

---

# Ayudantía Grafos y DFS

— Dafne Arriagada, Gustavo Salinas,  
Paula Grune, Agustín Gutiérrez —

---

<https://github.com/IIC2133-PUC/2024-1>

# MATERIAL DE APOYO

1. Cheatsheet C (notion resumen)
2. Ejercicios de práctica C
3. Cápsulas de semestres pasados

## Dónde encuentro esto?

Links en ReadMe carpeta "Ayudantías" del repo



\*Sonrisa coqueta\*

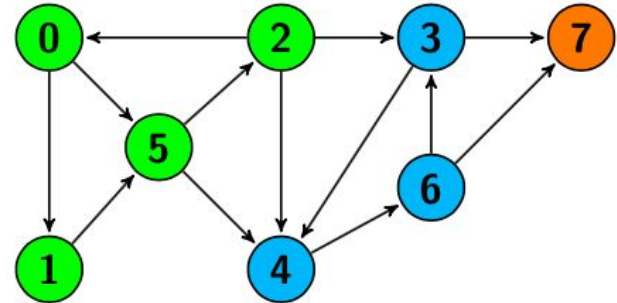
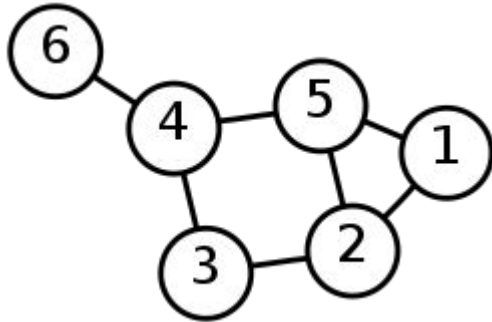
# Contenidos

1. Repaso grafos, caminos y ciclos
2. Algoritmos DFS (Depth First Search)
3. Orden Topológico
4. Componentes fuertemente conexas (CFC)
5. Algoritmo de Kosaraju y su relación con TopSort

# ¿Qué es un Grafo?

# Grafo $G = (V, E)$

- **Estructura de datos** compuesta por **nodos (V)** los cuales están unidos por **aristas (E)**. Estas aristas pueden estar dirigidas o no. Un grafo puede ser dirigido (pares ordenados) o no dirigido (conjuntos).

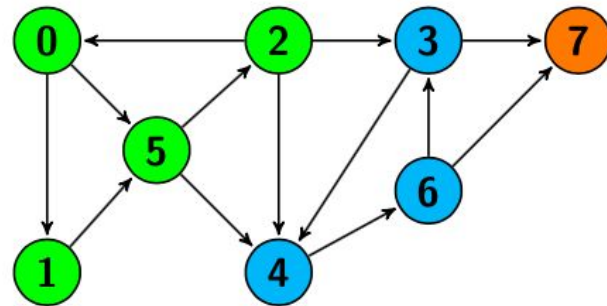


# Grafo $G = (V, E)$

- **Estructura de datos** compuesta por **nodos** los cuales están unidos por **aristas**. Estas aristas pueden estar dirigidas o no. Un grafo puede ser dirigido (pares ordenados) o no dirigido (conjuntos).

## ¿Cómo los podemos representar?

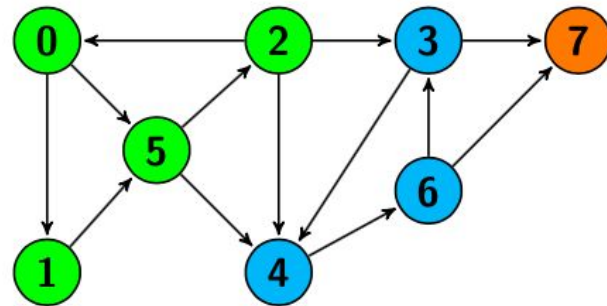
Según sea el caso, podemos representarlos a través de **listas de adyacencia** (para grafos poco densos) o como una **matriz de adyacencia** (para grafos muy densos)  
.... ¿Por qué? ¿Qué queremos decir con “denso”?



# Caminos y ciclos:

- Un **camino  $\pi$**  de largo  $n$ , es una secuencia de nodos  $v_0, \dots, v_n$  tal que  $(v_i, v_{i+1})$  (**par!**) pertenece al conjunto de aristas  $E$  para todo  $i < n$ .

¿Cuándo un camino se convierte en un ciclo?



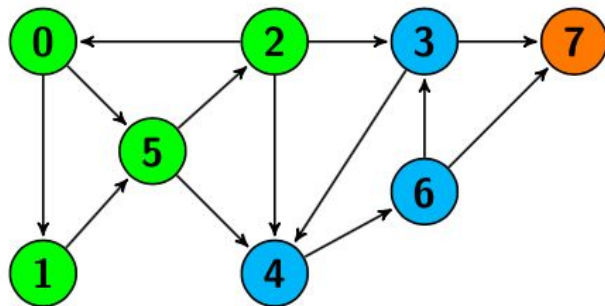
# Camino y ciclos:

- Un **camino  $\pi$**  de largo  $n$ , es una secuencia de nodos  $v_0, \dots, v_n$  tal que  $(v_i, v_{i+1})$  (**par!**) pertenece al conjunto de aristas  $E$  para todo  $0 \leq i \leq n-1$ .

¿Cuándo un camino se convierte en un ciclo?

Si  $v_0 = v_n$  cuando  $n > 0$  (al menos dos vértices)

¿Por qué es importante detectar estos ciclos?





# Caminos y ciclos:

- Un **camino  $\pi$**  de largo  $n$ , es una secuencia de nodos  $v_0, \dots, v_n$  tal que  $(v_i, v_{i+1})$  (**par!**) pertenece al conjunto de aristas  $E$  para todo  $i < n$ .

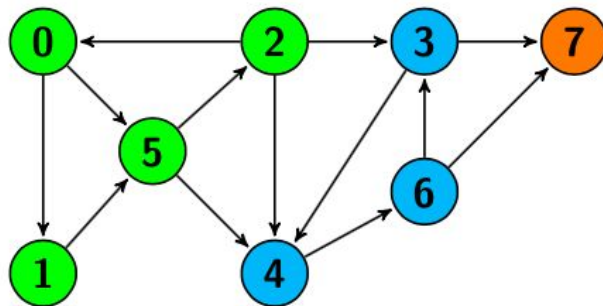
## ¿Cuándo un camino se convierte en un ciclo?

Si  $v_0 = v_n$  cuando  $n > 0$  (al menos dos vértices)

## ¿Por qué es importante detectar estos ciclos?

De no hacerlo podemos:

1. Quedarnos atrapados en loops infinitos!
2. No ser capaces de ordenar un conjunto de tareas a ejecutar.
3. No detectar proyectos inviables (Referencia circular en proyectos con requisitos)

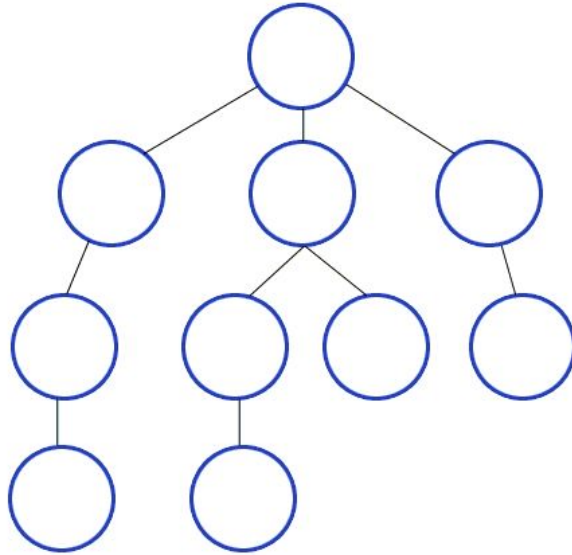


# DFS - Deep First Search

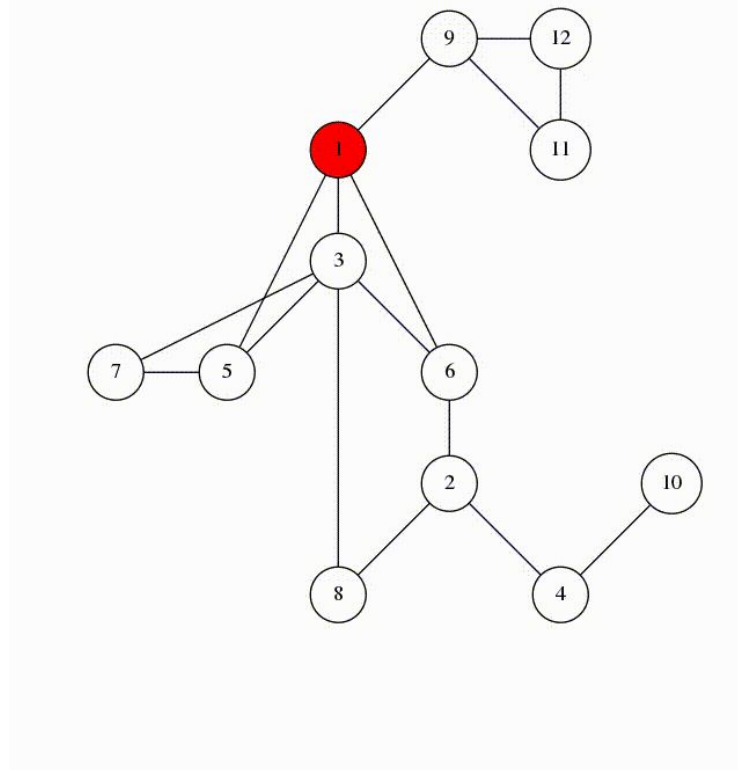
- **Depth First Search o Búsqueda en profundidad.**
- **Grafo no dirigido o no dirigido**
- Este algoritmo nos ayuda a recorrer un grafo de forma ordenada. Su funcionamiento es similar al **backtracking**. Recorre un camino hasta el fondo y luego sigue explorando otros caminos.

