- Grafos Dirigidos
- Orden Topológico

Bibliografía: “Introduction to Algorithms”. Third Edition.

The MIT Press. Massachusetts Institute of Technology. Cambridge, Massachusetts 02142.

http://mitpress.mit.edu

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Definición Grafo Dirigido

Definición de **Grafo dirigido**
$G=(V, A)$

V: Conjunto finito de vértices
A: Conjunto de **arcos** dirigidos

- Cada **arco** de A es un **par ordenado** de vértices (v, w)
- El **arco** (v, w) suele expresarse de la forma $v \rightarrow w$
- Se dice que el **arco** $v \rightarrow w$, va de v a w y que w y v son **vértices adyacentes**

Definición de camino – camino simple

Un **camino** 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 de A

Se dice que el camino va del vértice v_1 al v_n

Empieza en v_1, pasa por los vértices $v_2, v_3, \ldots, v_{n-1}$ y termina en el v_n

Un **camino es simple** si todos sus vértices excepto, tal vez, el primero y el último, son distintos

Un solo vértice denota un camino de longitud 0 de *v* a *v*

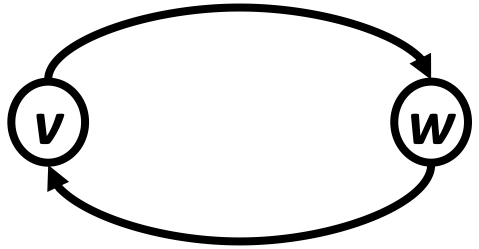
Definición de ciclo – ciclo simple

Un **ciclo** es un camino de longitud, por lo menos 2, que empieza y termina en el mismo vértice

En **MD** al **ciclo** lo llaman **camino cerrado** → empieza y termina en el mismo vértice y puede o no pasar más de una vez por el mismo vértice

Un **ciclo simple** es un **camino simple** de longitud, **por lo menos 2**, que empieza y termina en el mismo vértice

(v, w, v) es un **ciclo simple**.
camino simple de longitud 2, que involucra a los vértices *v* y *w* y a los **arcos** *v* → *w* y *w* → *v*

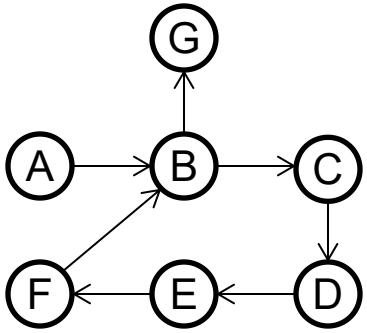


En **MD** al **ciclo simple** le llaman **ciclo** → **camino cerrado** de longitud ≥2 y simple (no repite vértices)

Un **grafo dirigido** es **acíclico** si no posee ciclos

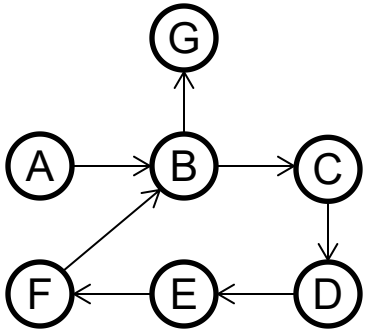
Otras definiciones para Grafos Dirigidos

- **cadena**: es **un camino** que **no repite arcos**



(B, C, D, E)
B→C, C→D, D→E

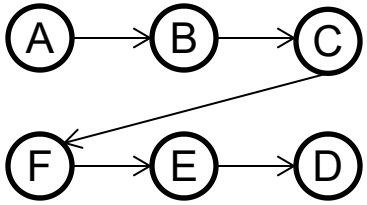
- **cadena de Euler**: es una **cadena** que recorre todos los arcos del grafo



(A, B, C, D, E, F, B, G)
A→B, B→C, C→D, D→E, E→F, F→B, B→G

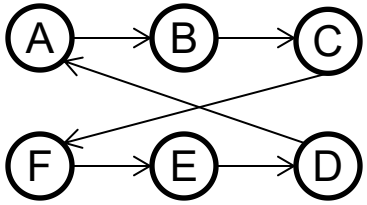
Otras definiciones para Grafos Dirigidos

- **camino de Hamilton**: un **camino simple** que recorre todos los vértices del grafo



(A, B, C, F, E, D)

- **ciclo de Hamilton**: es un **camino de Hamilton** que empieza y termina en el mismo vértice



(A, B, C, F, E, D, A)

Representaciones para Grafos Dirigidos

- **Matriz de Adyacencia:**

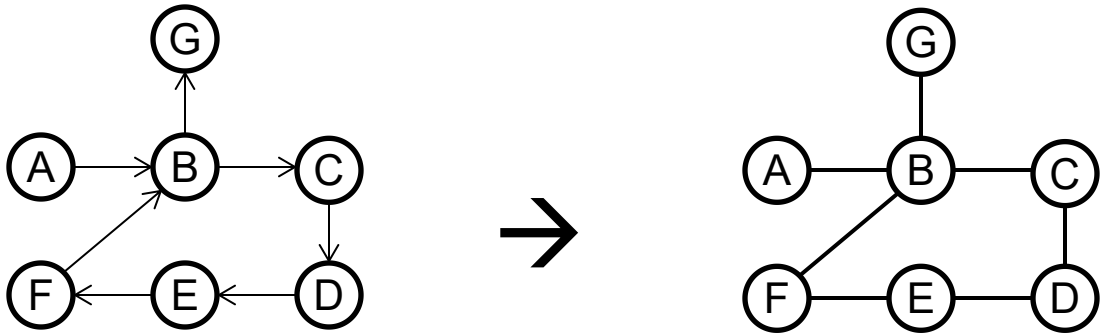
Similar a lo visto para los grafos no dirigidos.
Al ser el grafo dirigido, **se rompe la simetría** que había en la matriz cuando se trataba de grafos no dirigidos

- **Listas de Adyacencia:**

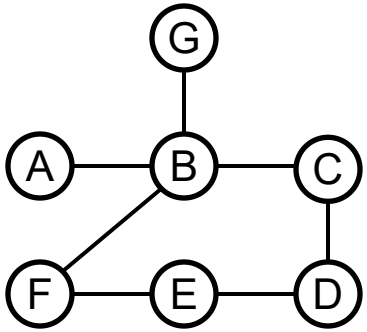
Similar a lo visto para los grafos no dirigidos.

Grafos Dirigidos Subyacentes y Conexos

Grafo subyacente de un grafo dirigido: se obtiene, reemplazando cada **arco** del mismo, por una **arista**

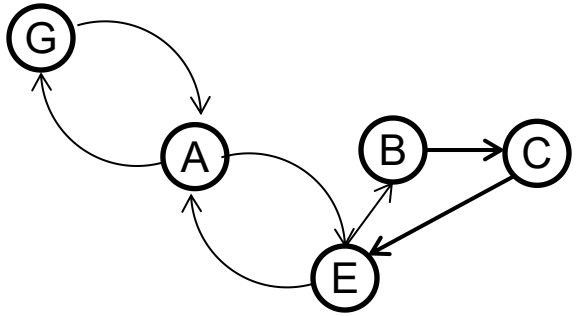


Un **grafo dirigido** es **conexo** si su **grafo subyacente** es **conexo**

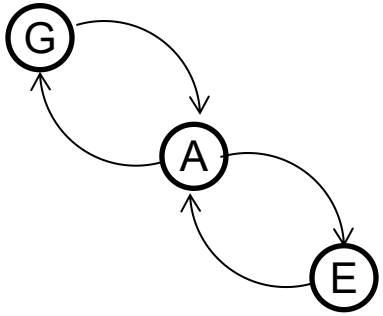


Grafo Dirigido – unilateralmente y fuertemente conexo

Un grafo dirigido es **unilateralmente conexo** si para cada par de vértices $u,v \in V$, existe un camino de u a v ó de v a u *(el "ó" implica que puede haber camino en ambos sentidos)*



Un grafo dirigido es **fuertemente conexo** si entre cualquier par de vértices **hay un camino, en ambas direcciones**, que los une



Recorridos de Grafos Dirigidos

BFS y DFS:

Los algoritmos vistos son aplicables a grafos dirigidos

En el **DFS,** para **grafos dirigidos**, al construir el bosque abarcador en profundidad, la clasificación de los arcos se amplía, además de los **arcos de árbol *(tree edge)*** y de **retroceso *(back edge)***, aparecen dos nuevos tipos de arcos:

- ***arcos de cruce***
- ***arcos de avance*** (o *hacia adelante*)

DFS - Grafos Dirigidos / clasificación de los arcos

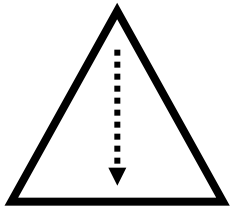
La clasificación de los arcos se establece de la manera siguiente:

- Durante la búsqueda en profundidad se construye el **bosque abarcador**. Los arcos que **aparecen representados en el bosque** son los **arcos de árbol**

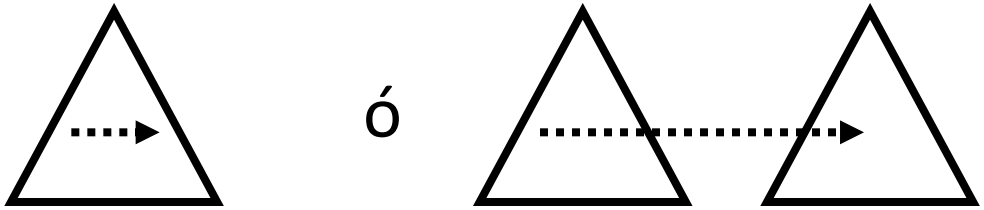
- Una vez establecido el bosque, en el grafo original **se detectan todos los arcos que no fueron representados en el bosque abarcador**. **De acuerdo a la disposición de sus vértices en el bosque**, se clasifican dichos **arcos** y se representan dentro del propio bosque por líneas discontinuas

DFS - Grafos Dirigidos / clasificación de los arcos

arcos de avance (*forward edge*): van **de un ancestro a un descendiente propio** en un cierto **árbol** del **bosque abarcador**



arcos cruzados (*cross edge*): son **arcos** que irían de un vértice a otro que no es, ni descendiente propio, ni antecesor en el bosque abarcador (o sea, ni hacia abajo ni hacia arriba). Pueden ir de una rama a otra de un mismo árbol o de árboles diferentes



NOTA: la dirección del arco puede ser en cualquier sentido

START

1 → 1 → 2 tree

2 → 2 → 3 tree

3 → 3 → 4 tree

4 → 4 → 2 back

2

1 → 1 → 5 tree

5 → 5 → 6 tree

6 → 6 → 8 tree

8

8

6 → 6 → 7 tree

7

7

7

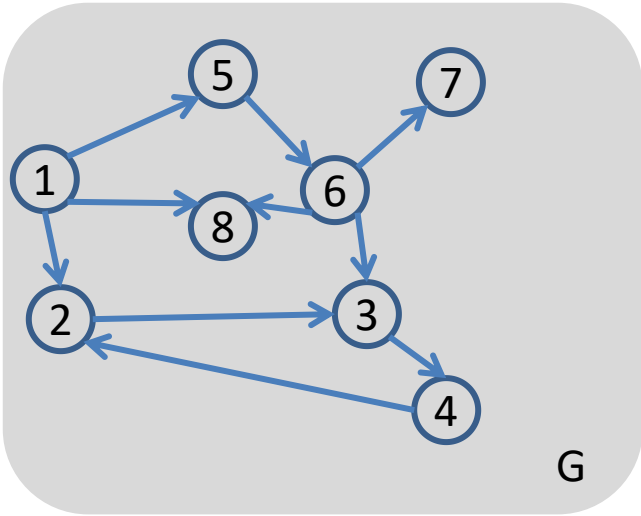
6 → 6 → 3 cross

6 d[6]> d[3]

5

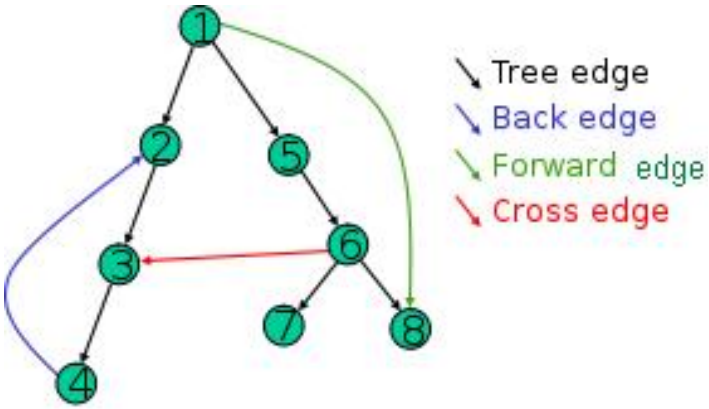
1 → 1 → 8 forward

1 d[1]<d[8]



DFS

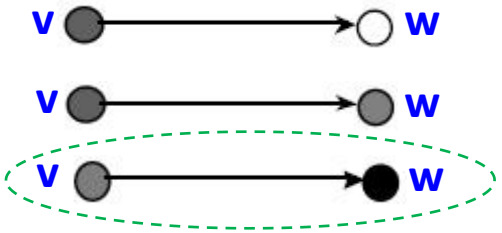
d[1]=1 f[1]=16
d[2]=2 f[2]=7
d[3]=3 f[3]=6
d[4]=4 f[4]=5
d[5]=8 f[5]=15
d[6]=9 f[6]=14
d[8]=10 f[8]=11
d[7]=12 f[7]=13



DFS - Grafos Dirigidos / clasificación de los arcos

DFS puede ser modificado para clasificar los **arcos** la primera vez que estos se alcanzan por el algoritmo

El arco *e* = <*v, w*> se clasifica, en función del color de *w*, cuando *e* se explora por primera vez (o sea, *v* es el nodo en curso) de la siguiente forma:



arco de árbol – si ***w*** is blanco
arco de retroceso – si ***w*** es **gris**
arco de avance o cruce – si ***w*** es **negro**:

