



# Algoritmos y Estructuras de Datos

Cursada 2020

Redictado

*Prof. Alejandra Schiavoni ([ales@info.unlp.edu.ar](mailto:ales@info.unlp.edu.ar))*

*Prof. Catalina Mostaccio ([catty@lifa.info.unlp.edu.ar](mailto:catty@lifa.info.unlp.edu.ar))*

*Prof. Claudia Queiruga ([claudiaq@info.unlp.edu.ar](mailto:claudiaq@info.unlp.edu.ar))*

*Prof. Pablo Iuliano ([piuliano@info.unlp.edu.ar](mailto:piuliano@info.unlp.edu.ar))*

# Agenda - Ordenación topológica

- Definición
- Ejemplos de aplicaciones de Grafos dirigidos Acíclicos (DAG)
- Algoritmos
  - Con complejidad  $O(|V|^2)$ : Implementación con Arreglo (versión 1)
  - Con complejidad  $O(|V| + |A|)$ 
    - Implementación con Pila o Cola (versión 2)
    - DFS (versión 3)

# Agenda - Ordenación topológica

## ➤ Definición

## ➤ Ejemplos de aplicaciones de Grafos dirigidos Acíclicos (DAG)

## ➤ Algoritmos

- Con complejidad  $O(|V|^2)$ : Implementación con Arreglo (versión 1)
- Con complejidad  $O(|V| + |A|)$ 
  - Implementación con Pila o Cola (versión 2)
  - DFS (versión 3)

# Definición

➤ *La ordenación topológica es una permutación:*

*$v_1, v_2, v_3, \dots, v_{|V|}$  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.*