Notar que si bien vamos a presentar un algoritmo DFS, DFS es en sí mismo una **estrategia** (estrategia de búsqueda en profundidad) en la cual se pueden basar los algoritmos.

# Ejemplo visual



# Ejemplo visual



# DFS, estrategia iterativa

```
dfs(graph G, node start)
    stack s
    s.push(start)
    label start as discovered
    while not s.empty()
        node u = s.pop()
        for v in G.adjacent[u]
            if v is not discovered
                s.push(v)
                label v as discovered
```

# DFS, estrategia recursiva

Recordemos el código de colores

- BLANCO: Nodo aún no visitado
- GRIS: Nodo visitado pero con vecinos por descubrir
- NEGRO: Nodo visitado con vecinos visitados

# DFS, estrategia recursiva

INPUT : Grafo G

Dfs(G):

  for u in V(G):

    u.color ← blanco

  for u in V(G):

    if u.color = blanco:

      DfsVisit(G,u)

INPUT : Grafo G, nodo u in V(G)

DfsVisit(G,u):

  u.color ← gris

  for v in *Ng(u)*:

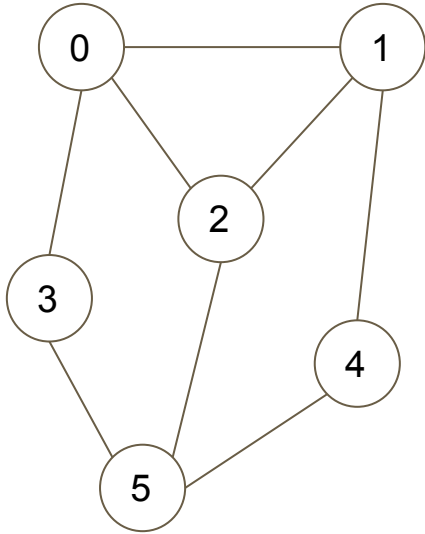
    if v.color = blanco:

      DfsVisit(G,v)

  u.color ← negro

*Ng(u):* vecinos de u, i.e. nodos  
apuntados por aristas desde u

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

    u.color ← gris

**for** v in Ng(u):

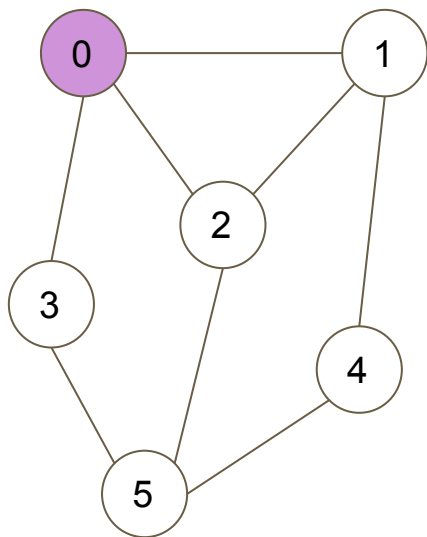
**if** v.color = blanco:

            DfsVisit(G,v)

    u.color ← negro



# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

$u.color \leftarrow \text{gris}$

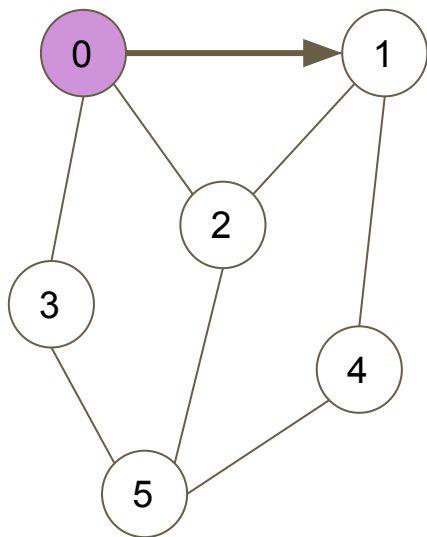
**for** v in  $Ng(u)$ :

**if** v.color = blanco:

DfsVisit(G,v)

$u.color \leftarrow \text{negro}$

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

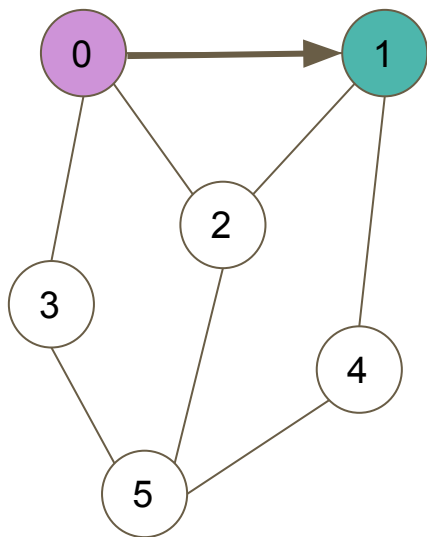
**for** v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

    u.color ← gris

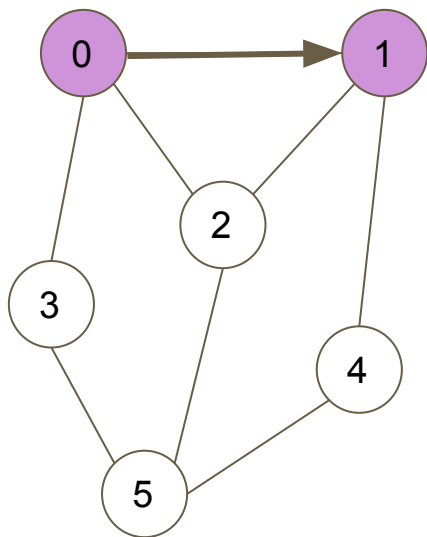
    for v in Ng(u):

        if v.color = blanco:

            DfsVisit(G,v)

    u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

**u.color** ← gris

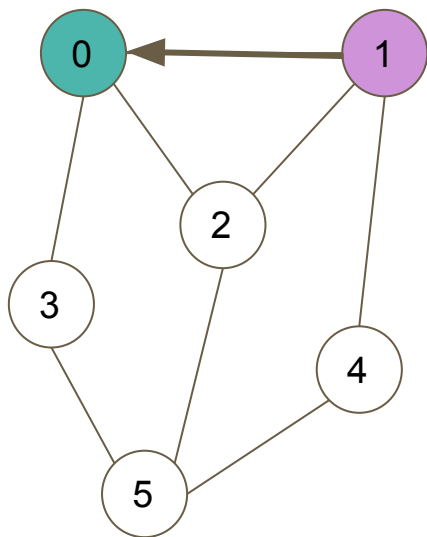
**for** v in Ng(u):

**if** v.color = blanco:

        DfsVisit(G,v)

**u.color** ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

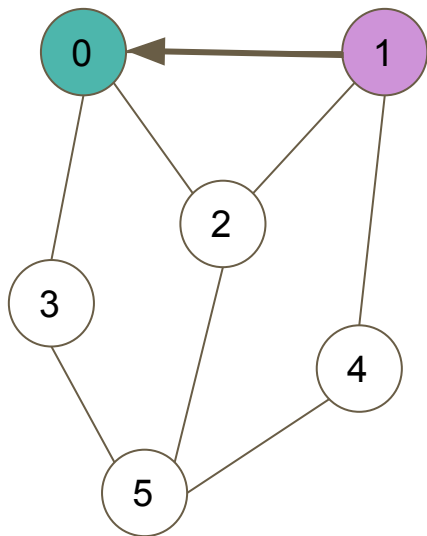
**for** v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