→ ***de avance*** – si ***w*** es **negro** y **d[v] < d[w]**
(***w*** fue descubierto después de ***v***)

→***de cruce*** – si ***w*** es **negro** y **d[v] > d[w]**
(***v*** fue descubierto después de ***w***)

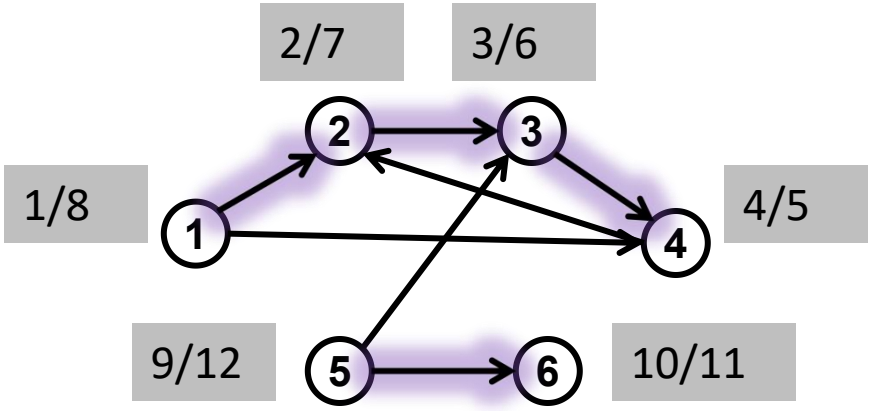
Una reflexión

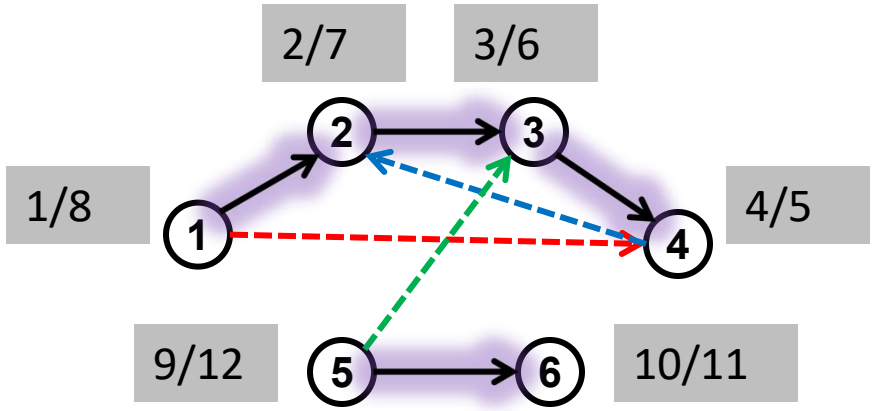
Como hemos visto, en un **grafo dirigido**, es posible clasificar los arcos en un recorrido DFS como "**árbol**", "**de retroceso**", "**de avance**" y "**de cruce**". ¿Por qué en un **grafo no dirigido**, al hacer un recorrido DFS no aparecen aristas "**de avance**" ni "**de cruce**"?

IDEA: Al modificar el DFS para clasificar las aristas cuando estas se alcanzan por el algoritmo. Cada arista $e = \langle v, w \rangle$ se clasifica, en función del color de w, cuando e se explora **por primera vez.**

En un Grafo No Dirigido puede presentarse el siguiente caso cuando e se explora por primera vez, o sea, desde v se descubre w?

DFS - Grafos Dirigidos / clasificación de los arcos





de avance
retroceso
de cruce

Orden Topológico

PROBLEMA

Un proyecto grande suele dividirse en una colección de tareas más pequeñas, algunas de las cuales han de realizarse respetando ciertos órdenes de precedencia para que se pueda culminar el proyecto total

Cómo representar las precedencias necesarias entre dichas tareas aplicando la **Teoría de Grafos** ?

Orden Topológico

Ejemplo 1 :

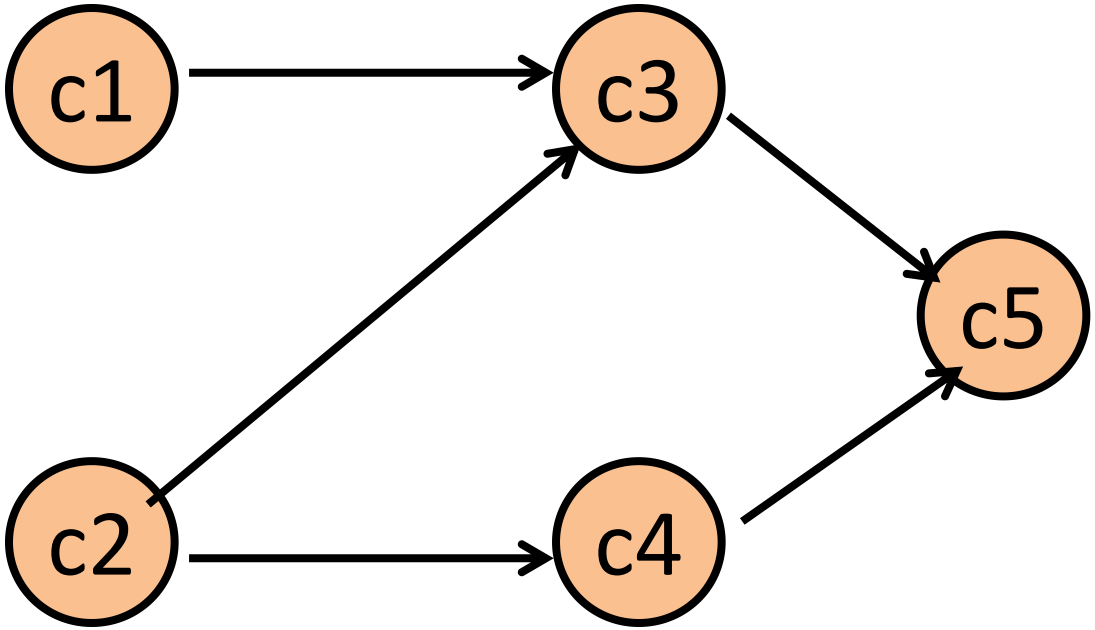
Una carrera puede incluir asignaturas que, para ser cursadas, es necesario que otras, consideradas prerrequisitos, sean aprobadas con anterioridad

Para modelar estas situaciones se pueden utilizar los **Grafos Dirigidos Acíclicos (DAG),** los cuales pueden usarse, en determinados proyectos, para indicar precedencia entre los eventos que intervienen en el mismo

Orden Topológico

En el grafo se muestra la estructura de prerrequisitos de **cinco cursos**, de manera tal que:

existe un **arco** del *curso **u*** al *curso **v***, si el *curso **u*** es un prerrequisito del *curso **v***



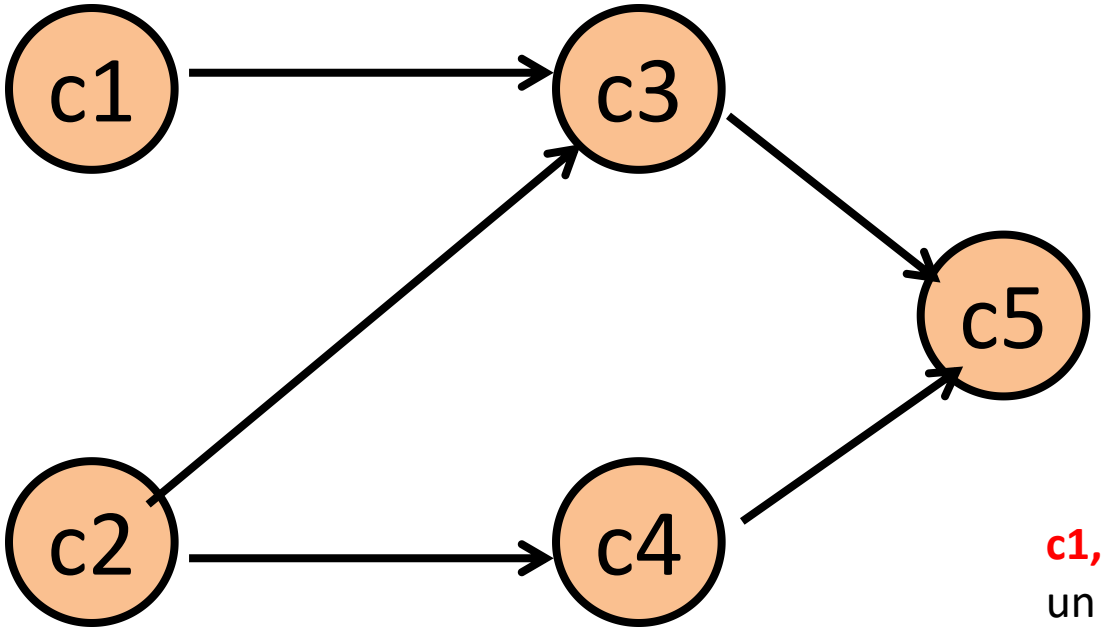
Orden Topológico

Definición:

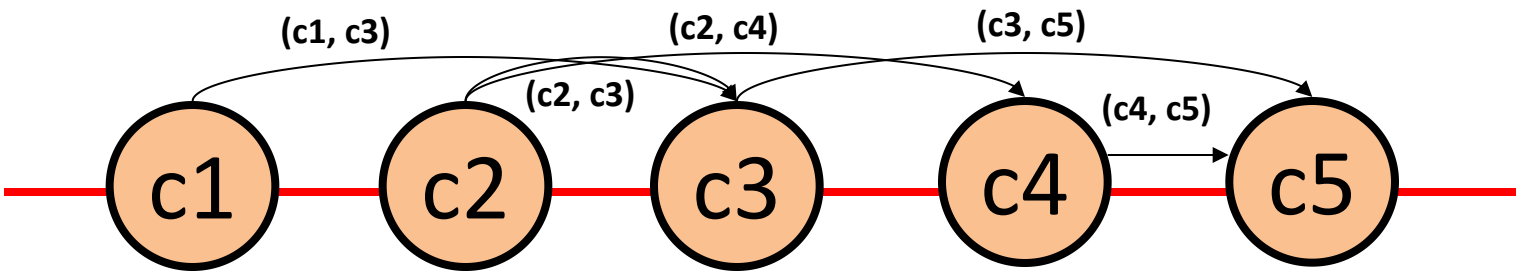
Un **orden topológico** de un grafo dirigido y acíclico *(suele llamárseles DAG a este tipo de Grafos)* G=(V, E) es **una ordenación lineal de todos sus vértices**, de manera tal que si el arco **(*u*, *v*)** está en E, entonces el vértice ***u*** aparece antes que el ***v*** en el orden lineal establecido

*Si en el grafo **HAY CICLOS**, entonces no tiene sentido hablar de esta ordenación pues la misma no es posible de obtener*

Orden Topológico



c1, c2, c3, c4 y **c5** es un orden topológico



Al tomar los cursos de esta forma, se puede satisfacer la relación de prerrequisitos dada

La **ordenación Topológica** puede verse como una ordenación de los vértices del grafo a través de una línea horizontal de manera tal que la dirección de todos los arcos es de *izquierda* a *derecha*

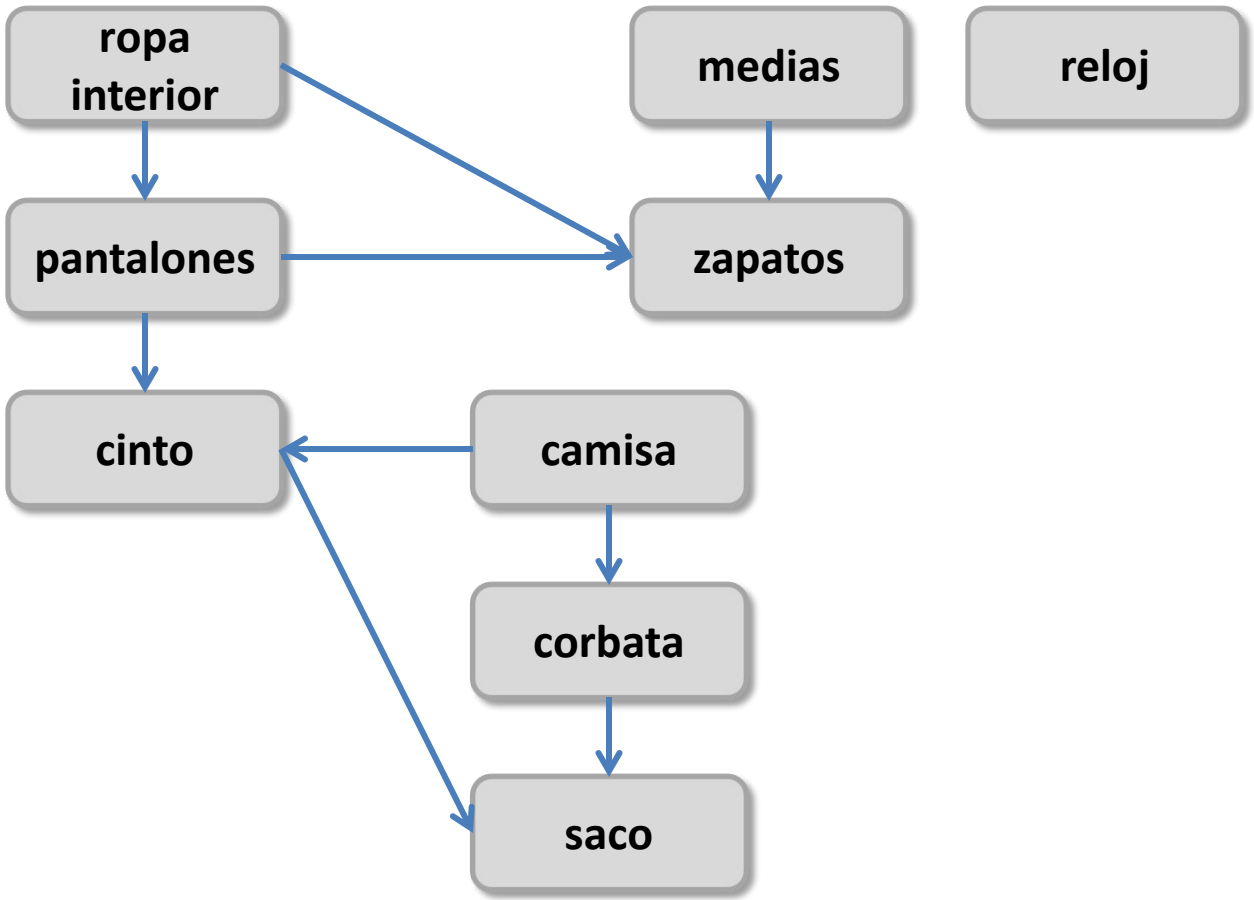
Orden Topológico

Ejemplo 2:

