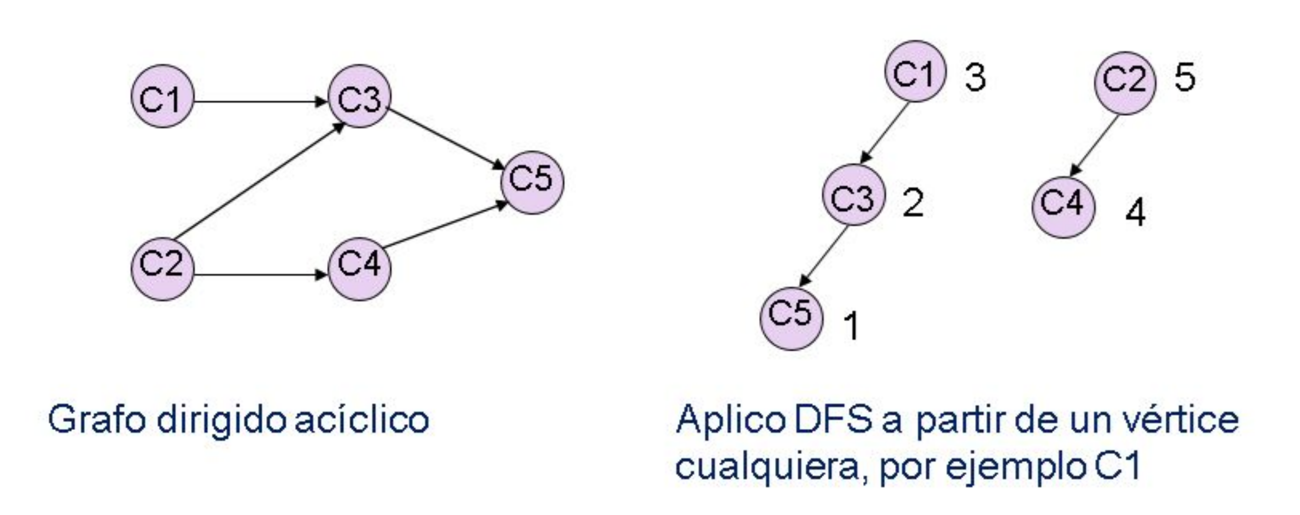
- En el else:
  - "seleccionar un vértice v con grado_in = 0" es tomar el vértice que este en el tope de la cola. Esto tiene O(1).
  - "todas sus aristas salientes" es decrementar el grado de entrada de los adyacentes de v. Si llego a 0 entonces encolarlo. Esto tiene O(del número de aristas de v)
  - **Por lo tanto $T_{sortTopologico}(V,E)$ es $O(|V|+|E|)$**

SORT TOPOLÓGICO – VERSIÓN 3

- En esta versión se aplica el recorrido en profundidad.
- Se realiza un recorrido DFS, marcando cada vértice en post-orden, es decir, una vez visitados todos los vértices a partir de uno dado, el marcado de los vértices en post-orden puede implementarse según una de las sig. opciones:
  a) numerándolos antes de retroceder en el recorrido; luego se listan los vértices según sus números de post-orden de mayor a menor.
  b) colocándolos en una pila P, luego se listan empezando por el tope.

SORT TOPOLÓGICO – VERSIÓN 3 (2)

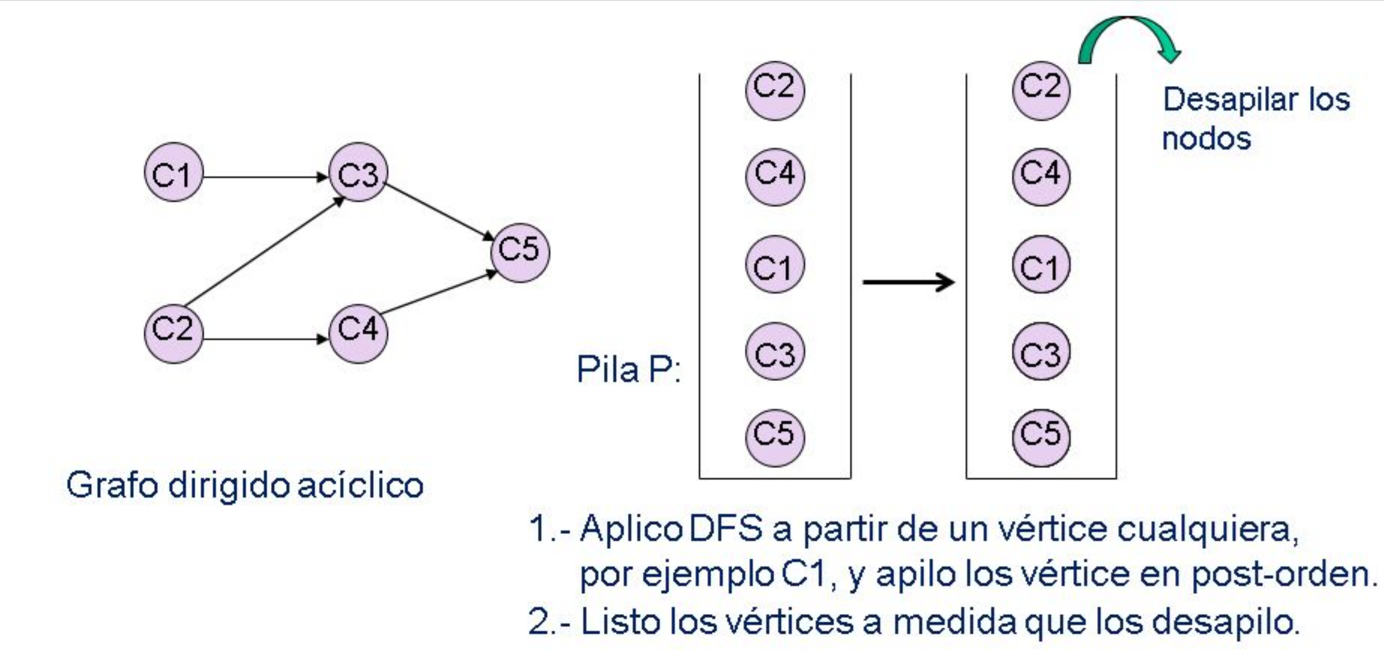
- Aplicando el recorrido en profundidad.
- Opción a) - numerando los vértices



- Ordenación Topológica: C2 C4 C1 C3 C5

SORT TOPOLÓGICO – VERSIÓN 3 (3)

- Aplicando el recorrido en profundidad.
- Opción b) - apilando los vértices



- Ordenación Topológica: C2 C4 C1 C3 C5

TRABAJO PRÁCTICO NRO. 6

- Volvemos al esquema de API propuesto por la cátedra, para realizar ejercicios de aplicación.
- Atención, los grafos dependen de interfaces (si, otra vez :P)
- a trabajar :)

DUDAS / CONSULTAS ☺