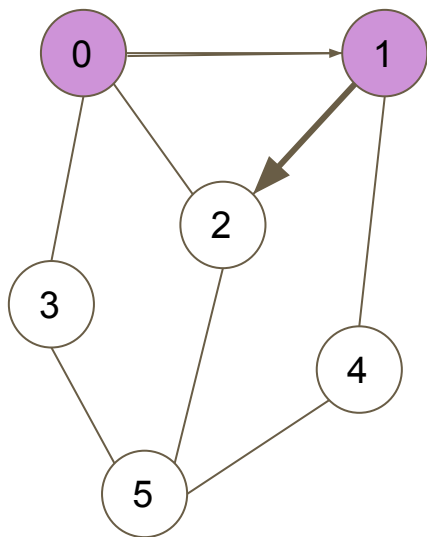
for v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

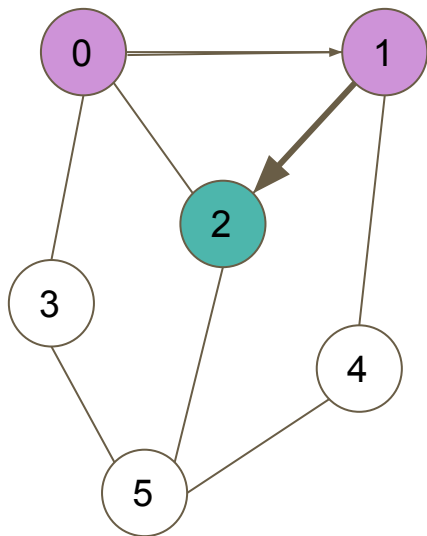
**for** v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

for v in Ng(u):

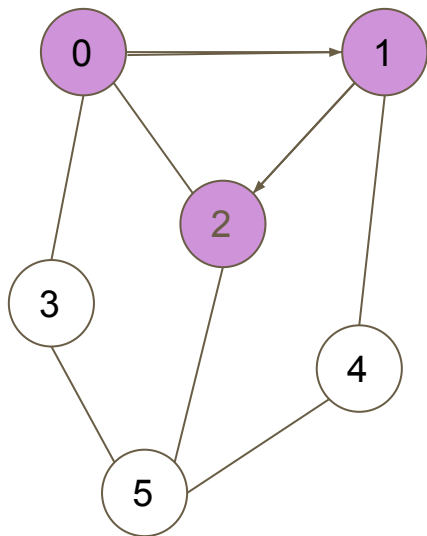
if v.color = blanco:

DfsVisit(G,v)

u.color ← negro



# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

$u.color \leftarrow \text{gris}$

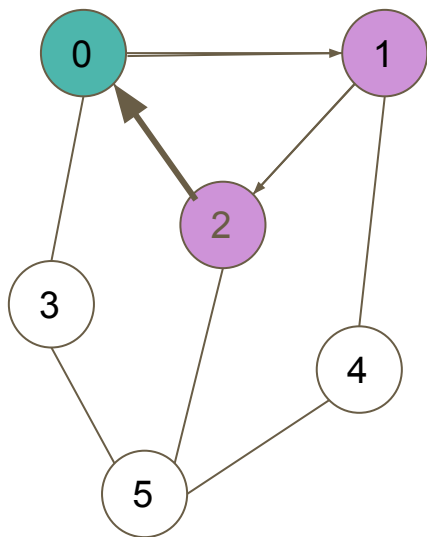
**for** v in  $Ng(u)$ :

**if** v.color = blanco:

DfsVisit(G,v)

$u.color \leftarrow \text{negro}$

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

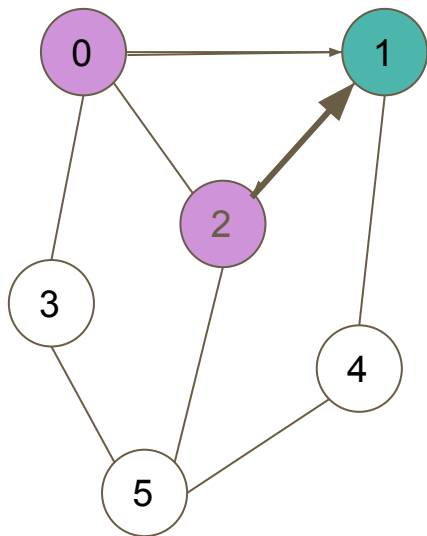
**for** v in Ng(u):

**if** v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

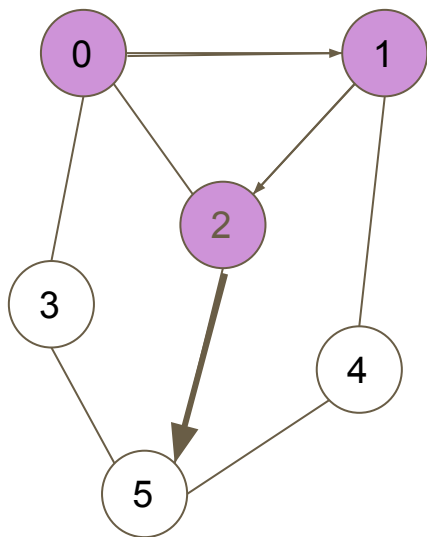
**for** v in Ng(u):

**if** v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

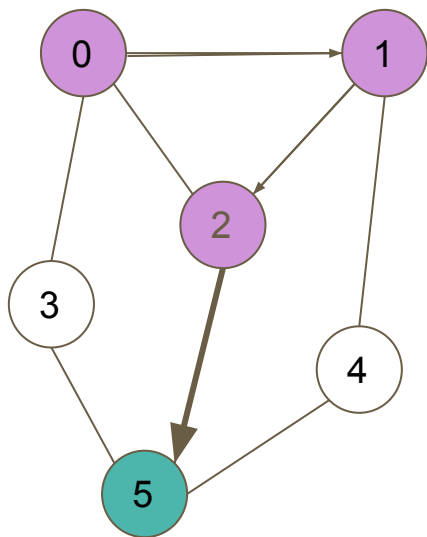
**for** v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

    u.color ← gris

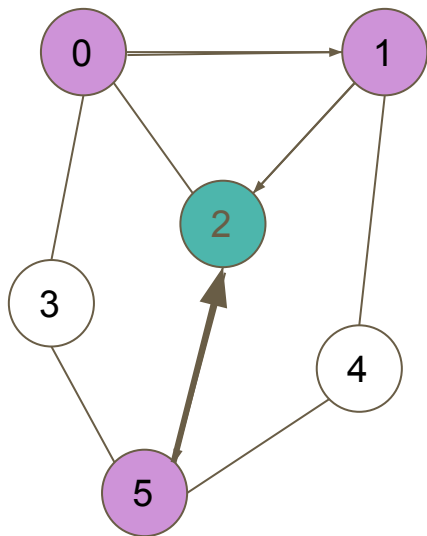
    for v in Ng(u):

        if v.color = blanco:

            DfsVisit(G,v)

    u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

    u.color ← gris

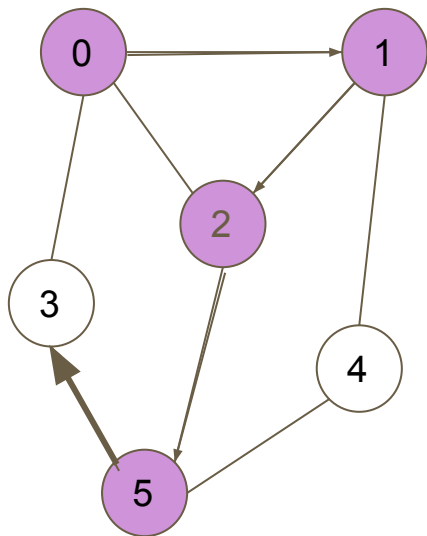
    for v in Ng(u):

        if v.color = blanco:

            DfsVisit(G,v)

    u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

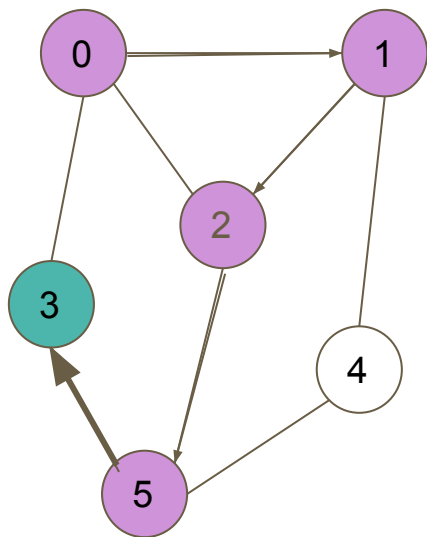
**for** v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

for v in Ng(u):

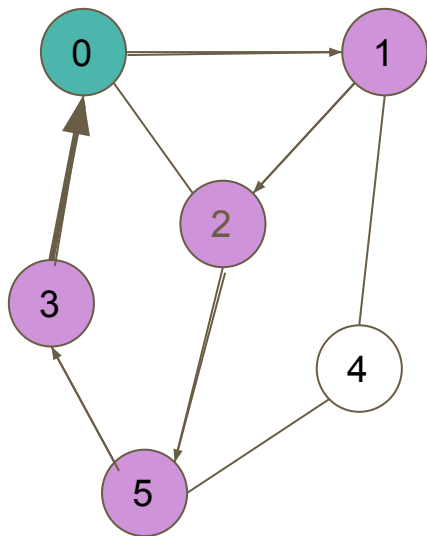
if v.color = blanco:

DfsVisit(G,v)

u.color ← negro



# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

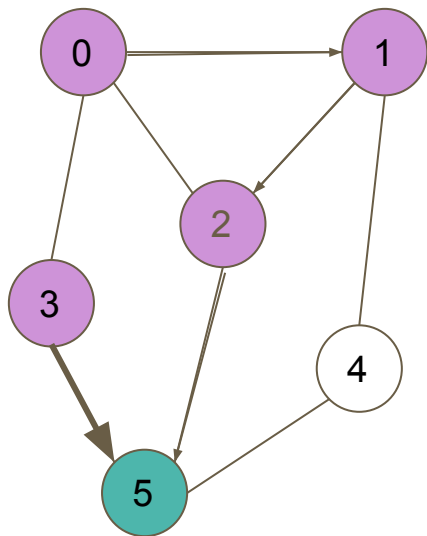
for v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



Nota: Todos los vecinos de 3 han sido visitados (gris)...