Precedencia en relación al modo de vestirse un hombre tras haberse levantado en la mañana

Haciendo una simple modificación al algoritmo de búsqueda en profundidad, podemos obtener un algoritmo para la **ordenación topológica**

Orden Topológico

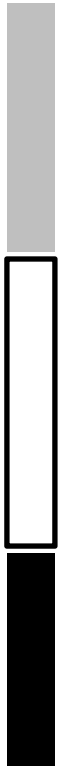


DFS - Orden Topológico

```
DFS-VISIT-TopologicalSort(G, u)
  u ← visited
  time = time + 1
  d[u] = time

  for each v ∈ Adj[u]
    do if v not visited
       π[v] ← u
       DFS-VISIT(G, v)

  time = time + 1
  f[u] = time
  stack.Push[u]
  return
```



stack.Push[u] → O(1)

DFS –VISIT- TS → Θ(*V* + *E*)

NOTAS

La pila "***stack***" devuelve los vértices en orden cuando se hace *pop()*

Si se utiliza una **Lista** hay que "invertirla" o insertar cada vértice por el inicio de la Lista

Siempre $d[u] \neq f[u]$

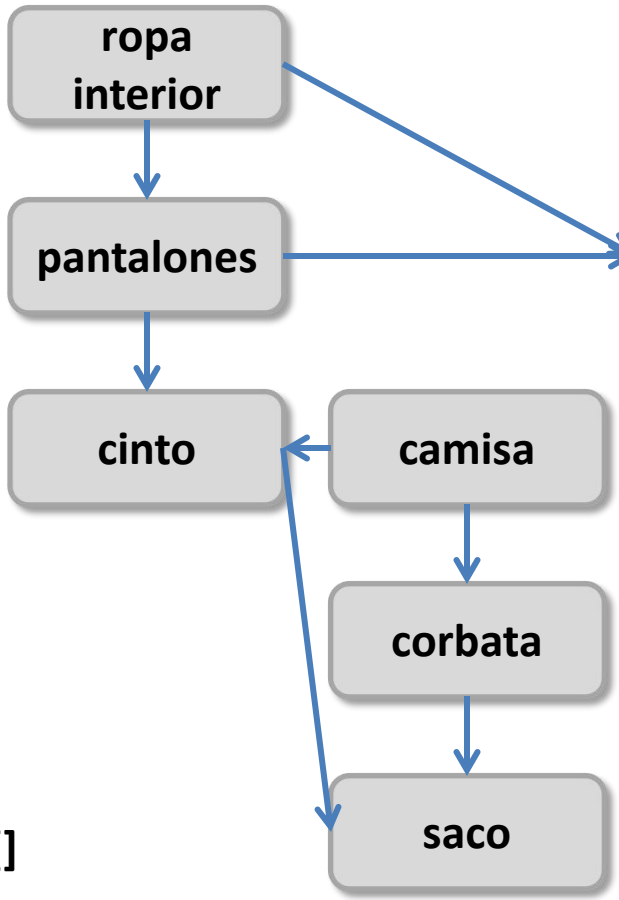
Los tiempos de descubrimiento y finalización (d[u] y f[u]) no son estrictamente necesarios para calcular un **orden topológico**, pero se pueden utilizar para otros problemas

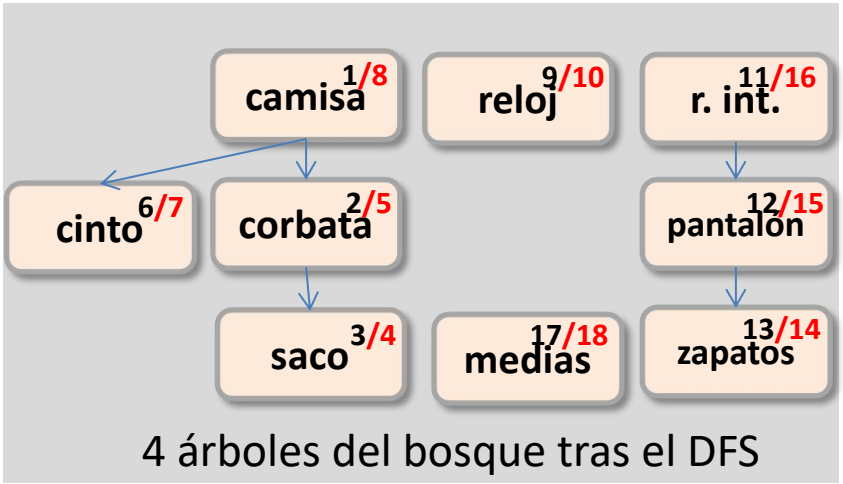
Orden Topológico

11	
10	
9	
8	medias
7	ropa int.
6	pantalón
5	zapatos
4	reloj
3	camisa
2	cinto
1	corbata
0	saco

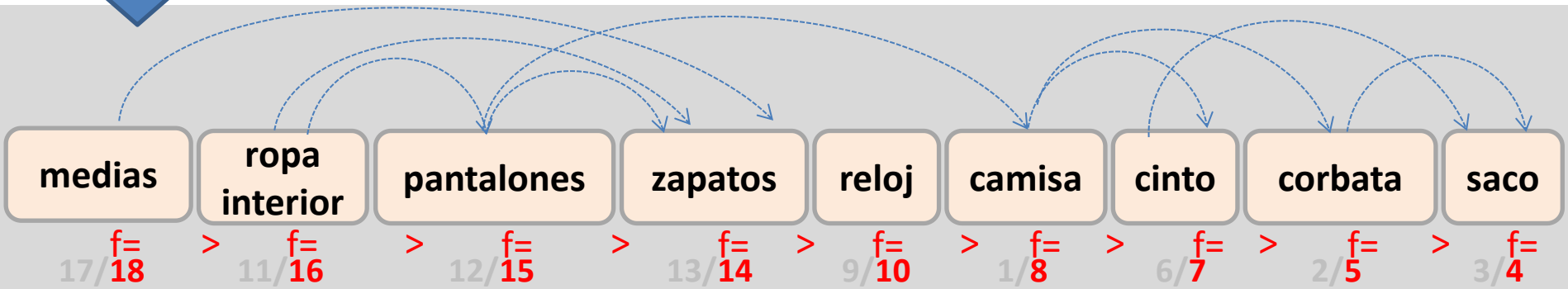
top

Stack[]



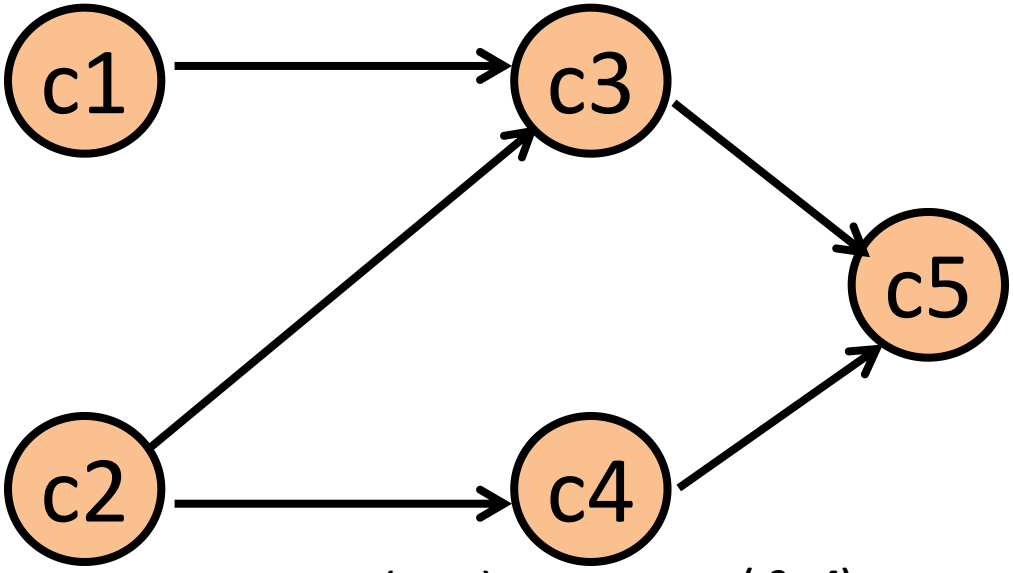


4 árboles del bosque tras el DFS

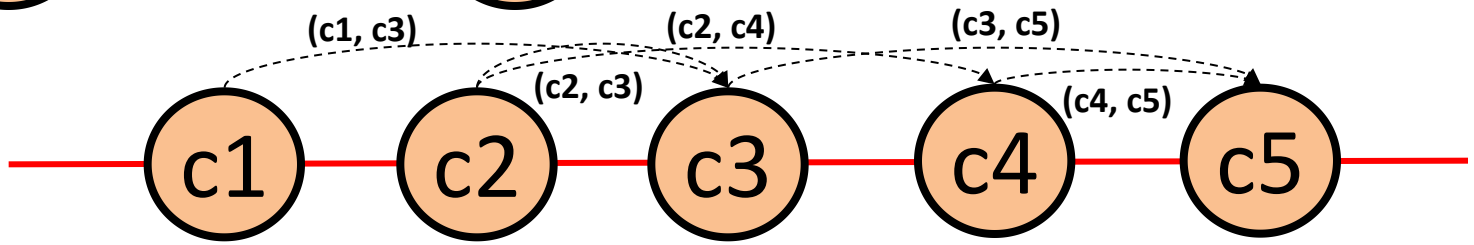


Orden topológico establecido a partir de los f[v]; v∈V, calculados por el DFS

DFS - Orden Topológico



A partir del algoritmo se obtiene **UN Orden Topológico**, lo cual no quiere decir que NO existan otros



f(c1) > f(c2) > f(c3) > f(c4) > f(c5)

Un orden topológico que se obtiene si el ***DFS*** se comienza por el vértice ***c2***

c1 c2 c4 c3 c5

Otro orden topológico que se obtiene si el ***DFS*** se comienza por el vértice **c2**

c2 c4 c1 c3 c5

Orden topológico que se obtiene si el ***DFS*** se comienza por el vértice ***c1***

Algoritmo DFS - Orden Topológico

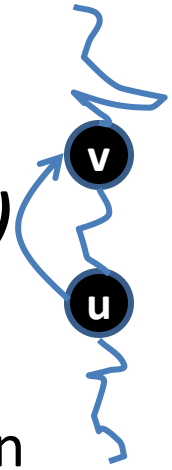
El siguiente **Lema** y el **Teorema** que posteriormente se enuncia, constituyen el marco teórico del algoritmo **DFS - Orden Topológico**

Lema: Un **grafo dirigido** *G* **es acíclico**, **si y solo si**, tras realizar una búsqueda "**primero en profundidad**" (*DFS*) sobre el mismo, no aparecen **arcos de retroceso**

Demostración: ***acíclico ⇒ NO arcos de retroceso***

⇒

Supongamos que tras el DFS aparece un arco de retroceso ***(u, v)***



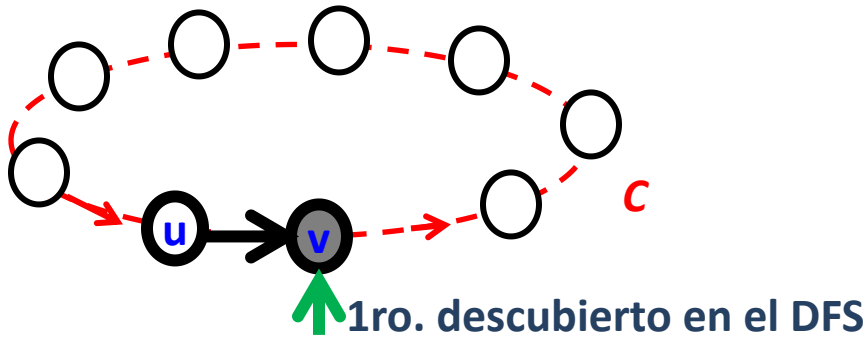
Entonces el vértice ***v*** es un ancestro del vértice ***u*** en el bosque primero en profundidad que se genera. Por tanto, ***G*** contiene un camino de ***v*** a ***u***, y el arco de retroceso completa un ciclo: **contradicción** pues ***G*** es acíclico

Algoritmo DFS - Orden Topológico

Demostración: ***acíclico ⇐ NO arcos de retroceso***

⇐

Supongamos que en G existe un ***ciclo c***. Sea ***v*** el primer vértice en ***c*** que fue descubierto. Sea ***(u, v)*** el arco precedente en c



En el instante **d[v]** los restantes vértices de ***c*** son blancos, por tanto, ***u*** es un descendiente de ***v*** en el ***árbol abarcador en profundidad***, por tanto, ***(u, v)*** es un ***arco de retroceso***: contradicción pues en ***G*** no hay ***arcos de retroceso***

Algoritmo DFS - Orden Topológico

Teorema: DFS-VISIT-OT(G, u) calcula UN orden topológico de un DAG G

Demostración: Supongamos que **DFS** se ejecuta sobre un **DAG $G = (V, E)$.** En el cual se calcula **f[x]**, $\forall x \in V$. Sería suficiente demostrar que para cualquier par de vértices u, $v \in V$, **si existe en G un arco (u, v), entonces tras el DFS, $f[v] < f[u]$**