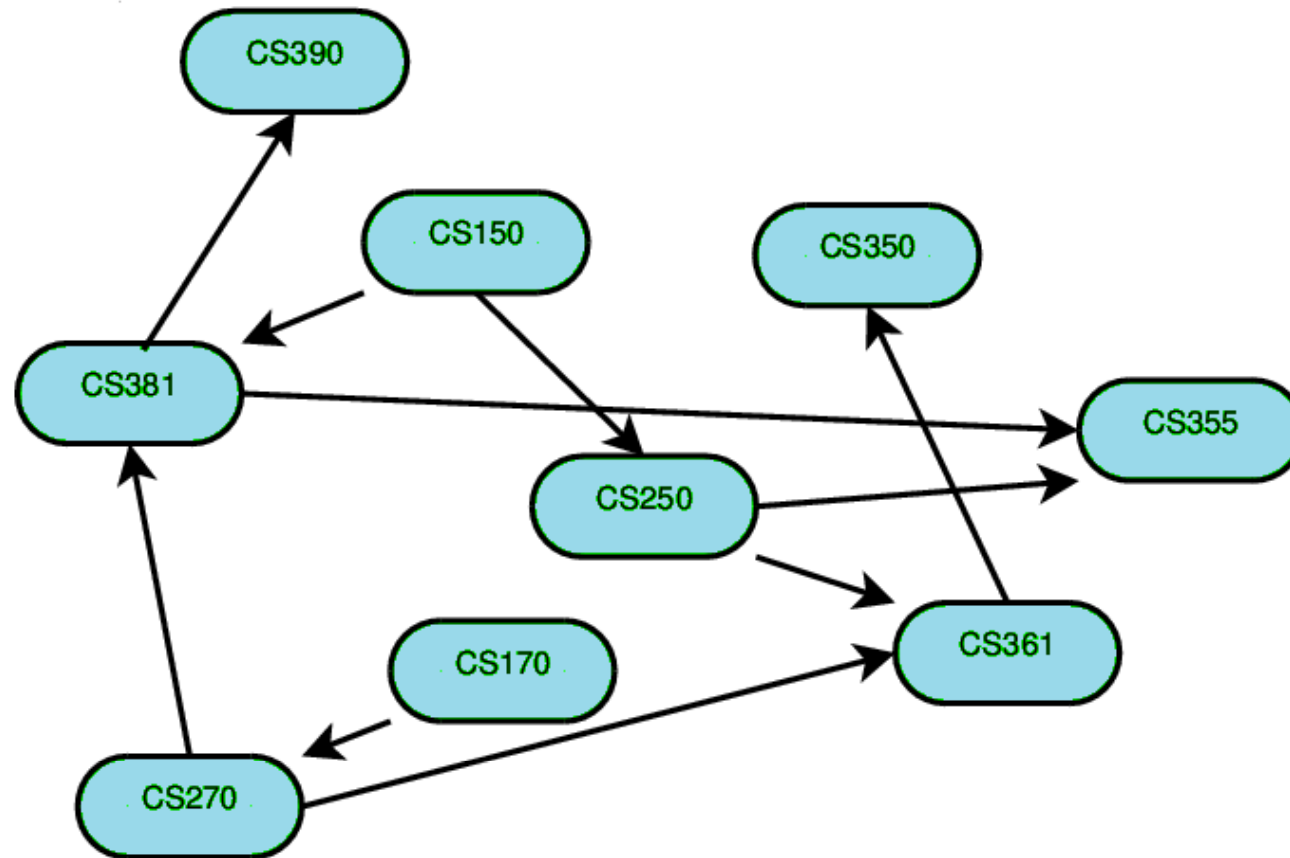
# Agenda - Ordenación topológica

- Definición
- **Ejemplos de aplicaciones de Grafos dirigidos Acíclicos (DAG)**
- Algoritmos
  - Con complejidad  $O(|V|^2)$ : Implementación con Arreglo (versión 1)
  - Con complejidad  $O(|V| + |A|)$ 
    - Implementación con Pila o Cola (versión 2)
    - DFS (versión 3)

# Aplicaciones

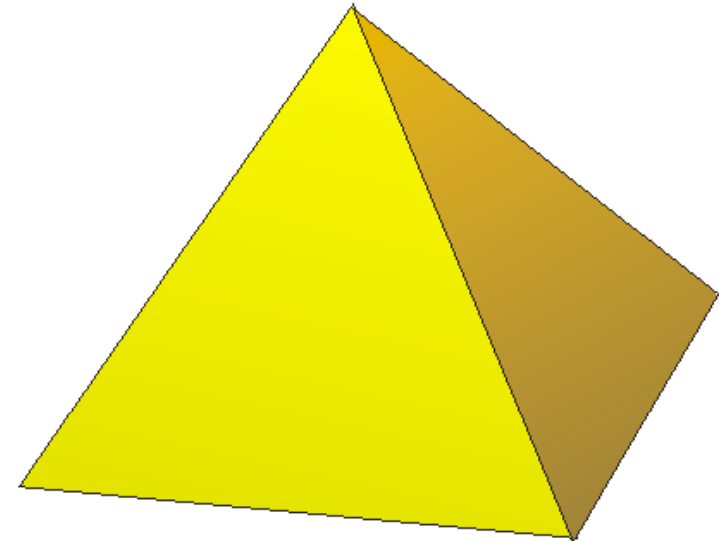
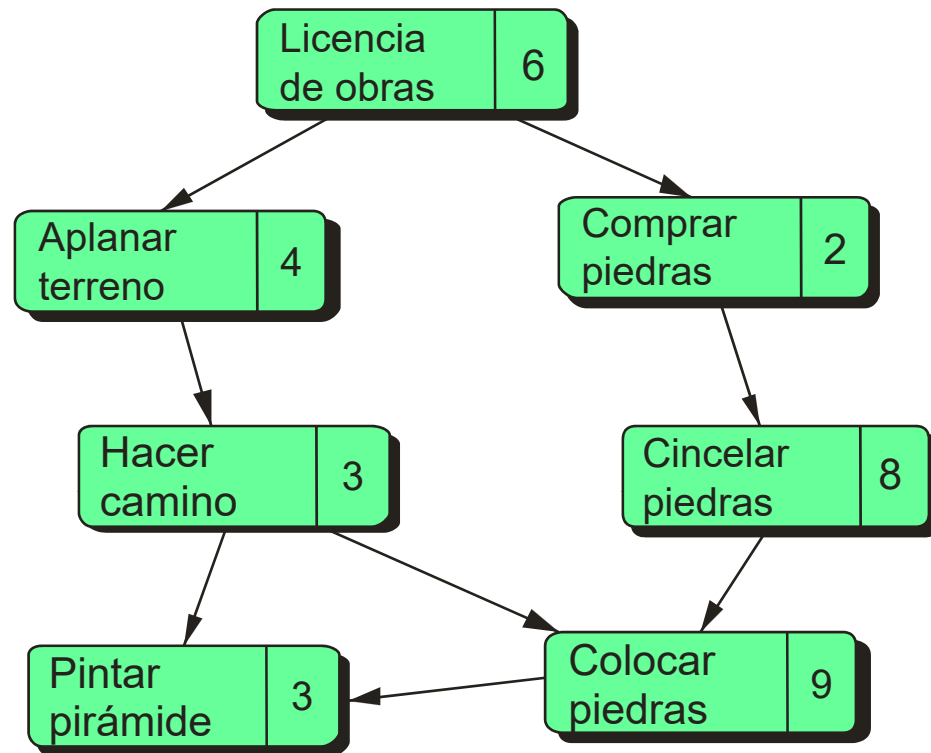
- *Para indicar la precedencia entre eventos*
- *Para planificación de tareas*
- *Organización curricular*