DfsVisit(G, 0)

DfsVisit(G,u):

u.color ← gris

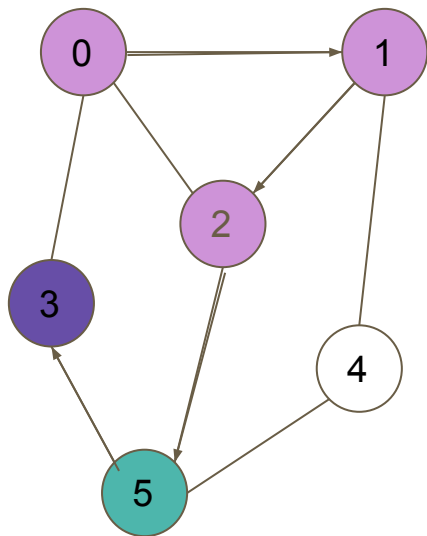
for v in Ng(u):

if v.color = blanco:

DfsVisit(G,v)

u.color ← negro

# Ejemplo



Nota: Todos los vecinos de 3 han sido visitados (gris). Por lo tanto, lo pintamos de **negro**

DfsVisit(G, 0)

DfsVisit(G,u):

$u.color \leftarrow \text{gris}$

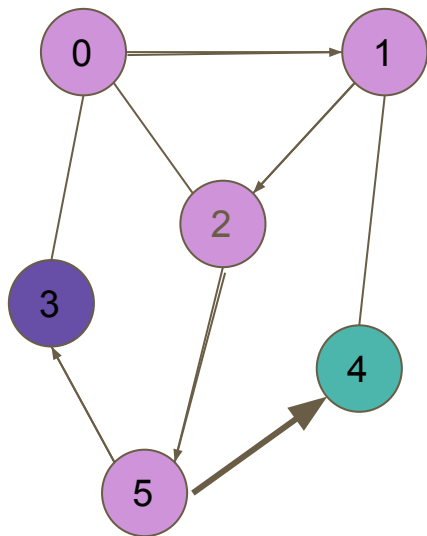
for v in  $Ng(u)$ :

if v.color = blanco:

DfsVisit(G,v)

$u.color \leftarrow \text{negro}$

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

    u.color ← gris

**for** v in Ng(u):

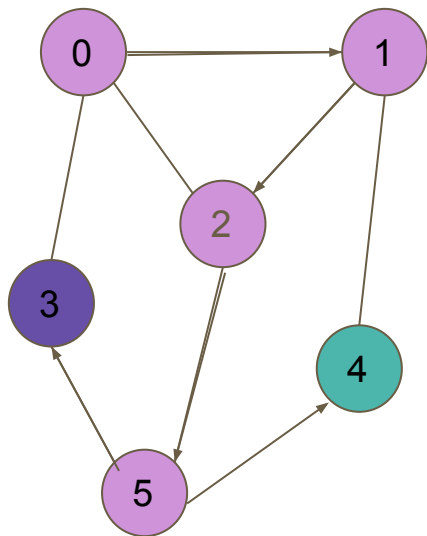
**if** v.color = blanco:

            DfsVisit(G,v)

    u.color ← negro

Nota: El algoritmo continúa por un rato más, pero no se mostrará el resto

# Ejemplo



DfsVisit(G, 0)

DfsVisit(G,u):

    u.color ← gris

    for v in Ng(u):

        if v.color = blanco:

            DfsVisit(G,v)

    u.color ← negro

Nota: El algoritmo continúa por un más, pero no se mostrará el resto

# Ejercicio

**Escriba un algoritmo que determine si es posible llegar del vértice  $v$  al  $u$  en un grafo dirigido. Indique el camino encontrado.**

# Ejercicio

**Escriba un algoritmo que determine si es posible llegar del vértice  $v$  al  $u$  en un grafo dirigido. Indique el camino encontrado.**

Para esto podemos seguir ocupando la estrategia DFS. Notar que solo tendremos que realizar un llamado a ***DfsVisit( $G, v$ )*** (asumiendo que es **conexo**), ***al cual vamos a tener que modificar***, terminado el algoritmo prematuramente si es que encontramos a  $u$ , retornando TRUE, de lo contrario el algoritmo recorrerá todo el grafo y al terminar retornará FALSE.

**friendly reminder:**

**"grafo conexo"** = todos sus vértices están conectados por un camino

# Ejercicio

Escriba un algoritmo que determine si es posible llegar del vértice  $v$  al  $u$  en un grafo dirigido. Indique el camino encontrado.

INPUT : Grafo  $G$ , nodo  $v$  in  $V(G)$

DfsVisit( $G, v, u$ ):

$v.color \leftarrow \text{gris}$

    for  $p$  in  $Ng(v)$ :

        if  $p = u$ :

            return TRUE

        if  $p.color = \text{blanco}$ :

            DfsVisit( $G, p, u$ )

$v.sgteCamino \leftarrow p$

$v.color \leftarrow \text{negro}$

*Recorrer  $v$  a  $p$  y borrar  $.sgteCamino$  (opcional)*

    return FALSE



# Ejercicio

**Escriba un algoritmo que determine si es posible llegar del vértice  $v$  al  $u$  en un grafo dirigido. Indique el camino encontrado.**

**¿Cómo se puede relacionar/diferenciar con backtracking?**

# Orden Topológico

## Palabras Clave

- $G$ , grafo **dirigido**
- Secuencia de nodos

# Orden Topológico

## Definición

- Sea **G** un grafo dirigido. Un **orden topológico** de **G** es una **secuencia** de sus nodos

$$v_0, v_1, \dots, v_n \quad v_i \in V(G)$$

- Tal que
  1. Todo nodo del grafo **aparece** en la secuencia
  2. En la secuencia **no hay** elementos repetidos
  3. Si  $(a,b) \in E(G)$  entonces el nodo **a** aparece antes que el nodo **b** en la secuencia

# Orden Topológico: Pseudocódigo

**input** : grafo  $G$

**output**: lista de nodos  $L$

TopSort( $G$ ):

```
1   $L \leftarrow$  lista vacía
2   $t \leftarrow 1$ 
3  for  $u \in V(G)$  :
4       $u.start \leftarrow 0$ 
5       $u.end \leftarrow 0$ 
6  for  $u \in V(G)$  :
7      if  $u.start = 0$  :
8          TopDfsVisit( $G, L, u, t$ )
9  return  $L$ 
```

**input** : grafo  $G$ , lista de nodos  $L$ ,  
nodo  $u \in V(G)$ , tiempo  $t$

**output**: tiempo  $t \geq 1$

TopDfsVisit( $G, L, u, t$ ):

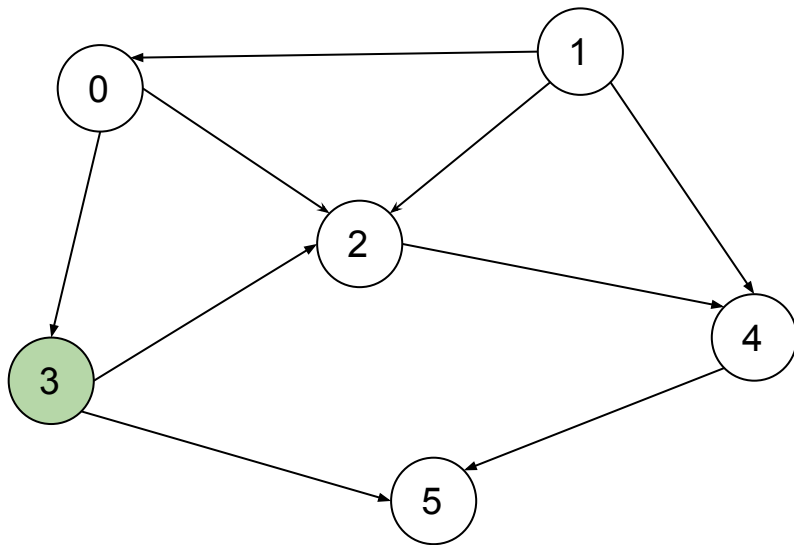
```
1   $u.start \leftarrow t$ 
2   $t \leftarrow t + 1$ 
3  for  $v \in N_G(u)$  :
4      if  $v.start = 0$  :
5          TopDfsVisit( $G, L, v, t$ )
6   $u.end \leftarrow t$ 
7  Insertar  $u$  como cabeza de  $L$ 
8   $t \leftarrow t + 1$ 
9  return  $t$ 
```