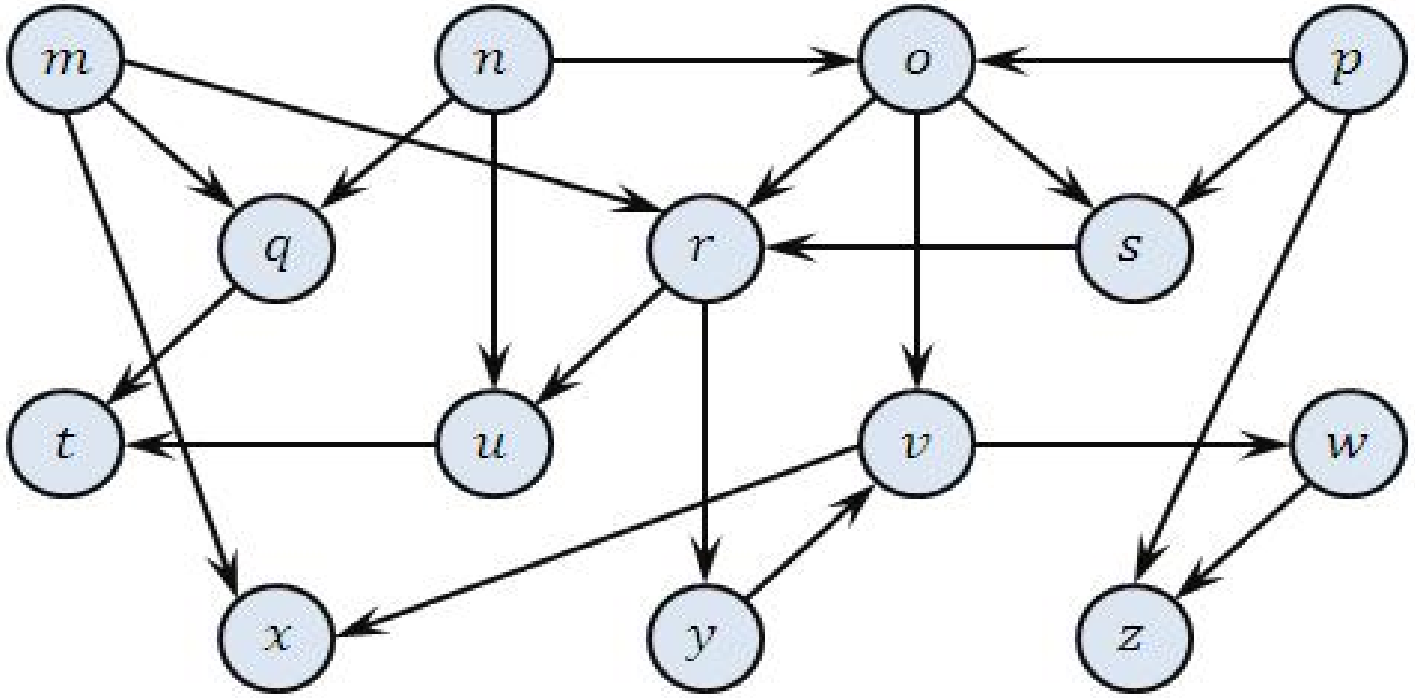
Consideremos cualquier arco **(u, v)** explorado durante el **DFS(G)**. Cuando esto sucede:
- v **no puede ser gris**, pues en tal caso, v sería un ancestro de u y el arco **(u, v)** sería un **arco de retroceso** (contradice el **Lema** pues **G** es un **DAG**). Por tanto, v tiene que ser **blanco o negro**:

- Si es **blanco**, se convierte en un descendiente de u, y por ello $f[v] < f[u]$

- Si v fuese **negro**, entonces, ya se terminó de analizar y por tanto, $f[v]$ ya se calculó. Como aun estamos explorando arcos desde u, entonces cuando el análisis finalice y se calcule $f[u]$, entonces se cumplira también $f[v] < f[u]$

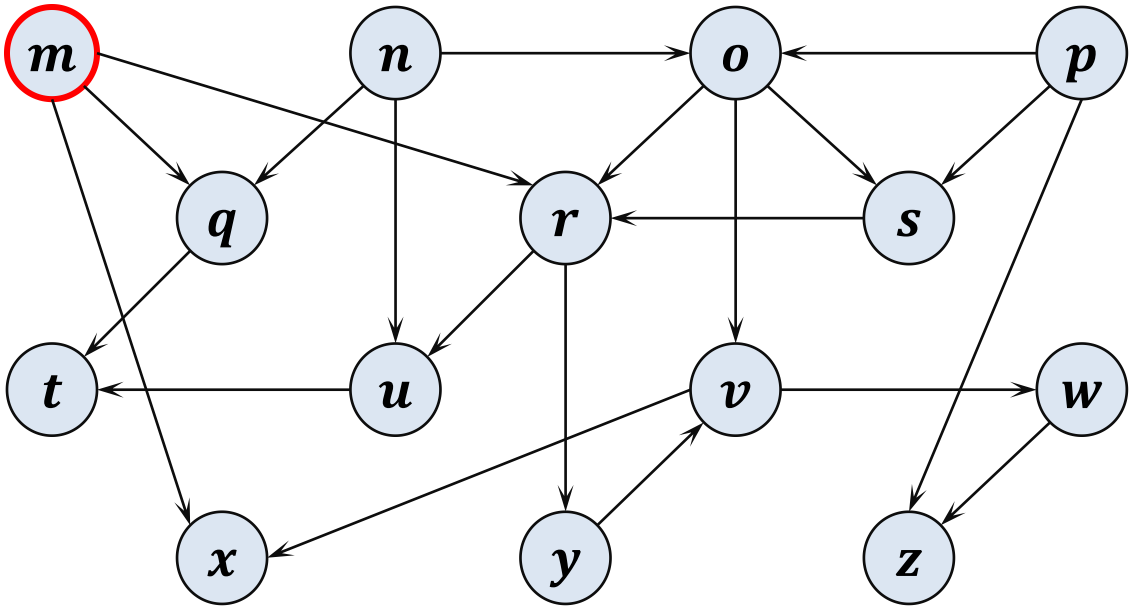
Por tanto, para cualquier arco **(u, v)** en el **DAG** se cumplirá $f[v] < f[u]$

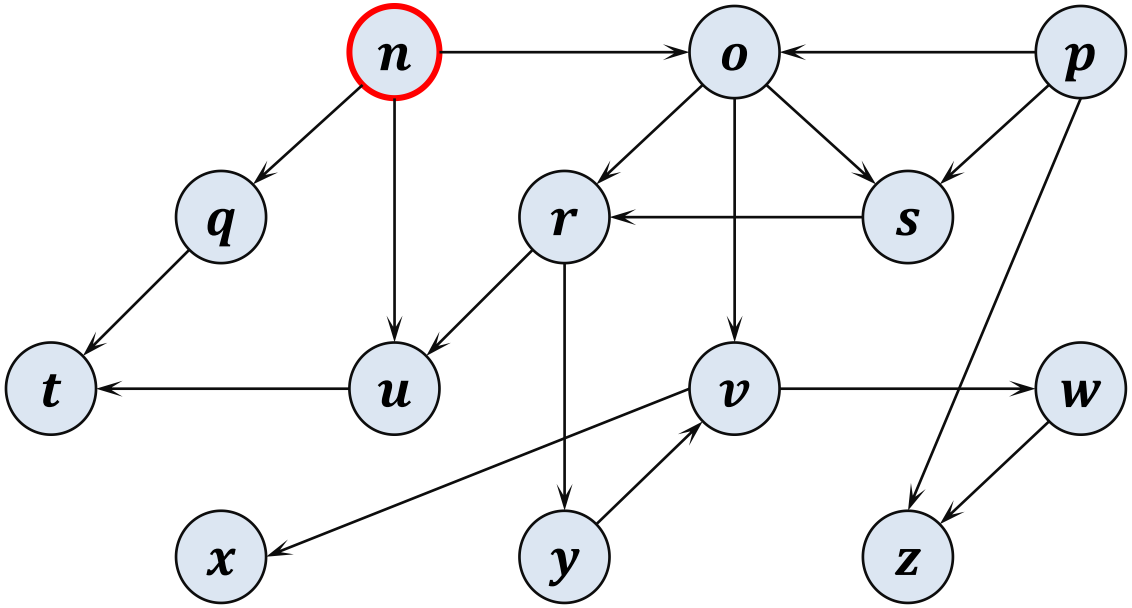
Otras variantes del Algoritmo - Orden Topológico



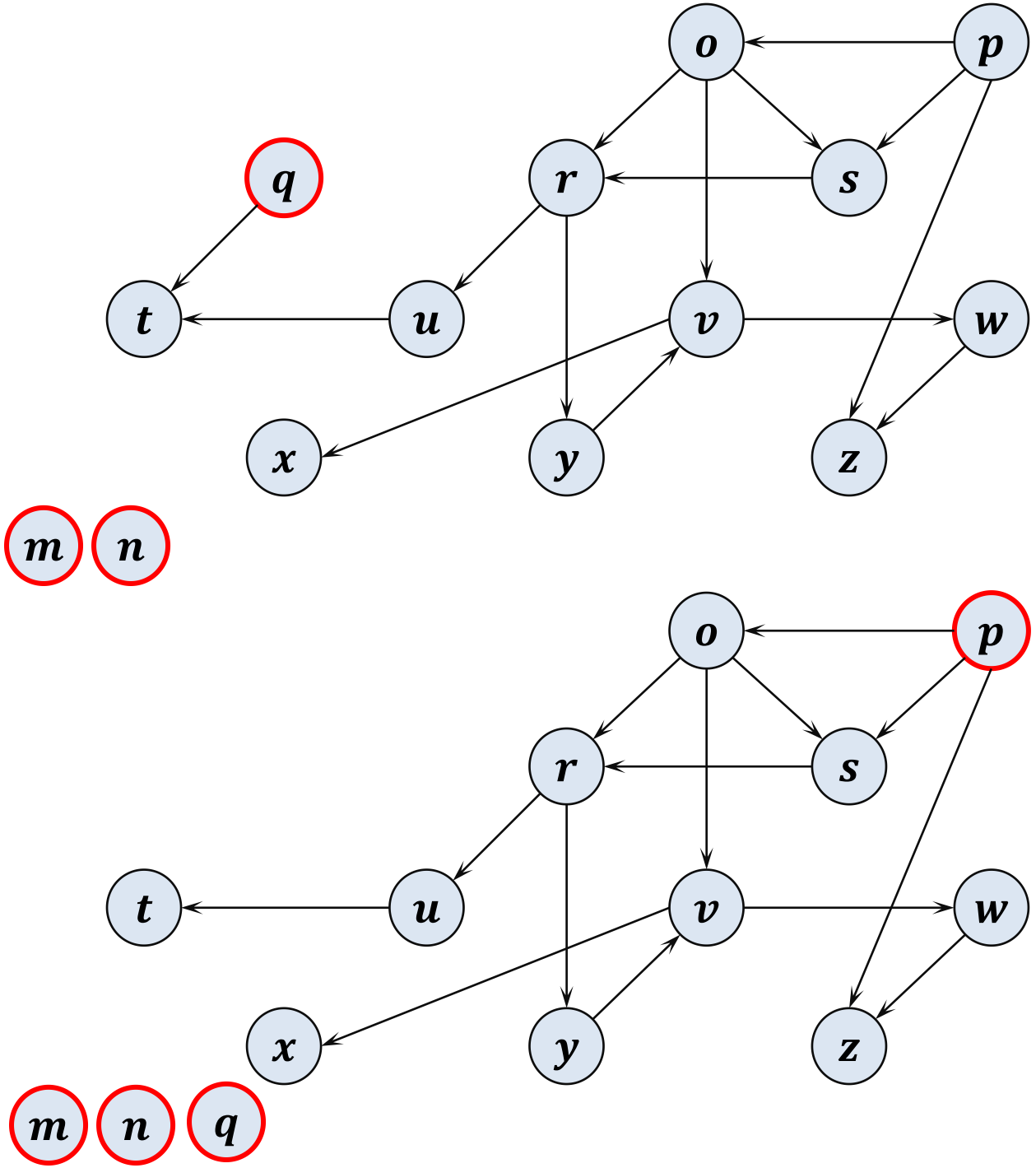
IDEA: Ir eliminando vértices de de ***G=(V, E)*** con *in-degree* = 0 consecutivamente, hasta que ***V*** y ***E*** sean $=\phi$

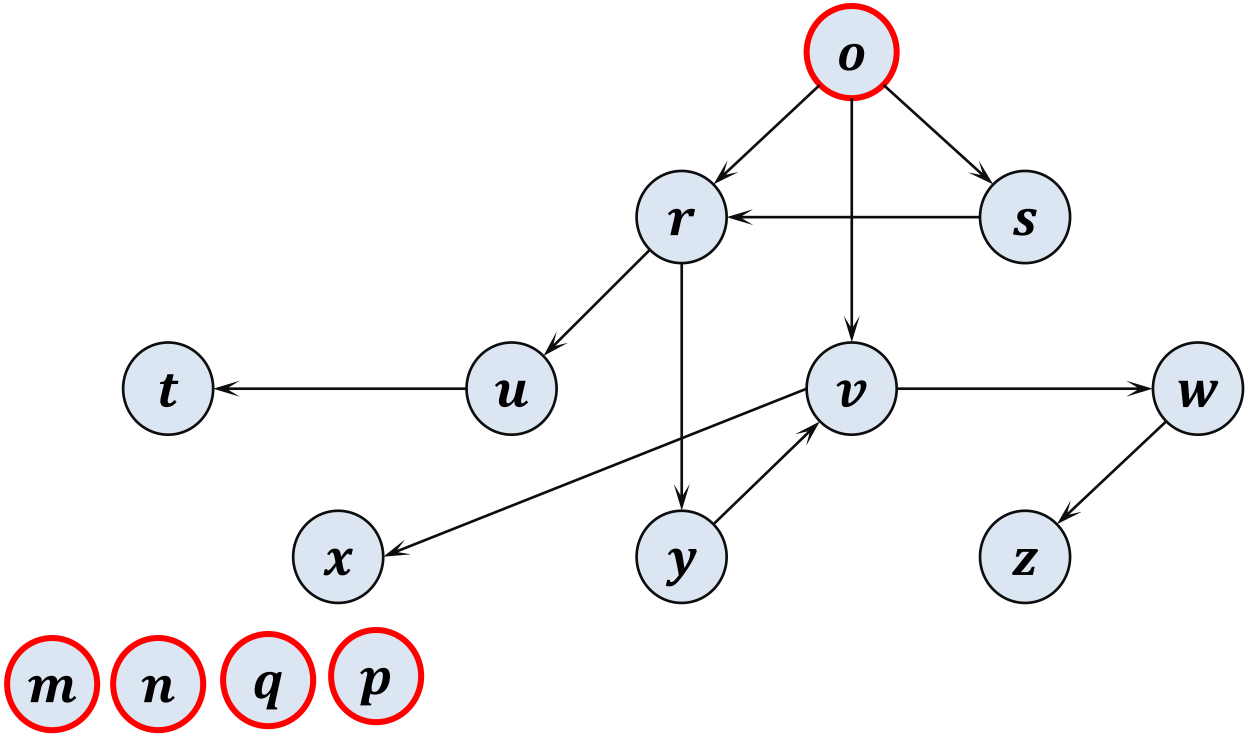
En cada iteración se elimina de G un vértice de *in-degree* = 0, y todos los arcos que salen de él. La secuencia de vértices que se van eliminando van conformando un orden topológico de G