# Ejemplo 1: prerrequisito



**Cursos** conectados por aristas que representan la **relación** de “prerrequisito”

# Ejemplo 2: Planificación de tareas





# Agenda - Ordenación topológica

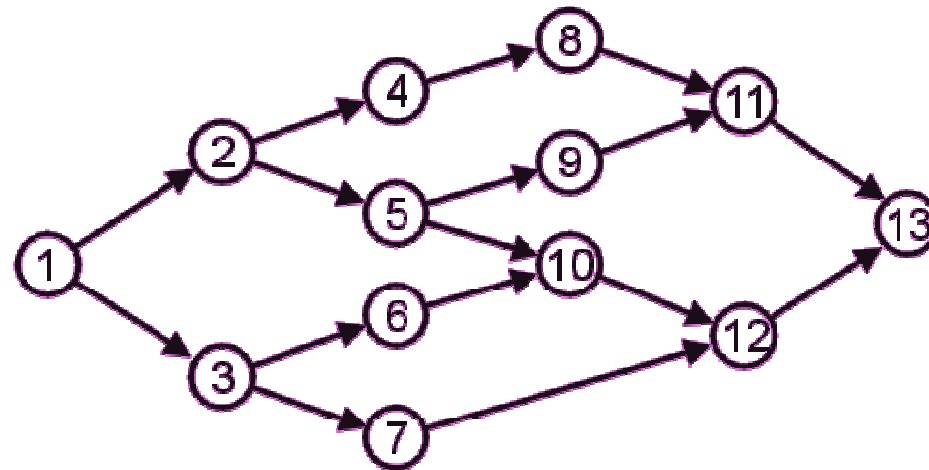
- Definición
- Ejemplos de aplicaciones de Grafos dirigidos Acíclicos (DAG)
- **Algoritmos**
  - Con complejidad  $O(|V|^2)$ : Implementación con Arreglo (versión 1)
  - Con complejidad  $O(|V| + |A|)$ 
    - Implementación con Pila o Cola (versión 2)
    - DFS (versión 3)

# Ordenación topológica

*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 muchas más.....*

# Ordenación topológica - (versión 1)

➤ *En esta versión el algoritmo utiliza un arreglo  $\text{Grado\_in}$  en el que se almacenan los grados de entradas de los vértices y en cada paso se toma de allí un vértice con  $\text{grado\_in} = 0$ .*