# Orden topológico

## ¿Cómo se ve?

1. Elegimos un nodo para comenzar, en este caso usaremos el nodo 3

t=0

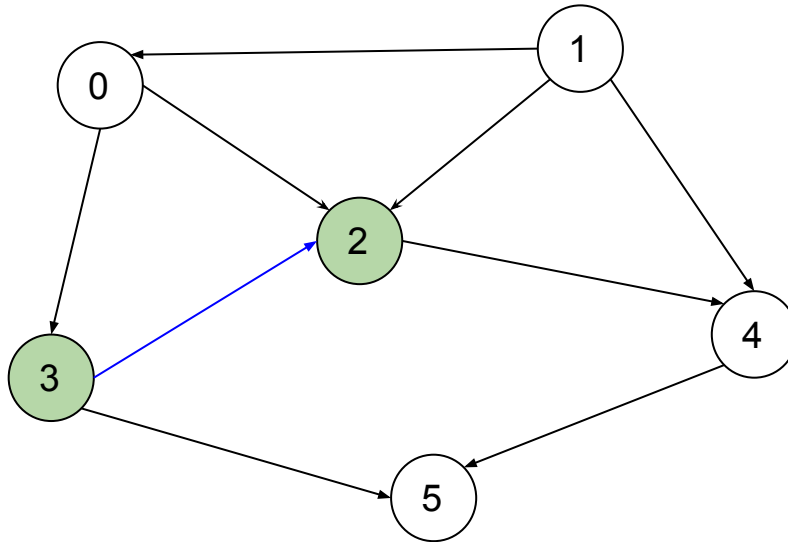


Nodo	Start	End	OT
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	
5	0	0	

# Orden topológico

- Elegimos un nodo hijo del nodo actual: 2

t=1

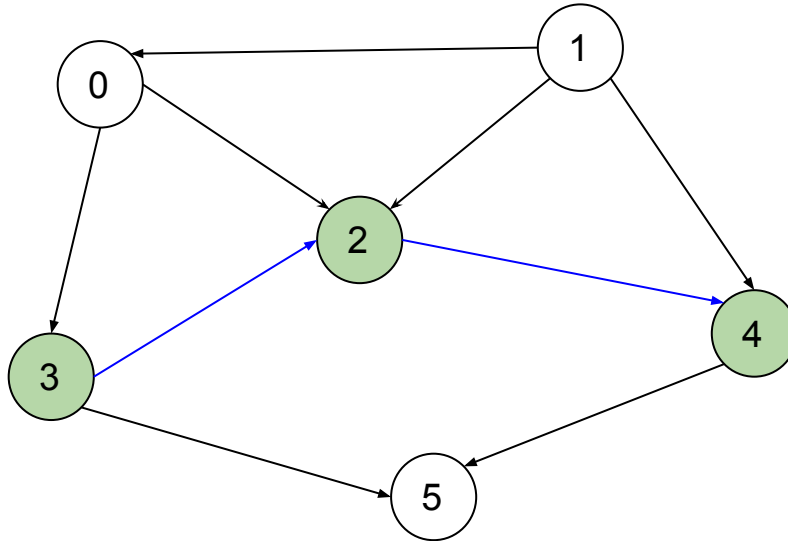


Nodo	Start	End	OT
0	0	0	
1	0	0	
2	1	0	
3	0	0	
4	0	0	
5	0	0	

# Orden topológico

- Elegimos un nodo hijo del nodo actual: 4

t=2

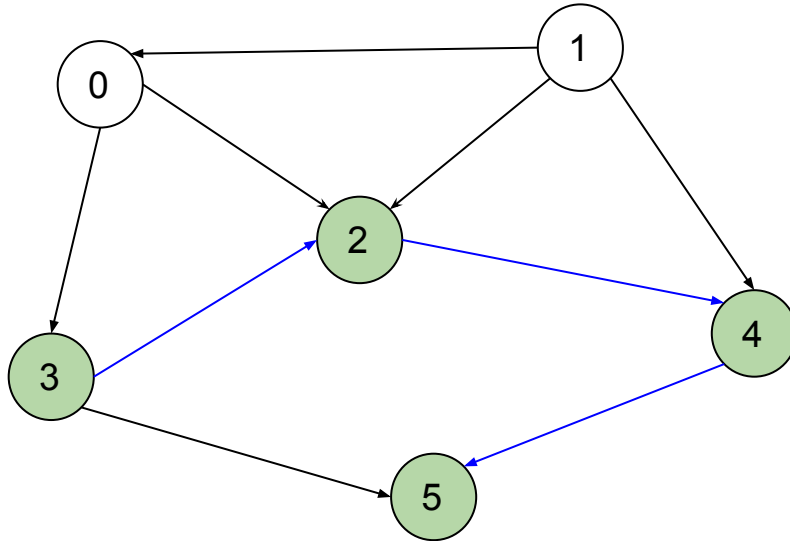


Nodo	Start	End	OT
0	0	0	
1	0	0	
2	1	0	
3	0	0	
4	2	0	
5	0	0	

# Orden topológico

- Elegimos un nodo hijo del nodo actual: 5

t=3



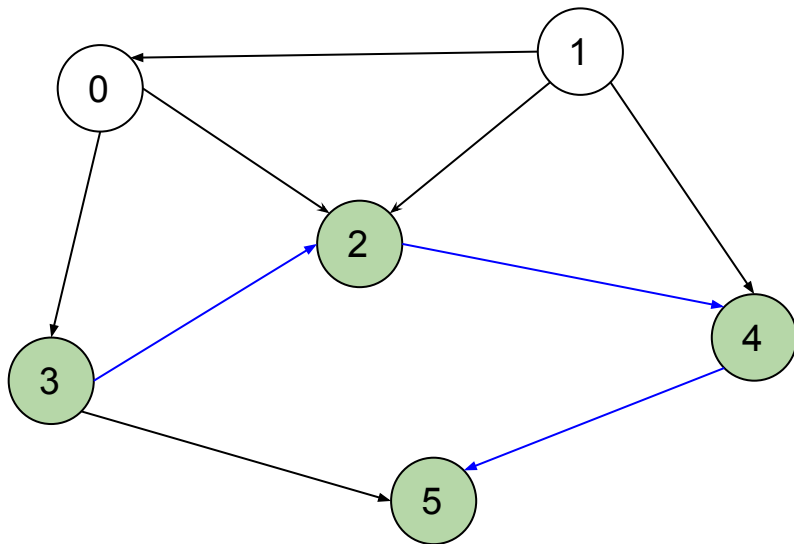
Nodo	Start	End	OT
0	0	0	
1	0	0	
2	1	0	
3	0	0	
4	2	0	
5	3	0	



# Orden topológico

- El nodo actual no tiene nodos hijos, por lo cual lo agregamos al orden topológico
- Seteamos su End en  $t=4$

$t=4$

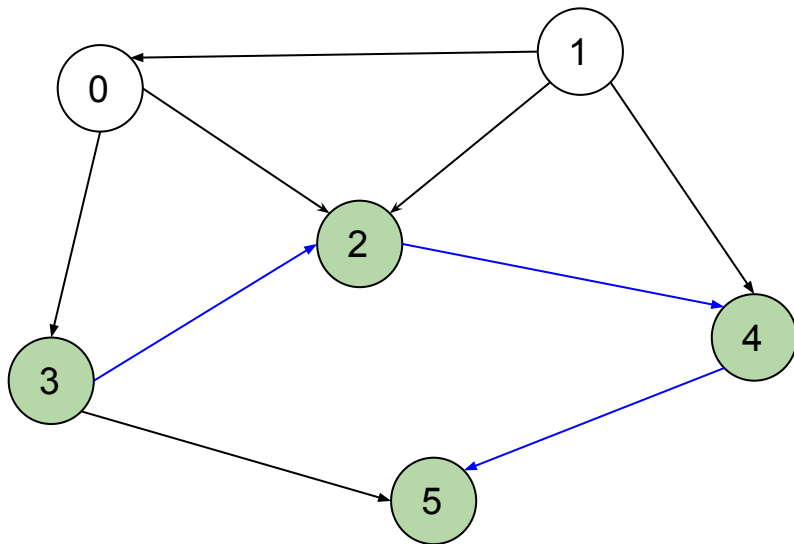


Nodo	Start	End	OT
0	0	0	5
1	0	0	
2	1	0	
3	0	0	
4	2	0	
5	3	4	

# Orden topológico

- Volvemos al nodo (4) y este no tiene más hijos. Lo agregamos al OT
- Seteamos su End en  $t=5$