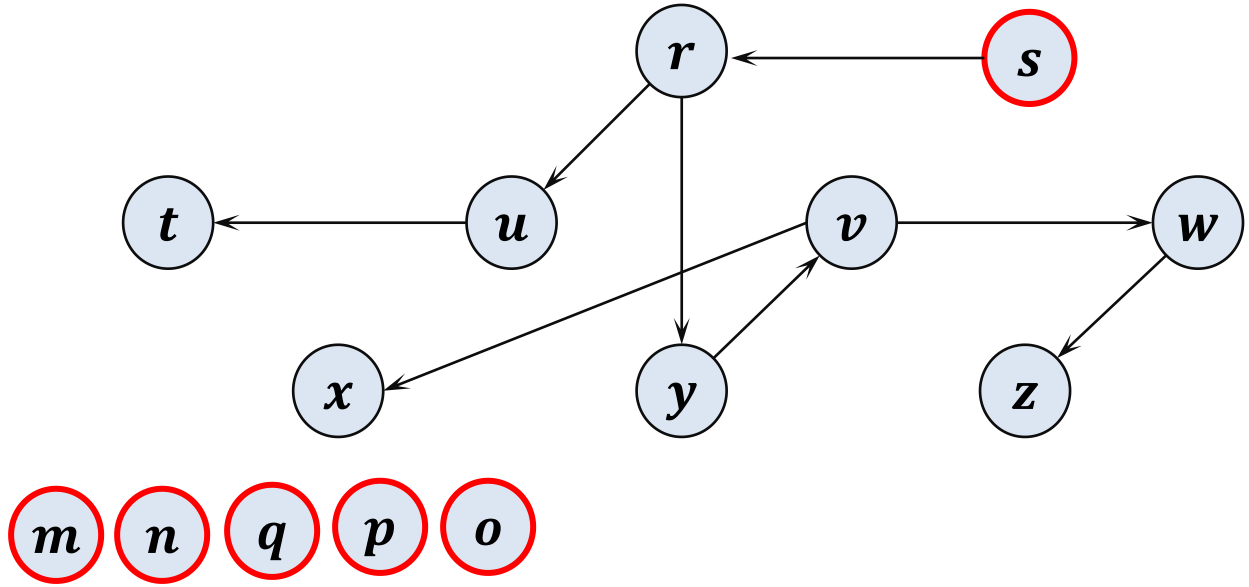

m
n
o
p
q
r
s
t
u
v
w
x
y
z

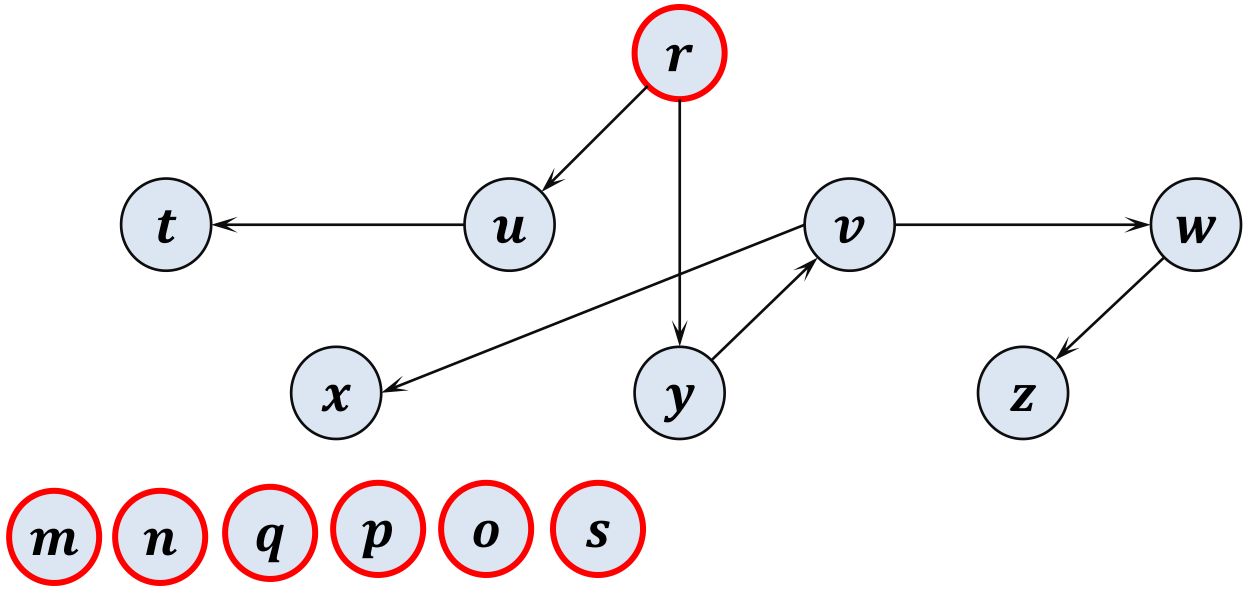

n
o
p
q
r
s
t
u
v
w
x
y
z

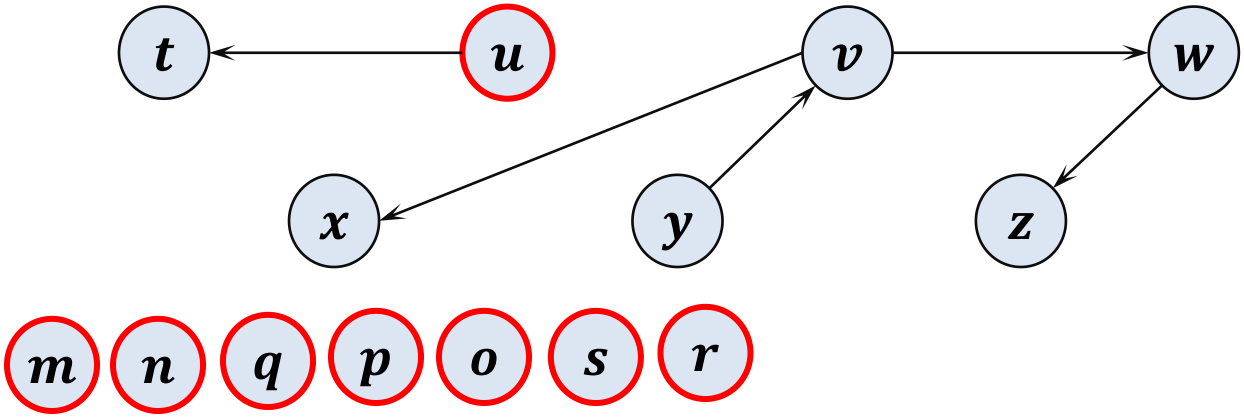

m

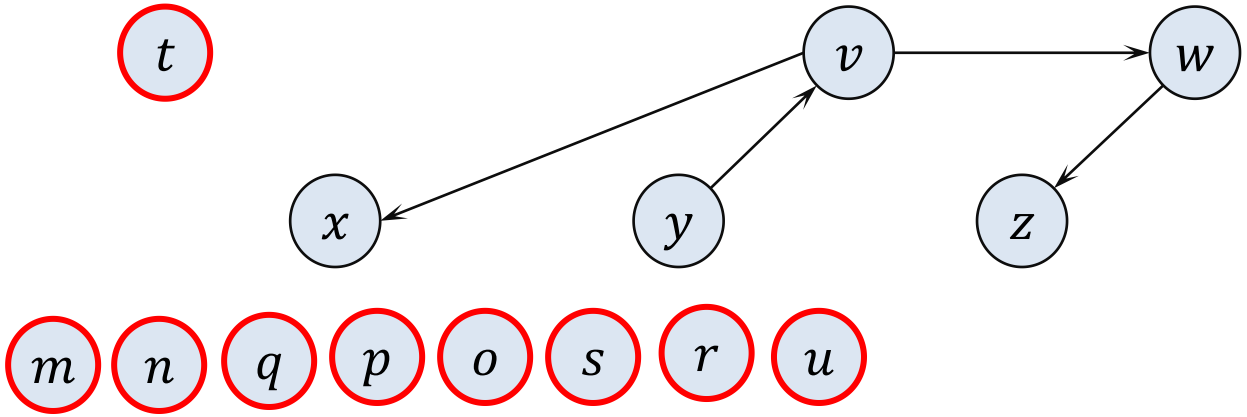

o
p
q
r
s
t
u
v
w
x
y
z
m
n
o
p
r
s
t
u
v
w
x
y
z
m
n
q

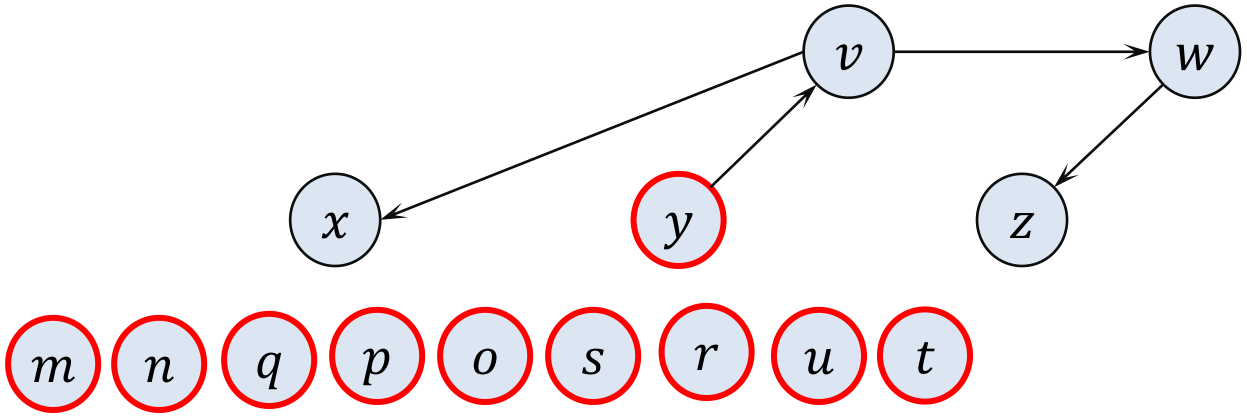

o
r
s
t
u
v
w
x
y
z
m
n
q
p

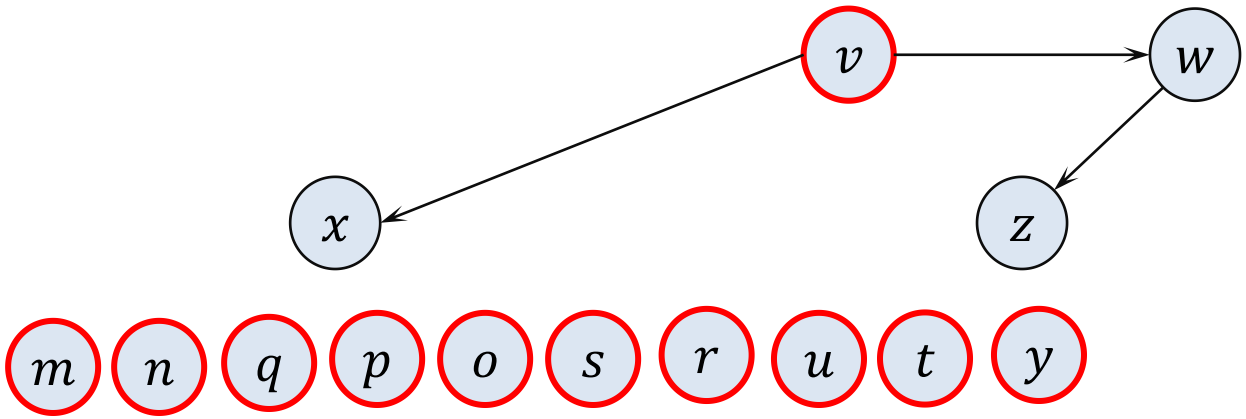

r
s
t
u
v
w
x
y
z
m
n
q
p
o

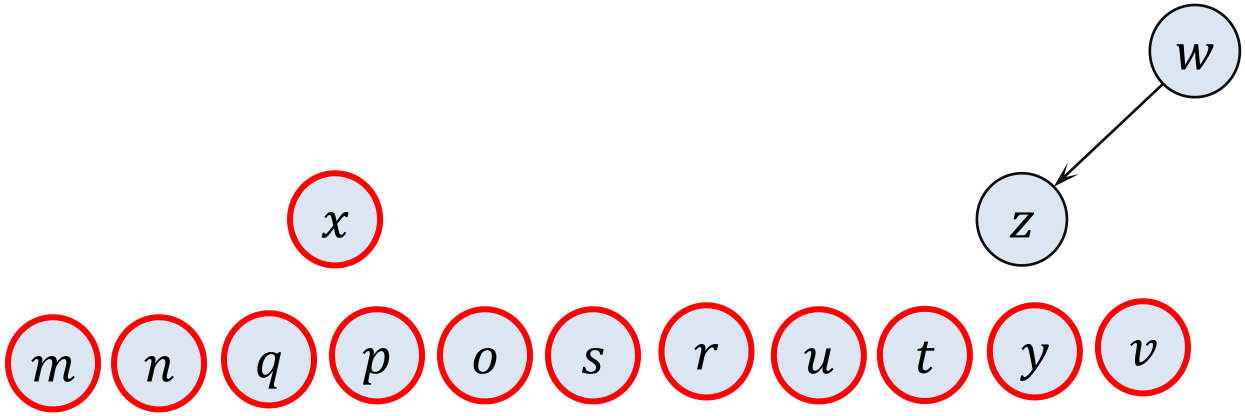

r
t
u
v
w
x
y
z
m
n
q
p
o
s

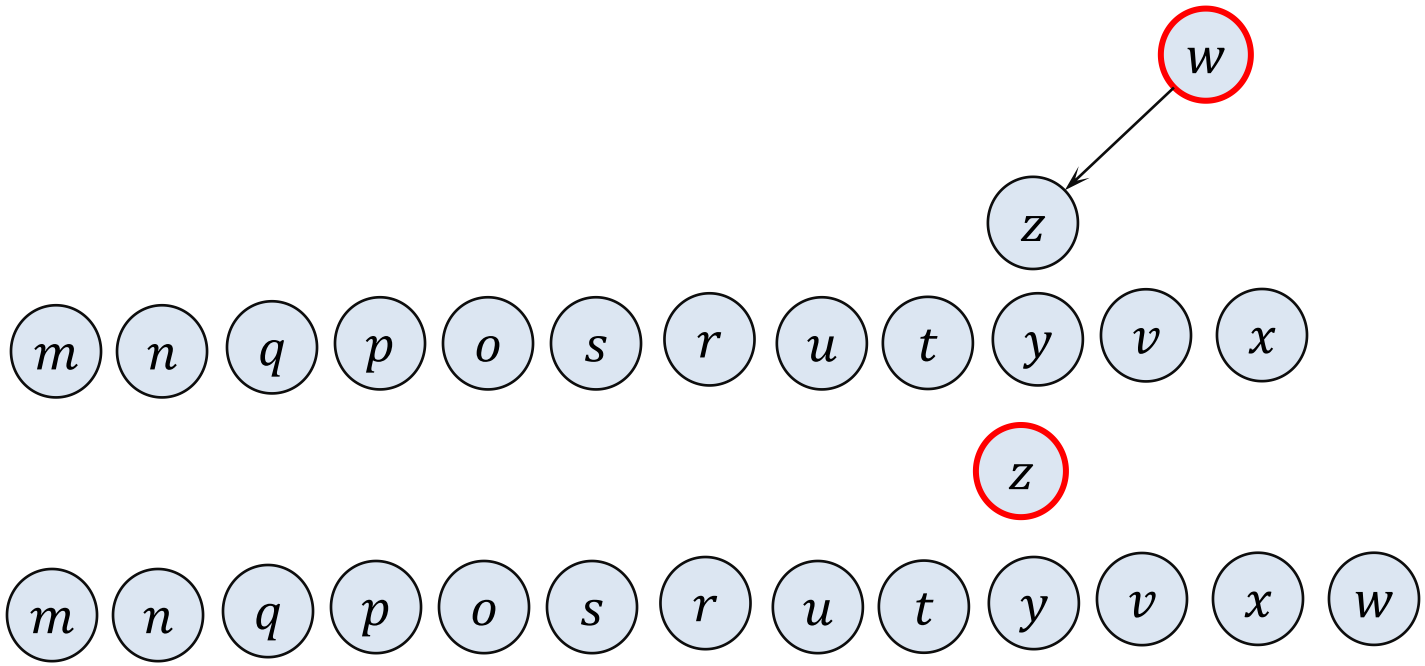

t
u
v
w
x
y
z
m
n
q
p
o
s
r


t
v
w
x
y
z
m
n
q
p
o
s
r
u

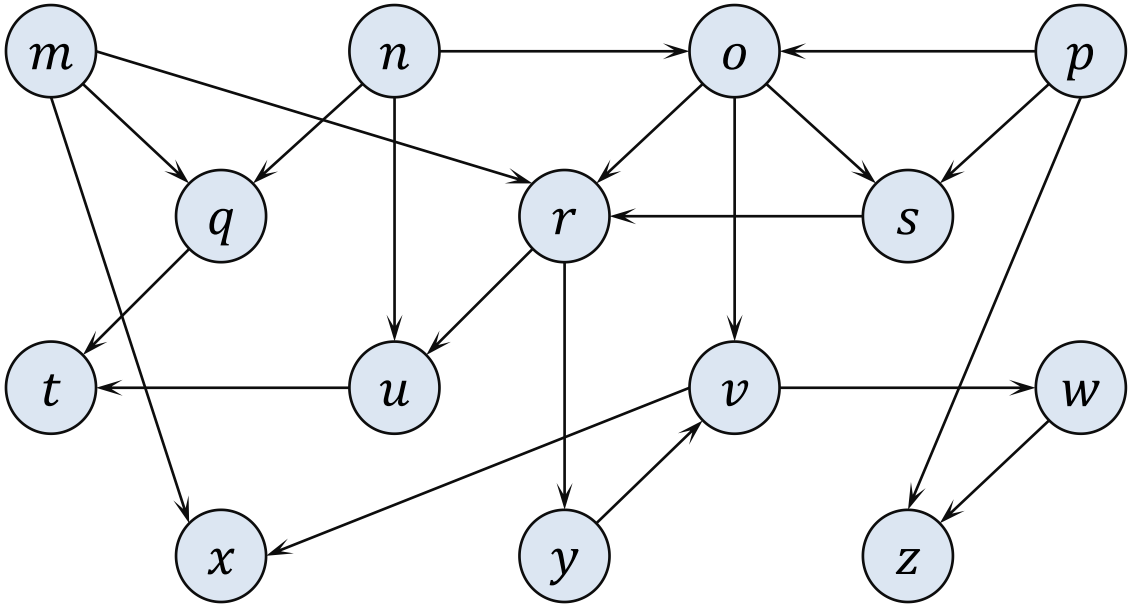

v
w
x
y
z
m
n
q
p
o
s
r
u
t


v
w
x
z
m
n
q
p
o
s
r
u
t
y


w
x
z
m
n
q
p
o
s
r
u
t
y
v



