



# **Algoritmos y Estructuras de Datos**

Cursada 2013

**Prof. Catalina Mostaccio  
Prof. Alejandra Schiavoni**

*Facultad de Informática - UNLP*



# GRAFOS

# Agenda - Grafos

1. Ejemplos y terminología
2. Representaciones
3. Recorridos
4. Sort topológico

# Agenda - Grafos

- ❖ Sort topológico
  - ❖ Definición
  - ❖ Ejemplos de aplicaciones de DAGs
  - ❖ Algoritmos
    - ❖  $O(|V|^2)$  Implementación con Arreglo (versión 1)
    - ❖  $O(|V| + |A|)$ 
      - ❖ Implementación con Pila o Cola (versión 2)
      - ❖ DFS (versión 3)

## Sort topológico - 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.

## Sort topológico - Aplicaciones

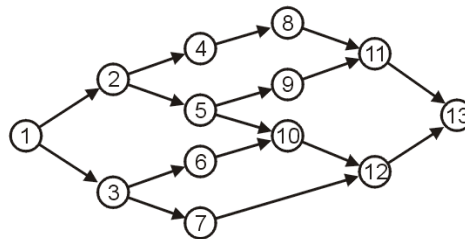
- Para indicar la precedencia entre eventos
- Para planificación de tareas
- Organización curricular

## 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



## Sort topológico - (versión 1)

→ *Aplicando el recorrido en amplitud*

➤ *Este algoritmo utiliza un arreglo `Grado_in` en el que se almacenan los grados de entradas de los vértices.*

## Sort topológico - (versión 1)

*Pasos generales:*

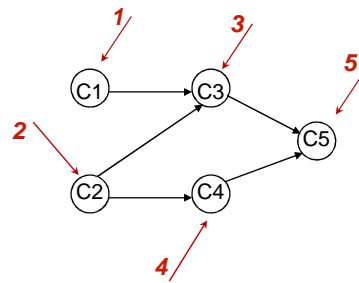
1. Seleccionar un vértice  $v$  con grado de entrada cero
2. Visitar  $v$
3. Eliminar  $v$ , junto con sus aristas salientes
4. Repetir el paso 1. hasta seleccionar todos los vértices

## Sort topológico - (versión 1)

→ *Aplicando el recorrido en amplitud*

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)

```
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

## Sort topológico - (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

## Sort topológico - (versión 2)

→ Aplicando el recorrido en amplitud y una Pila (o Cola)

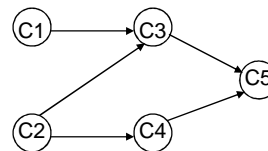
➤ Este algoritmo utiliza un arreglo *Grado\_in* en el que se almacenan los grados de entradas de los vértices y una pila *P* (o una cola *Q*) en donde se almacenan los grados de entrada igual a cero de los vértices.

## Sort topológico - (versión 2)

→ Aplicando el recorrido en amplitud y una Pila (o Cola)

Grado\_in

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



Pila *P* : C1 – C2

: C1 // C1 – C4

: C1 // C1

: // C3

: // C5

Sort Topológico :

**C2 C4 C1 C3 C5**

## Sort topológico - (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

## Sort topológico - (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;
}
```

$O(1)$

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

Orden del  
número de  
aristas de v



## Sort topológico - (versión 3)

→ Aplicando el recorrido en profundidad.

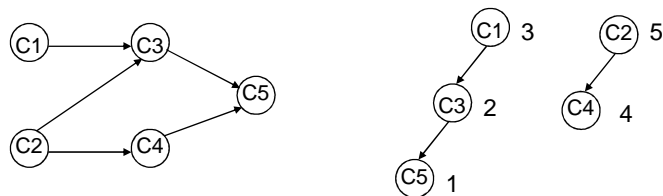
➤ Se realiza un recorrido DFS, marcando cada vértice en post-orden, es decir, una vez visitados todos los vértices a partir de uno dado, éste es marcado con un número antes de retroceder en el recorrido.

➤ Se listan los números según sus números de post-orden de mayor a menor.

➤ El marcado de los vértices en post-orden puede implementarse colocando los vértices en una pila *P*, luego se listan empezando por el tope.

## Sort topológico - (versión 3)

→ Aplicando el recorrido en profundidad.



Dag G

DFS(G)

**Sort Topológico : C2 C4 C1 C3 C5**