$t=5$

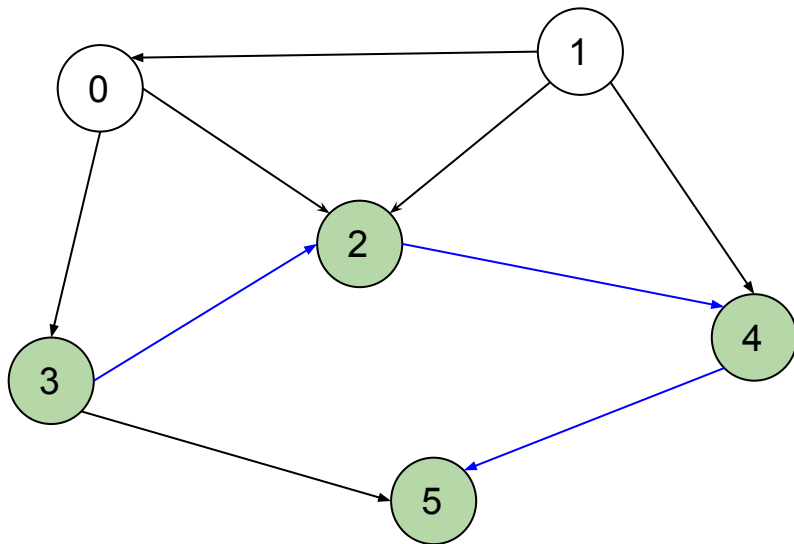


Nodo	Start	End	OT
0	0	0	4
1	0	0	5
2	1	0	
3	0	0	
4	2	5	
5	3	4	

# Orden topológico

- Volvemos al nodo (2) y este no tiene más hijos. Lo agregamos al OT
- Seteamos su End en  $t=6$

$t=6$

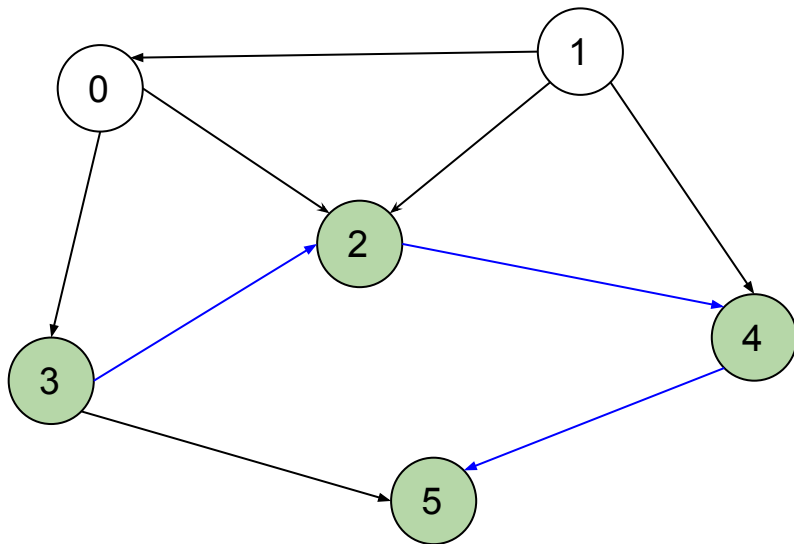


Nodo	Start	End	OT
0	0	0	2
1	0	0	4
2	1	6	5
3	0	0	
4	2	5	
5	3	4	

# Orden topológico

- Volvemos al nodo (3) y tiene el nodo (5) de hijo. El start del nodo (5) no es 0. Lo agregamos al OT
- Seteamos su End en  $t=7$

$t=7$

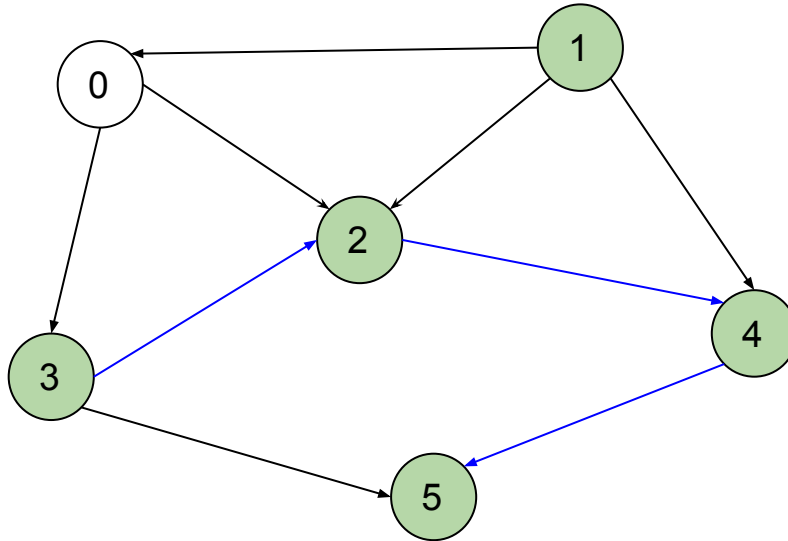


Nodo	Start	End	OT
0	0	0	3
1	0	0	2
2	1	6	4
3	0	7	5
4	2	5	
5	3	4	

# Orden topológico

- Elegimos un nodo restante, elegimos el (1)

t=8

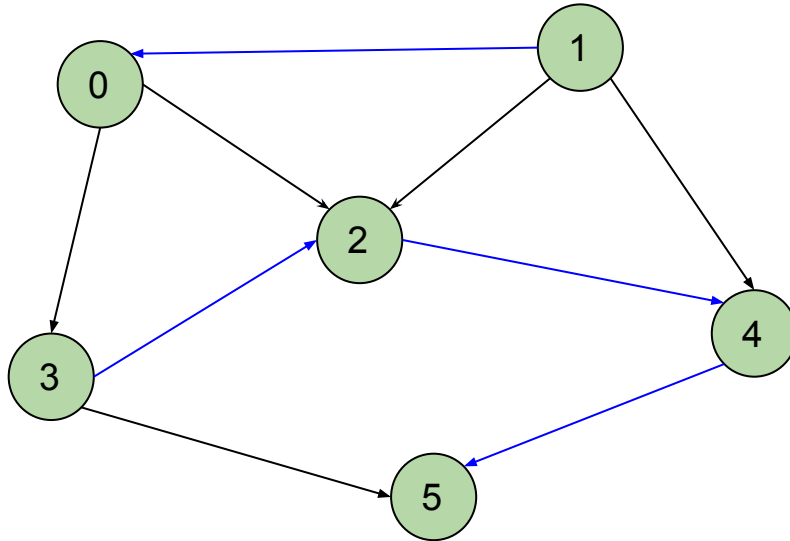


Nodo	Start	End	OT
0	0	0	3
1	8	0	2
2	1	6	4
3	0	7	5
4	2	5	
5	3	4	

# Orden topológico

- Elegimos un nodo hijo: (0)

t=9

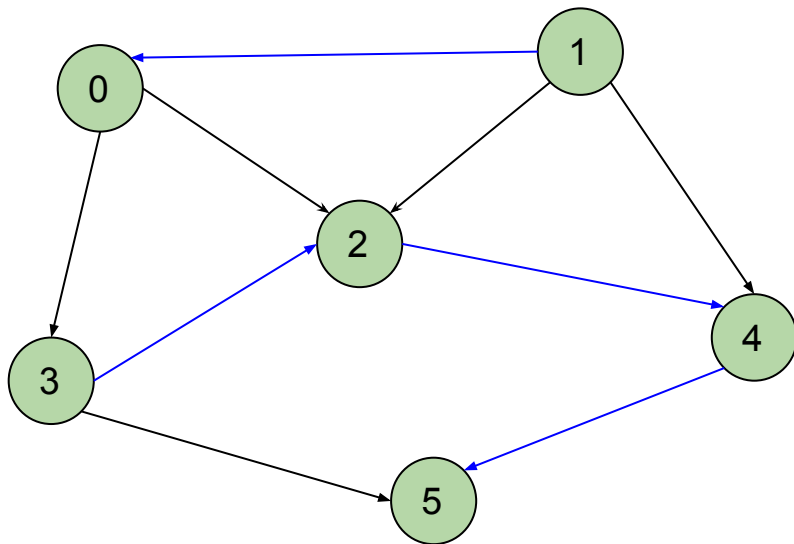


Nodo	Start	End	OT
0	9	0	3
1	8	0	2
2	1	6	4
3	0	7	5
4	2	5	
5	3	4	

# Orden topológico

- El nodo 0 no tiene hijos con start=0. Lo agregamos al OT
- Seteamos su End en  $t=10$