Un **orden topológico** para G: ***m*** ***n*** ***q*** ***p*** ***o*** ***s*** ***r*** ***u*** ***t*** ***y*** ***v*** ***x*** ***w*** ***z***



Otros Algoritmos relacionados - Orden Topológico

¿ Cómo, dada una ordenación lineal de los vértices de G=(V, E), poder determinar si es un *Orden Topológico* para G ?

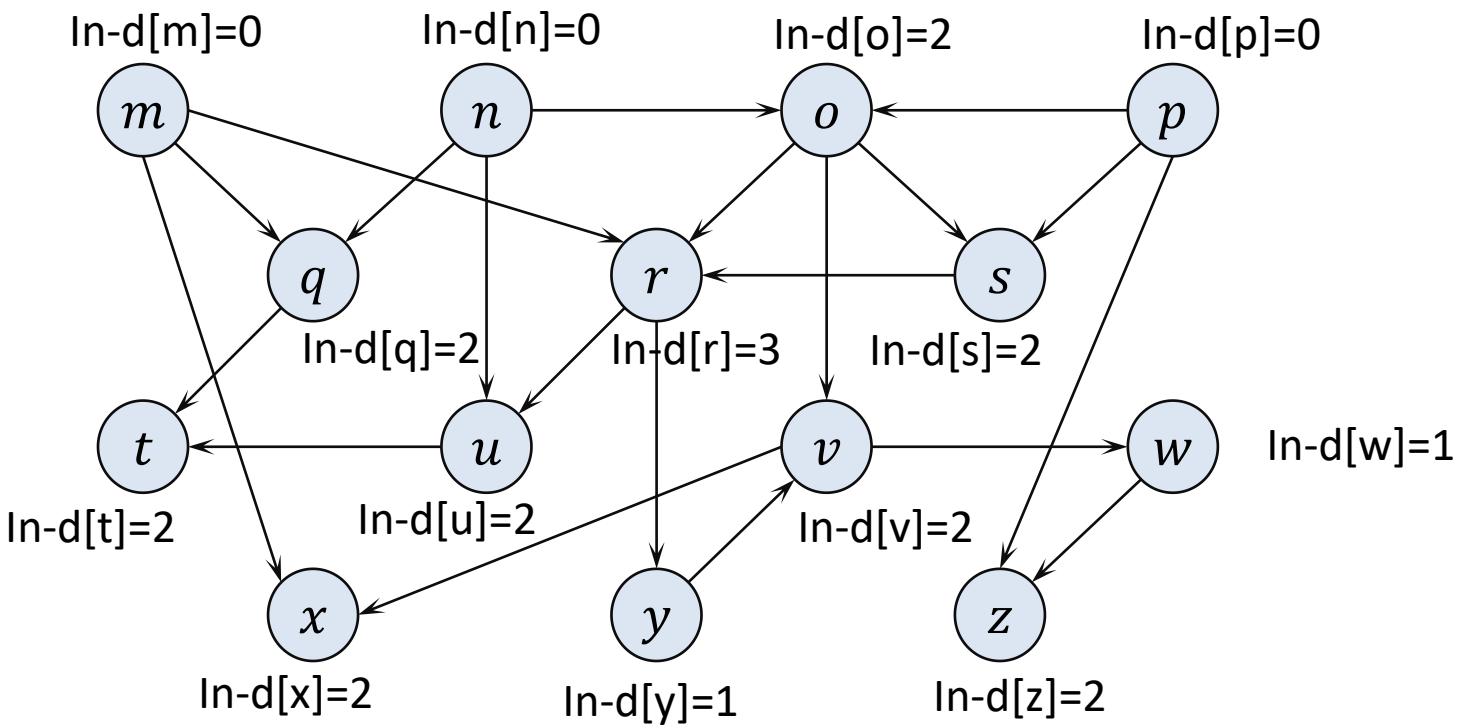
E/ para el algoritmo:

- G=(V, E)
- ordenación lineal de los vértices de G

Salida: *true* o *false*

IDEA: Ir inspeccionando los vértices en el orden lineal de entrada y tomar decisiones en función del ***in-degree*** de ellos

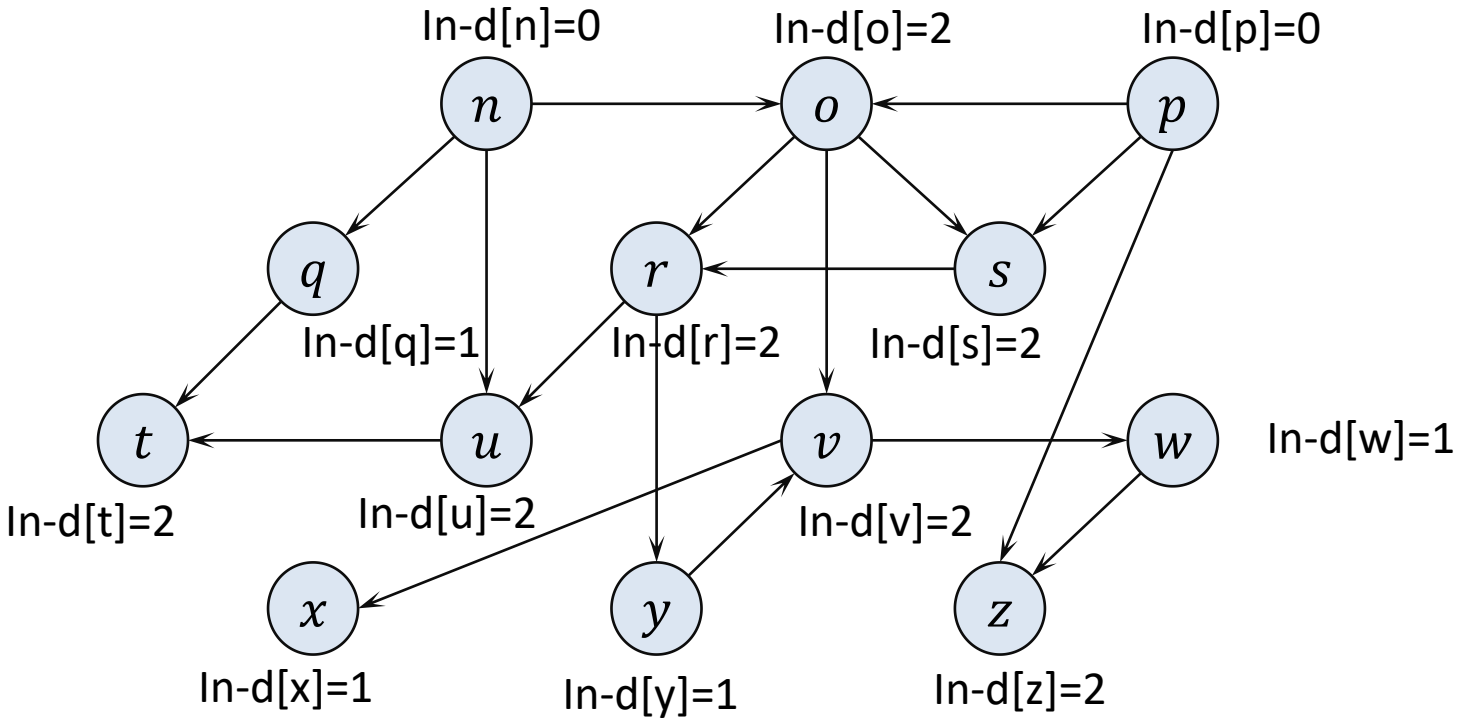
Un **orden topológico** para G: **m n q p o s r u t y v x w z**



m In-degree[m]≠0 entonces *false*

In-degree[m]=0 entonces: *-eliminar m del grafo*
-decrementar en 1 el in-degree de todos los adyacentes de m