# Ordenación topológica - (versión 1)

*Pasos generales:*

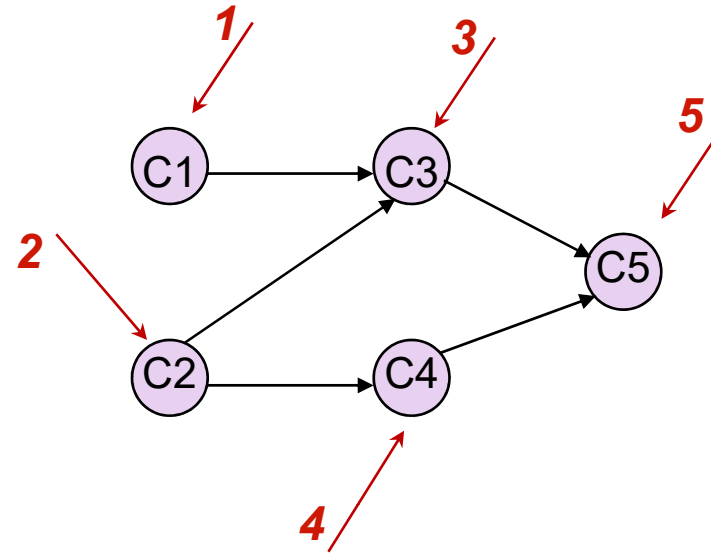
1. Seleccionar un vértice  $v$  con grado de entrada cero
2. Visitar  $v$
3. “Eliminar”  $v$ , junto con sus aristas salientes
4. Repetir el paso 1 hasta seleccionar todos los vértices

# Ordenación topológica - (versión 1)

→ Tomando vértice con  $\text{grado\_in} = 0$  del vector  $\text{Grado\_in}$

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**

# Ordenación topológica - (versión 1)

```
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;
}
```

# Ordenación topológica - (versión 1)

```
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;  
}
```

Búsqueda  
secuencial  
en el  
arreglo

Decrementar  
el grado de  
entrada de  
los  
adyacentes  
de v

# Ordenación topológica - (versión 1)

*El tiempo total del algoritmo es:*

```
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;  
}
```

$O(|V|)$

Orden del  
número de  
aristas de v



# Ordenación topológica - (versión 1)

*El tiempo total del algoritmo es:*

```
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;
}
```

$O(|V|)$

$O(|V|^2 + |E|)$

Orden del número de aristas de v

# Ordenación topológica - (versión 2)

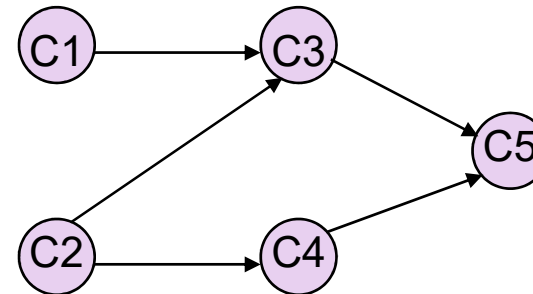
➤ *En esta versión el algoritmo utiliza un arreglo  $\text{Grado\_in}$  en el que se almacenan los grados de entradas de los vértices y una pila  $P$  (o una cola  $Q$ ) en donde se almacenan los vértices con grados de entrada igual a cero.*

# Ordenación topológica - (versión 2)

→ Tomando los vértices con  $\text{grado\_in} = 0$  de una Pila (o Cola)

Grado\_in

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



Pila **P** : **C1** – **C2**  
: C1 // C1 – **C4**  
: C1 // C1  
: // **C3**  
: // **C5**

**Sort Topológico :**

**C2 C4 C1 C3 C5**

# Ordenación topológica - (versión 2)

```
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;
}
```

# Ordenación topológica - (versión 2)

```
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;  
}
```

Tomar el  
vértice de la  
cola

Decrementar el  
grado de  
entrada de los  
adyacentes de  
v. Si llegó a 0,  
encolarlo

# Ordenación topológica - (versión 2)

*El tiempo total del algoritmo es:*

```
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;
}
```

$O(1)$

Orden del  
número de  
aristas de v

# Ordenación topológica - (versión 2)

*El tiempo total del algoritmo es:*

```
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;
}
```

$O(1)$

$O(|V|+|E|)$

Orden del número de aristas de  $v$

# Ordenación topológica - (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.*

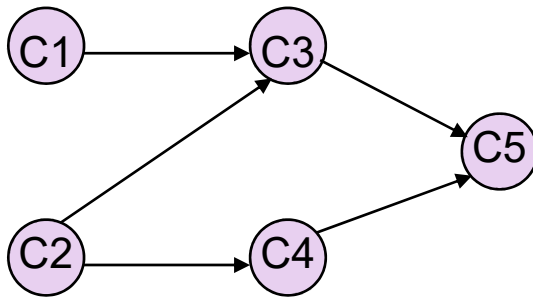
*b) colocándolos en una pila P, luego se listan empezando por el tope.*