$t=10$

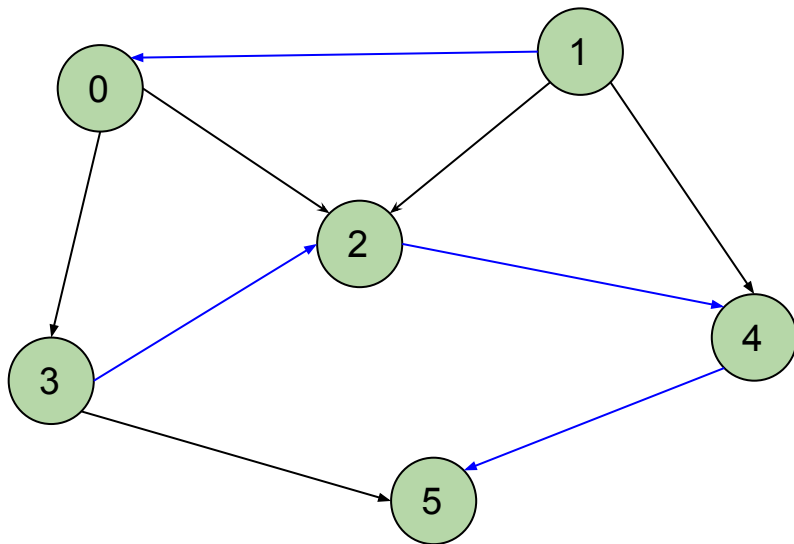


Nodo	Start	End	OT
0	9	10	0
1	8	0	3
2	1	6	2
3	0	7	4
4	2	5	5
5	3	4	

# Orden topológico

- Volvemos al nodo 1 y no tiene más hijos con start=0. Lo agregamos al OT
- Seteamos su End en t=11

t=11

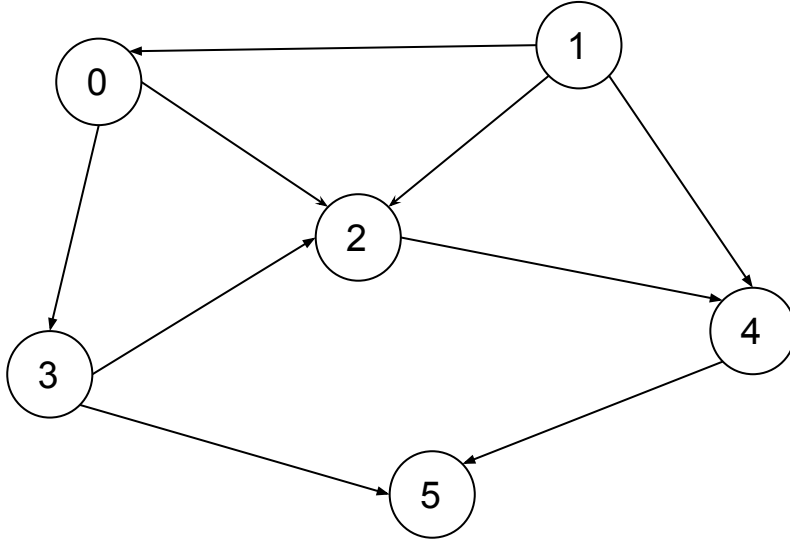


Nodo	Start	End	OT
0	9	10	1
1	8	11	0
2	1	6	3
3	0	7	2
4	2	5	4
5	3	4	5



# Orden topológico

Finalmente el orden topológico del grafo es: (1)(0)(3)(2)(4)(5)



# Orden topológico: Notas Importantes

- En resumen, orden en que vamos “descubriendo los nodos”
- Los intervalos de tiempo de los nodos no se traslapan entre ellos, por construcción

**Los grafos cíclicos no tienen orden topológico ¿Por qué?**

# Orden topológico: Notas Importantes

- En resumen, orden en que vamos “descubriendo los nodos”
- Los intervalos de tiempo de los nodos no se traslapan entre ellos, por construcción

**Los grafos cíclicos no tienen orden topológico ¿Por qué?**

No se puede definir el nodo inicial del orden del orden (regla 3)

# Componentes Fuertemente Conexas

## Palabras Clave

- $G$ , grafo **dirigido**
- Conjunto **maximal**

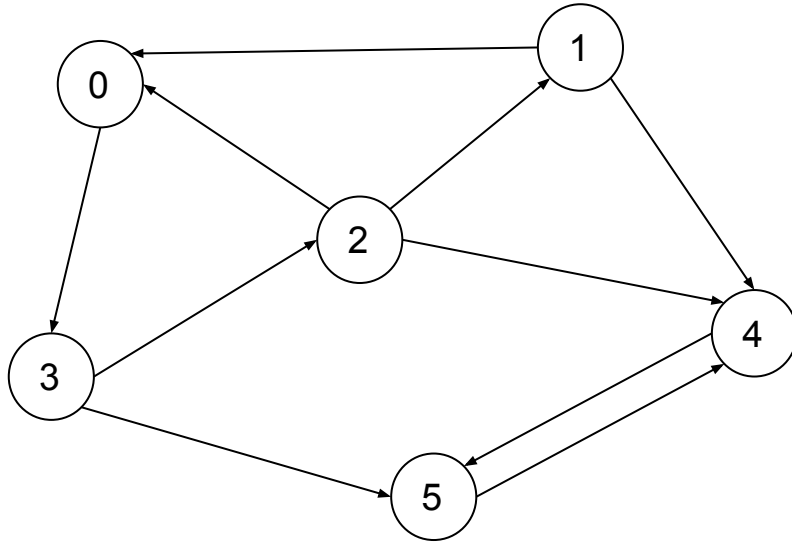
# Componentes Fuertemente Conexas (CFC)

## Definición

- En un grafo **dirigido**  $G$ , una **CFC** es un conjunto **maximal** de nodos  $C \subseteq G$  de tal manera que dados  $u, v \in C$  existe un camino dirigido desde  $u$  hasta  $v$

# Componentes Fuertemente Conexas (CFC)

¿Cómo se ve una CFC?

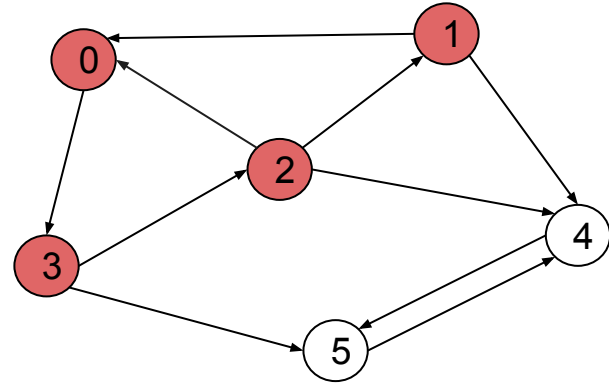
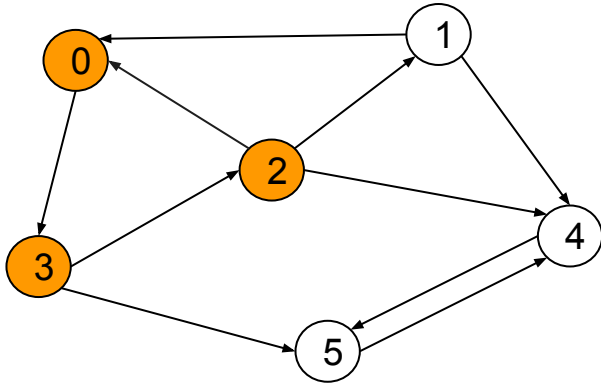


1. Grafo dirigido

# Componentes Fuertemente Conexos (CFC)

¿Cómo se ve una CFC?

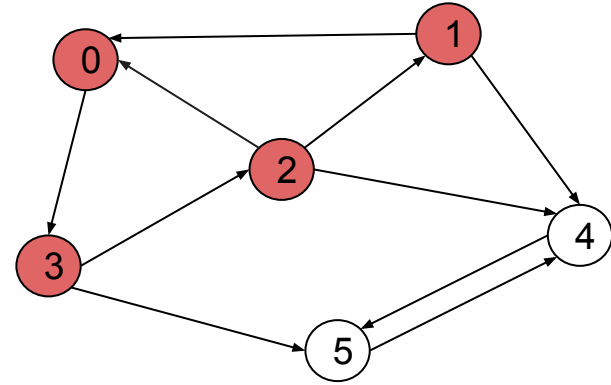
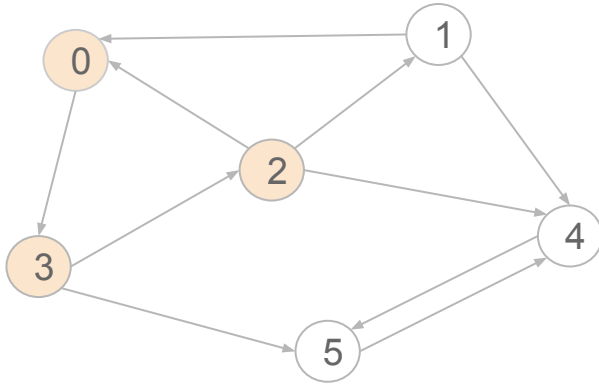
1. Grafo dirigido
2. Conjunto maximal



# Componentes Fuertemente Conexos (CFC)

¿Cómo se ve una CFC?