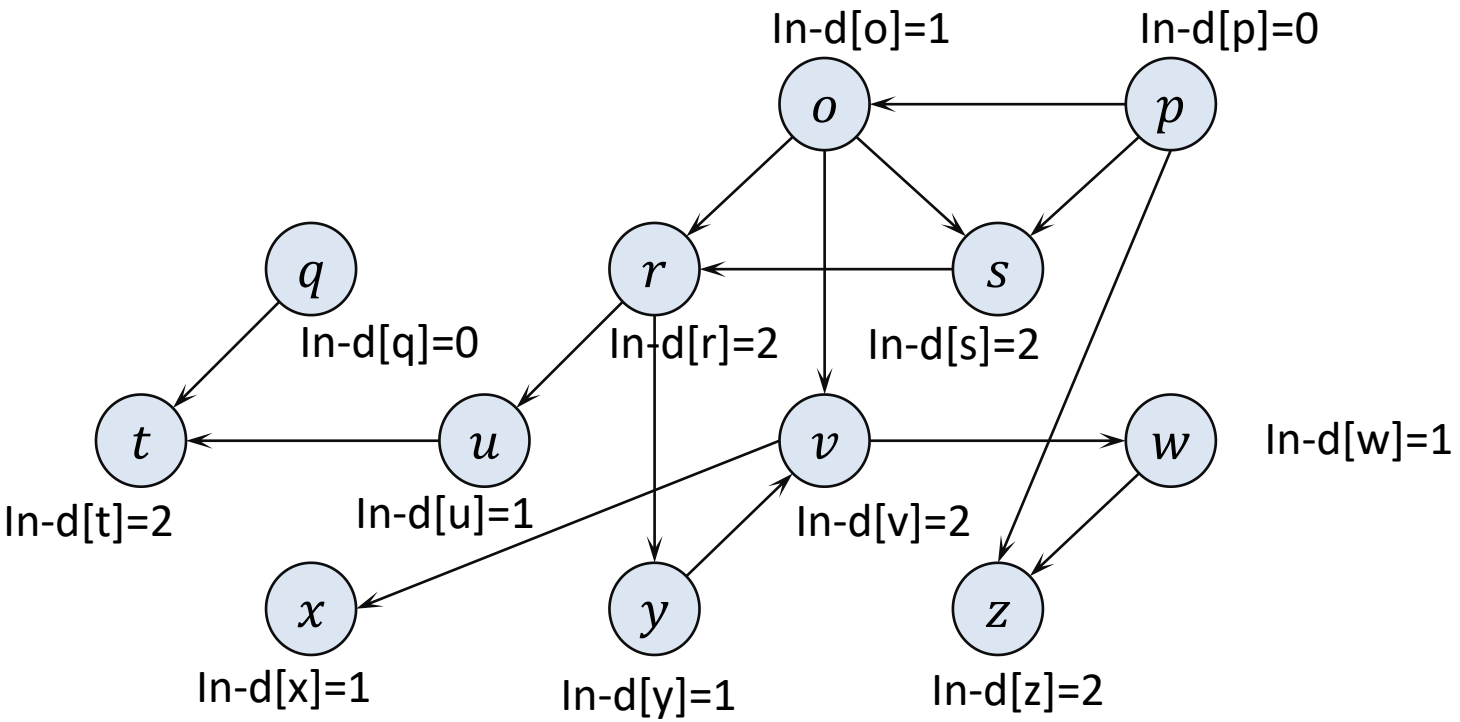
Un **orden topológico** para G: *n* *q* *p* *o* *s* *r* *u* *t* *y* *v* *x* *w* *z*



n In-degree[n]≠0
entonces *false*

In-degree[n]=0 entonces: -*eliminar n del grafo*
-decrementar en 1 el in-degree de todos los adyacentes de n

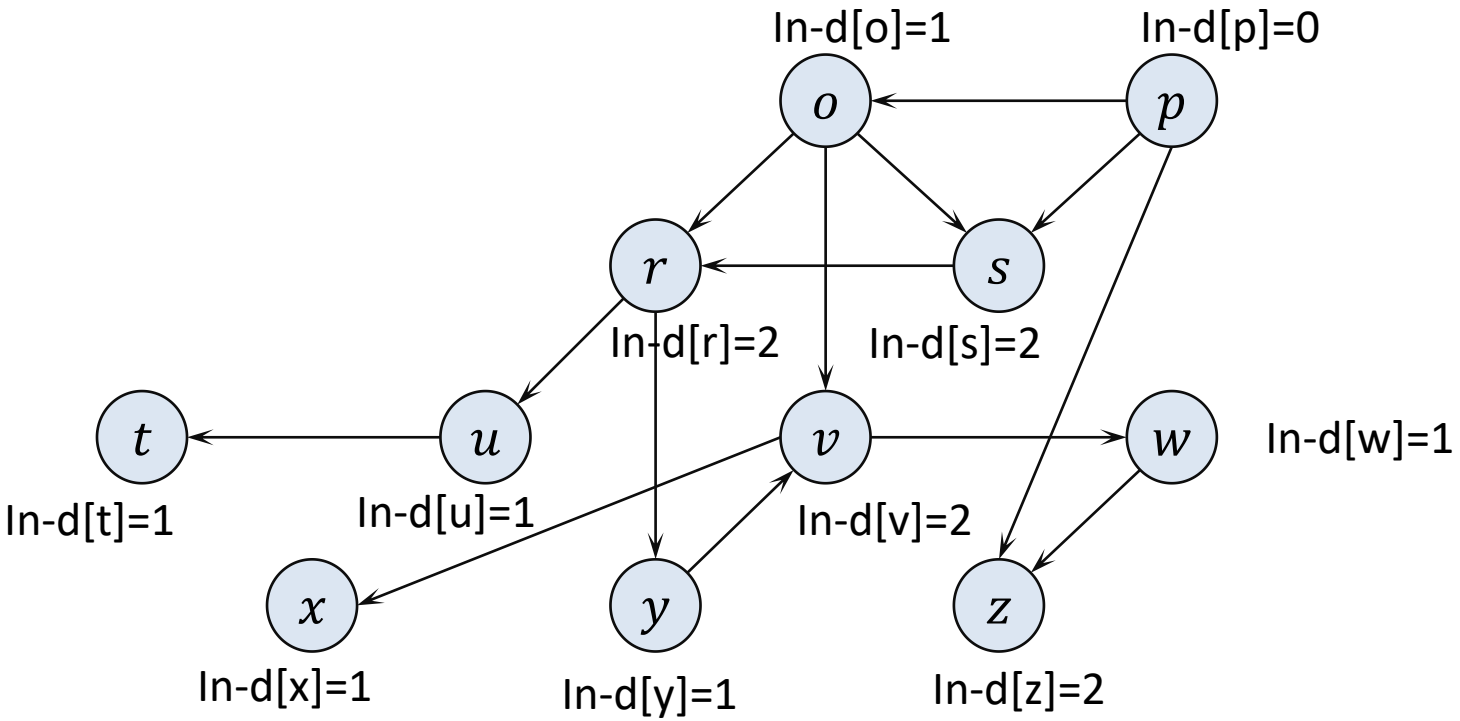
Un **orden topológico** para G: *q* *p* *o* *s* *r* *u* *t* *y* *v* *x* *w* *z*



q In-degree[q]≠0
entonces *false*

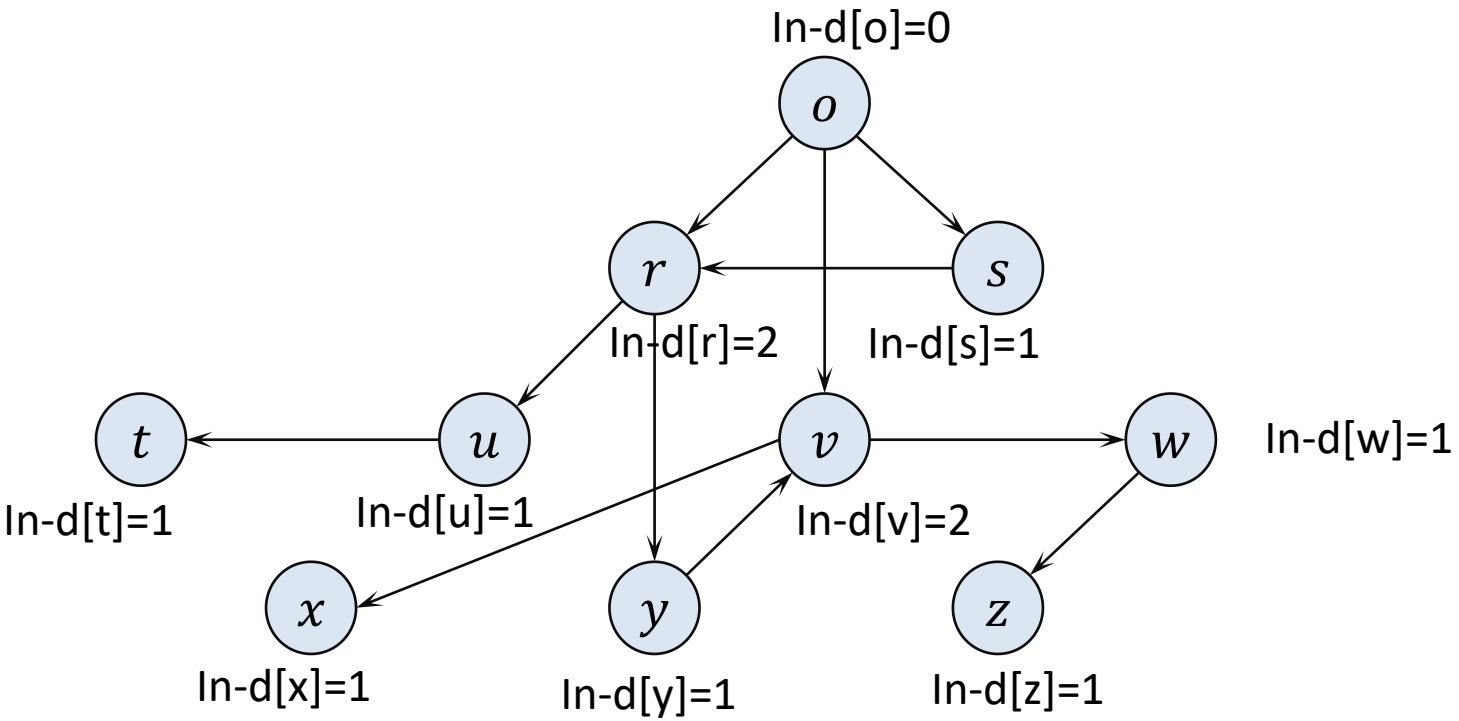
In-degree[q]=0 entonces: -*eliminar q del grafo*
-decrementar en 1 el in-degree de todos los adyacentes de q

Un **orden topológico** para G: **p o s r u t y v x w z**



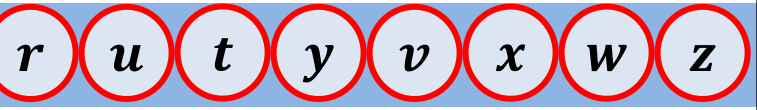
p In-degree[p]≠0
entonces *false*
In-degree[p]=0 entonces: -*eliminar p del grafo*
-decrementar en 1 el in-degree de todos los adyacentes de p

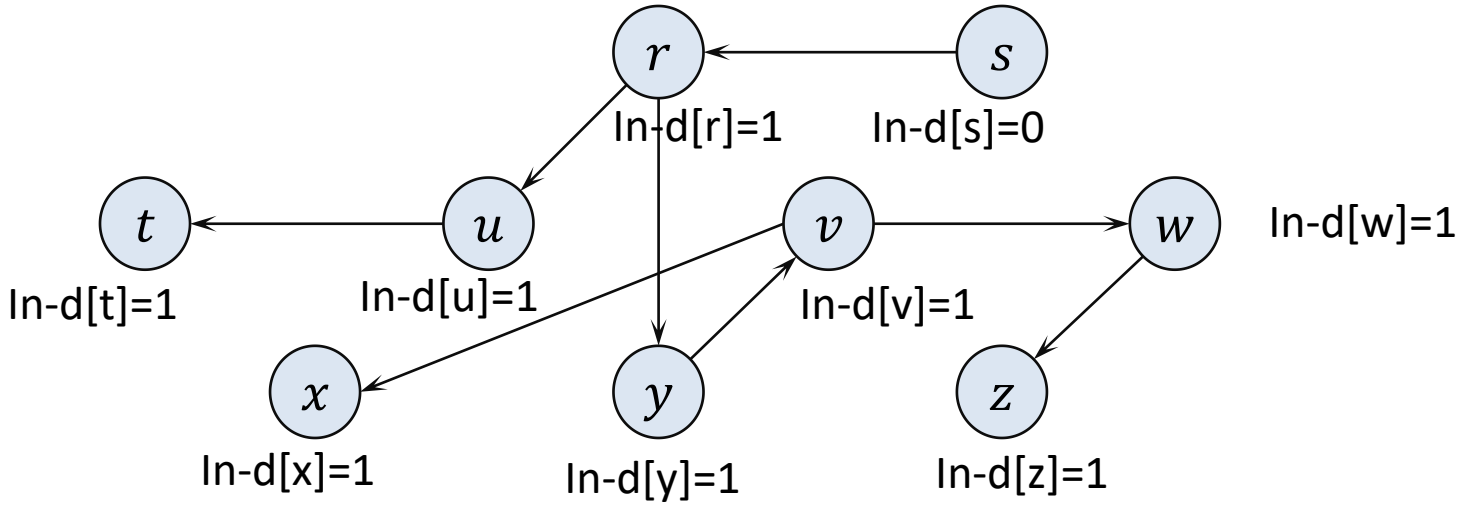
Un **orden topológico** para G: *o* *s* *r* *u* *t* *y* *v* *x* *w* *z*



o In-degree[o]≠0
entonces *false*
In-degree[o]=0 entonces: -*eliminar o del grafo*
-decrementar en 1 el in-degree de todos los adyacentes de o