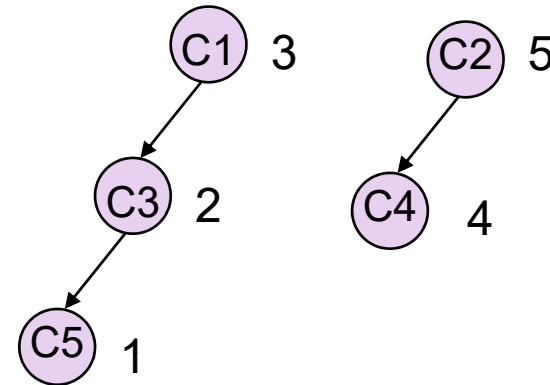
# Ordenación topológica - (versión 3)

→ *Aplicando el recorrido en profundidad.*

Opción **a)** - numerando los vértices



Grafo dirigido acíclico



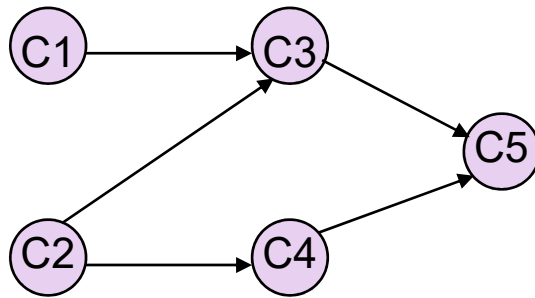
Aplico DFS a partir de un vértice cualquiera, por ejemplo C1

**Ordenación Topológica: C2 C4 C1 C3 C5**

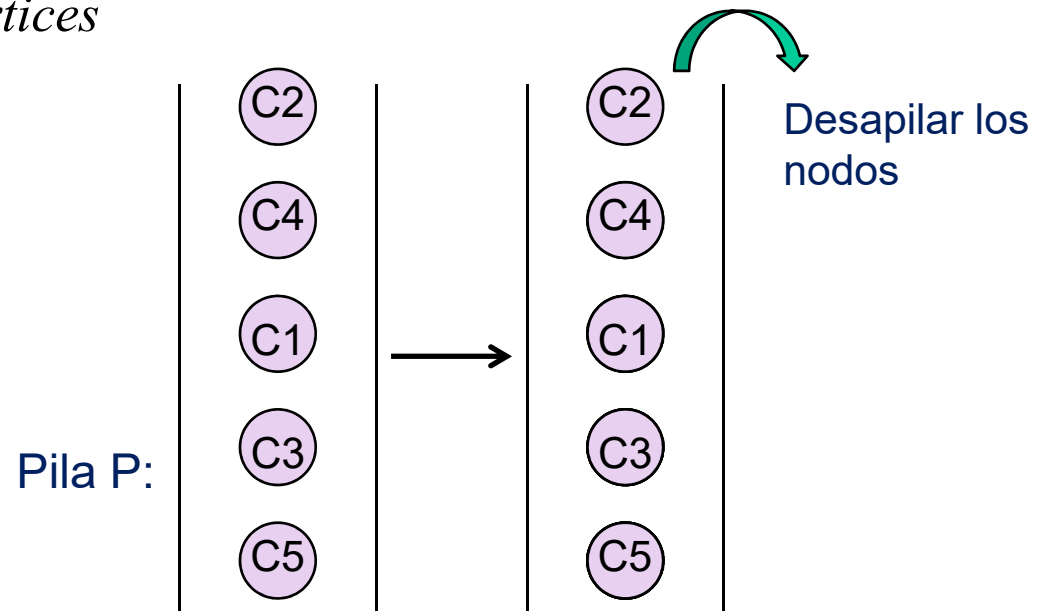
# Ordenación topológica - (versión 3)

→ *Aplicando el recorrido en profundidad.*

Opción **b)** - *apilando los vértices*



Grafo dirigido acíclico



- 1.- Aplico DFS a partir de un vértice cualquiera, por ejemplo C1, y apilo los vértice en post-orden.
- 2.- Listo los vértices a medida que los desapilo.

**Ordenación Topológica: C2 C4 C1 C3 C5**