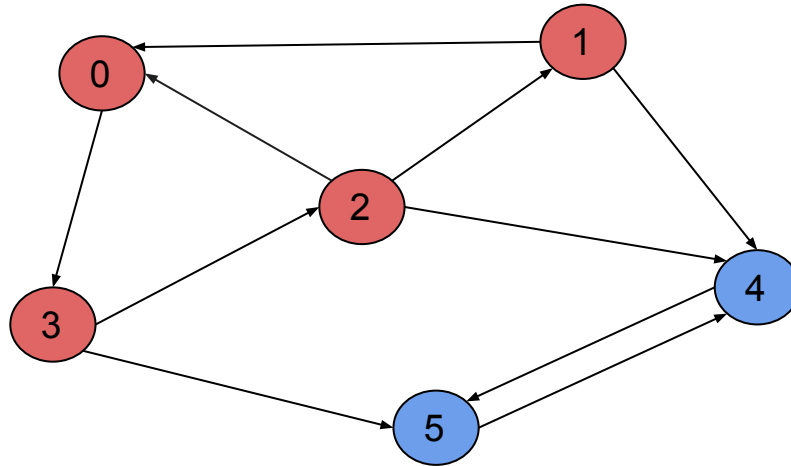
1. Grafo dirigido
2. Conjunto maximal



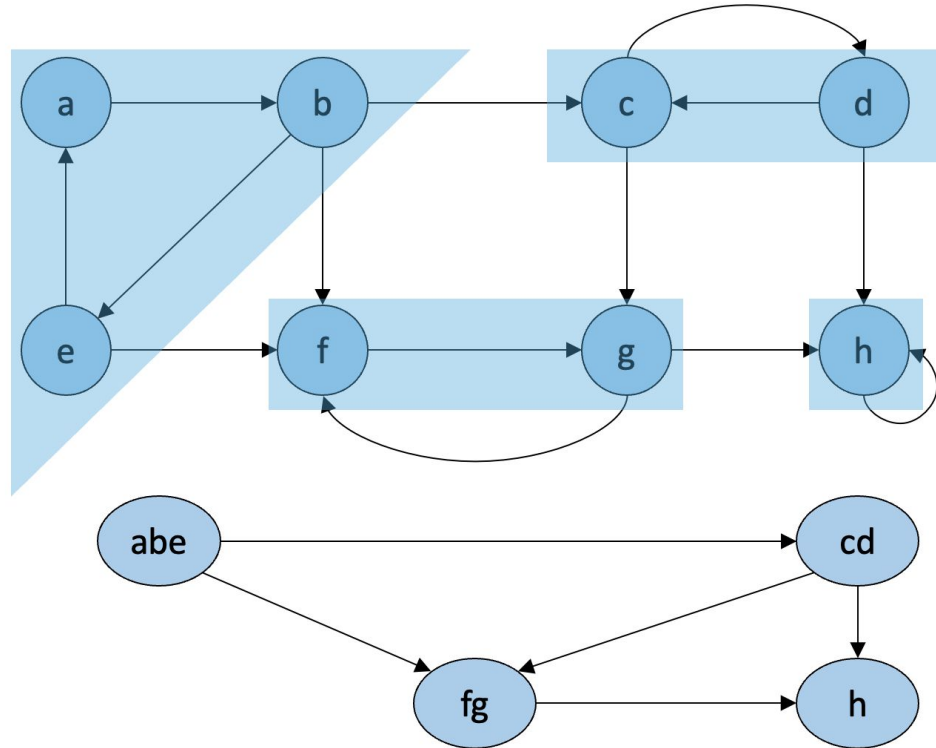


# Componentes Fuertemente Conexas (CFC)

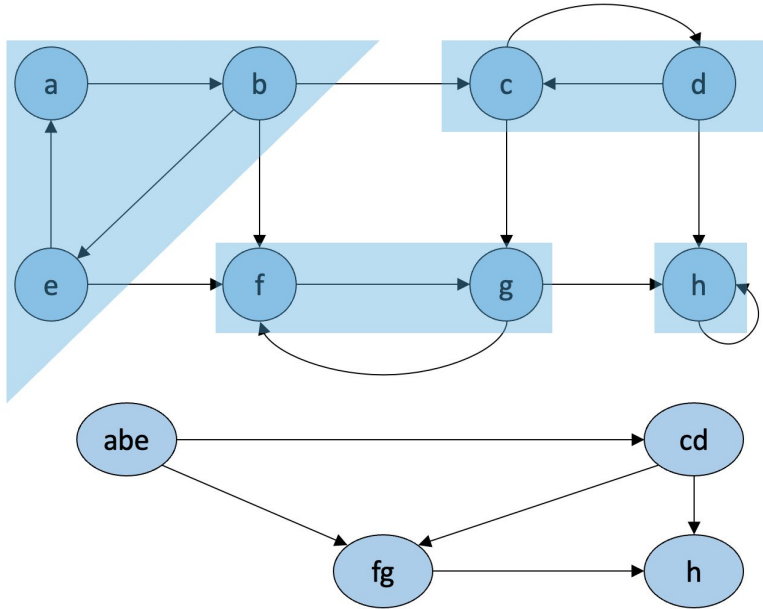
¿Cómo se ve una CFC?



# Grafo de Componentes

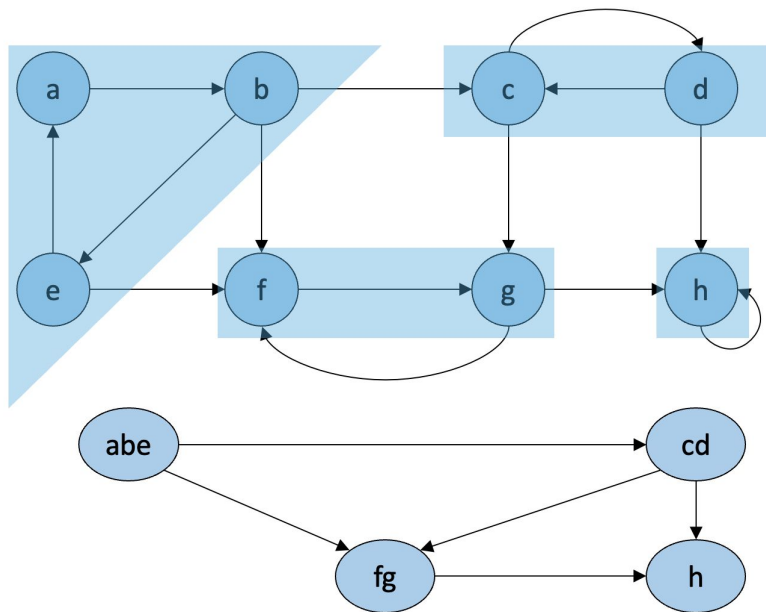


# Grafo de Componentes



**El nuevo grafo de componentes es acíclico, ¿Por qué?**

# Grafo de Componentes



**El nuevo grafo de componentes es acíclico, ¿Por qué?**

Si tuviera ciclos, los nodos de ambas CFC estarían en realidad en una misma CFC

# Kosaraju

## Palabras Clave

- $G$ , grafo **dirigido**
- Grafo de **componentes**

# Kosaraju

## Definición

- Dado un grafo **G** dirigido, sean  $C_1, \dots, C_k$  sus **CFC**. Se define el **grafo de componentes**  $G^{CFC}$  según:
  - $V(G^{CFC}) = \{C_1, \dots, C_k\}$
  - Si  $(u, v) \in E(G)$  y  $u \in C_i, v \in C_j$ , entonces  $(C_i, C_j) \in E(G^{CFC})$

# Kosaraju: Pseudocódigo

**input** : grafo  $G$

Kosaraju( $G$ ):

```
1    $L \leftarrow \text{TopSort}(G)$ 
2   for  $u \in L$  :
3       Assign( $u, u$ )
```

**input** : grafo  $G$ , nodo  $u \in V(G)$ , nodo representante  $r$

Assign( $G, u, r$ ):

```
1   if  $u.\text{rep} = \emptyset$  :
2        $u.\text{rep} \leftarrow r$ 
3       for  $v \in N_{G^T}(u)$  :
4           Assign( $G, v, r$ )
```

# Ejemplo de Kosaraju

<https://www.programiz.com/dsa/strongly-connected-components>