Un **orden topológico** para G:

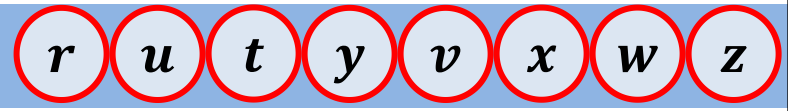


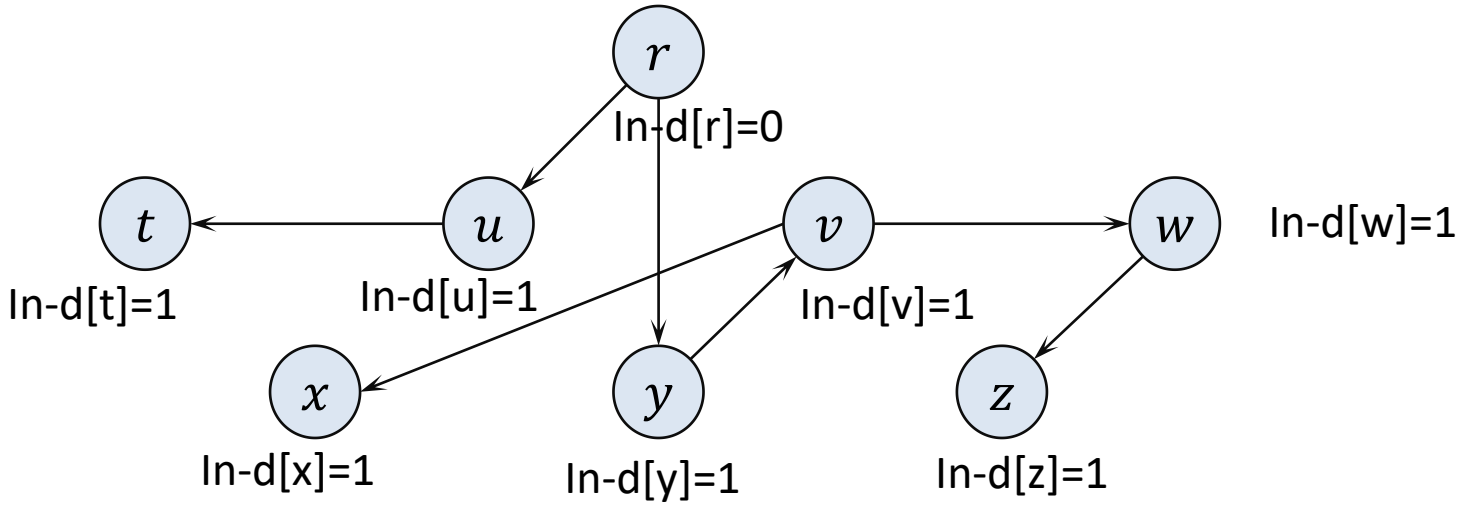


In-degree[s]≠0
entonces *false*

In-degree[s]=0 entonces: -*eliminar s del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de s*

Un **orden topológico** para G:

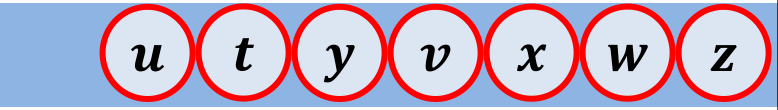


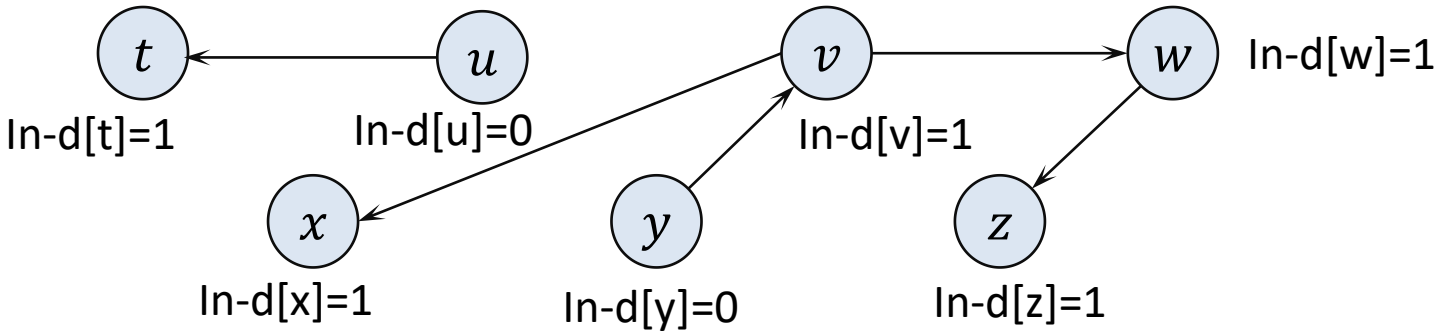


In-degree[r]≠0
entonces *false*

In-degree[r]=0 entonces: -*eliminar r del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de r*

Un **orden topológico** para G:

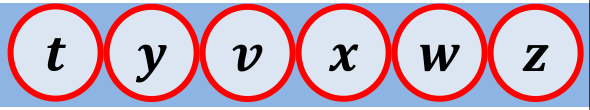


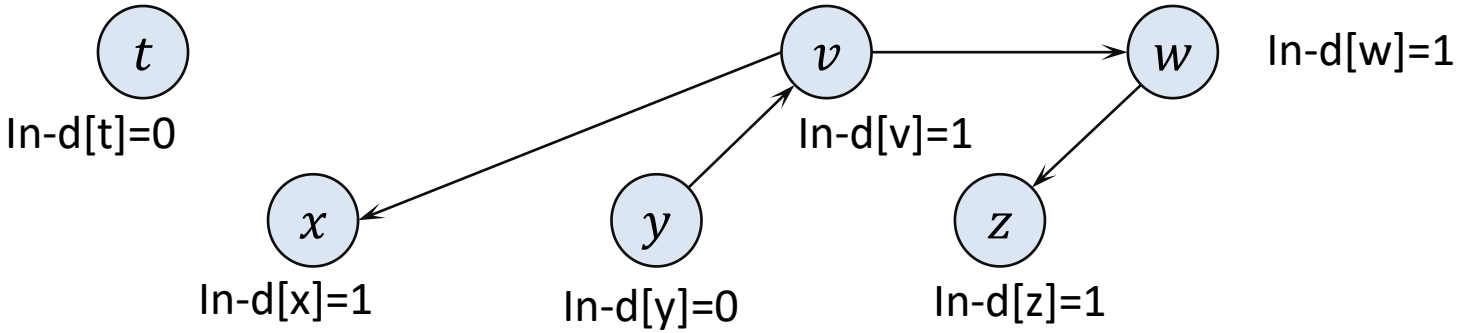


In-degree[u]≠0
entonces *false*

In-degree[u]=0 entonces: -*eliminar u del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de u*

Un **orden topológico** para G:

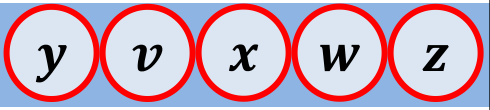


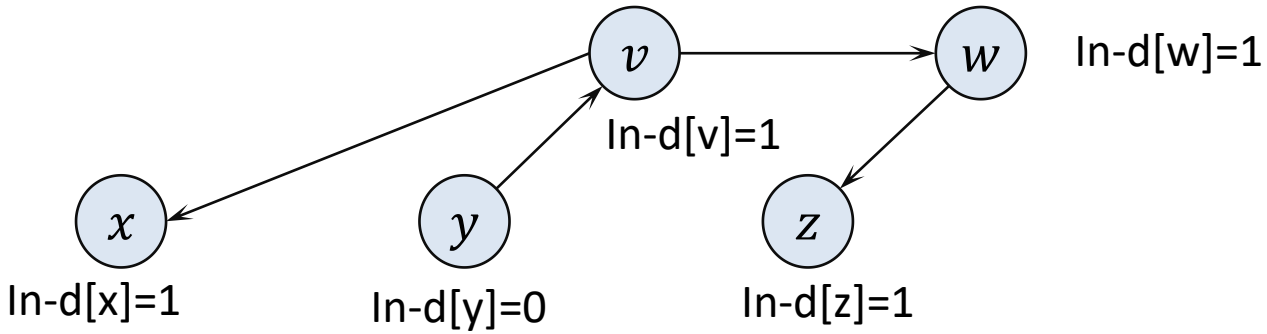


In-degree[t]≠0
entonces *false*

In-degree[t]=0 entonces: -*eliminar t del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de t*

Un **orden topológico** para G:

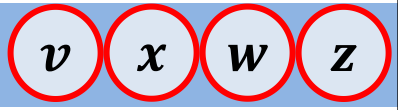


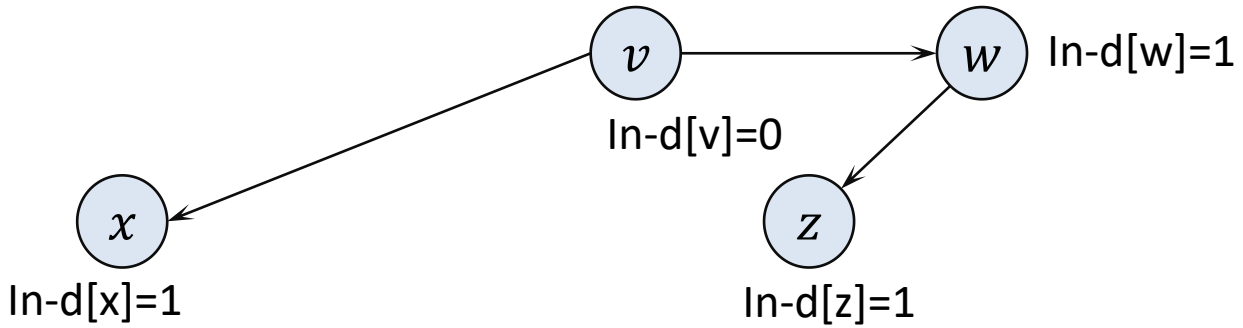


In-degree[y]≠0
entonces *false*

In-degree[y]=0 entonces: -*eliminar y del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de y*

Un **orden topológico** para G:

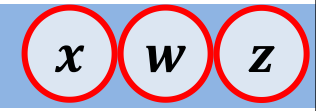


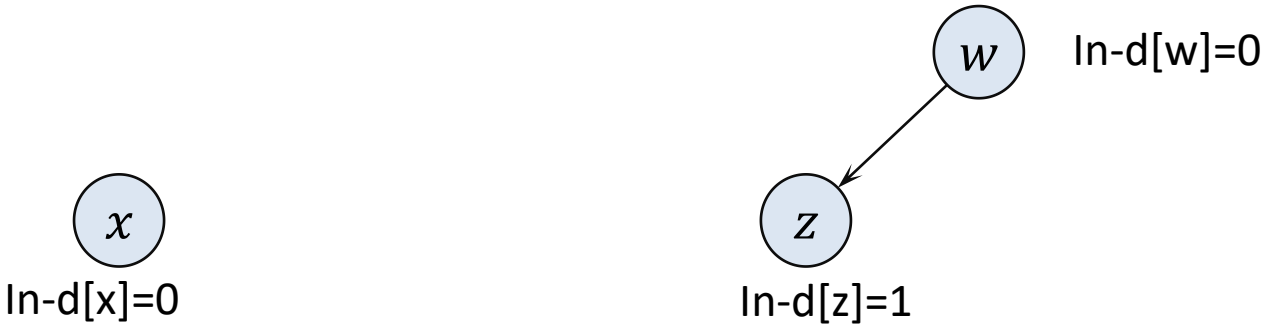


In-degree[v]≠0
entonces *false*

In-degree[v]=0 entonces: -*eliminar v del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de v*

Un **orden topológico** para G:

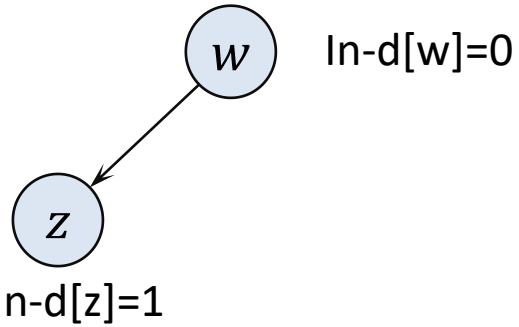




In-degree[y]≠0
entonces *false*

In-degree[x]=0 entonces: -*eliminar x del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de x*

Un **orden topológico** para G: *w* *z*

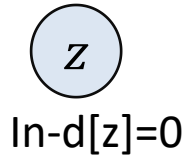


w In-degree[w]≠0
entonces *false*

In-degree[w]=0 entonces: -*eliminar w del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de w*

Un **orden topológico** para G: *z*

z In-degree[z]≠0
entonces *false*



In-degree[z]=0 entonces: -*eliminar z del grafo*
-*decrementar en 1 el in-degree de todos los adyacentes de z*

***TRUE*: Si era un orden topológico**

La correctitud de este algoritmo se demuestra a partir de un Teorema que dice:

Existe un órden topológico en G $\Leftrightarrow$ G es un DAG

el cual será demostrado en Clase Práctica

Otras Algoritmos relacionados - Orden Topológico

¿ Cómo, dada una ordenación lineal de los vértices de G=(V, E), poder determinar si es un **orden topológico** para G ?

E/ para el algoritmo: - G=(V, E) - ordenación lineal de los vértices de G	**S/**: *true* o *false*

IDEA: Ir inspeccionando los vértices en el **orden lineal** de entrada.

- El primero tiene que tener in-degree=0, si no, return ***false***
- Eliminar el vértice del Grafo y a todos los vértices adyacentes a él, disminuir en 1 su in-degree
- El siguiente vértice en el orden lineal debe tener in-degree=0, si no, return ***false***

Repetir el proceso hasta haber inspeccionado todo el orden lineal, si se llegó hasta el último vértice, return ***true***