Exploración en profundidad, o DFS

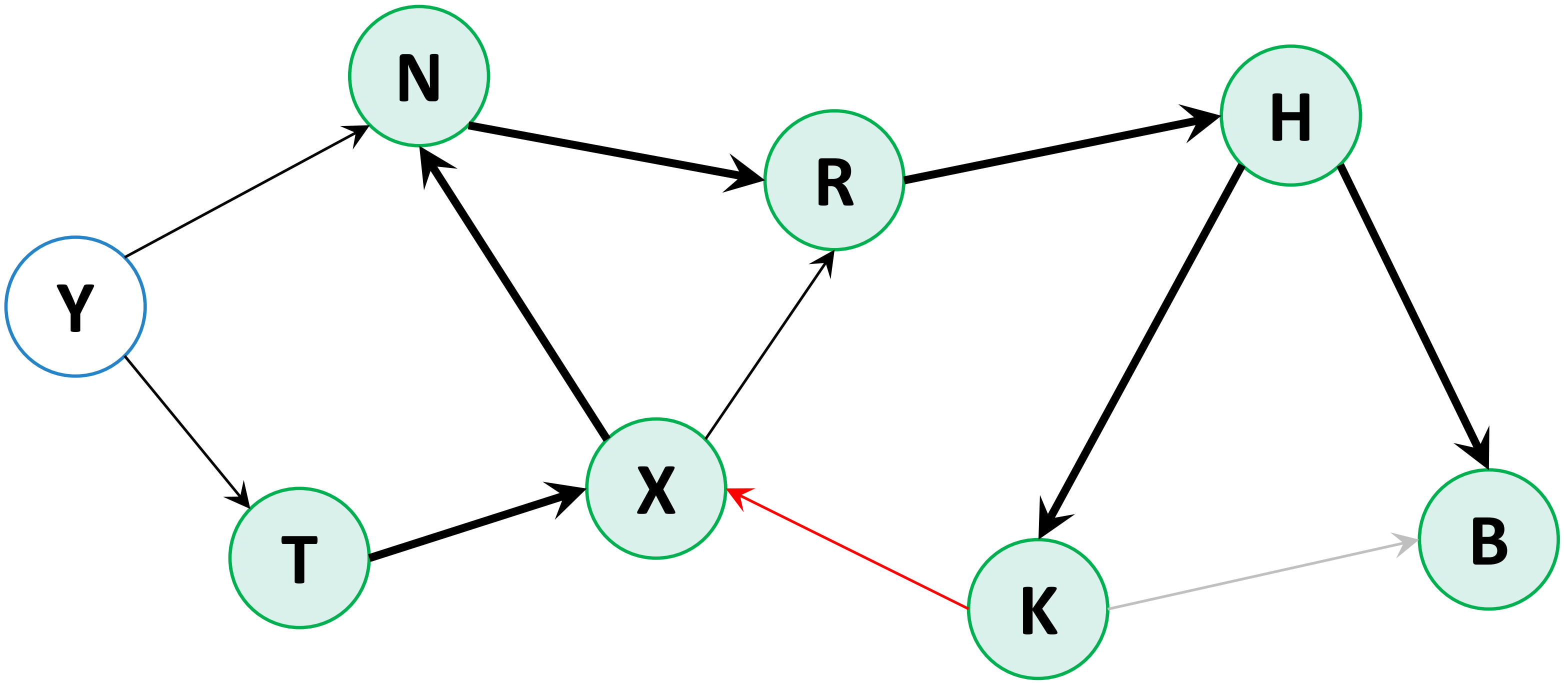
Es una forma de recorrer sistemáticamente un grafo:

- visitar todos sus nodos
- transitar todas sus aristas

Es una forma de obtener información sobre algunas propiedades del grafo —p.ej., determinar si el grafo tiene ciclos:

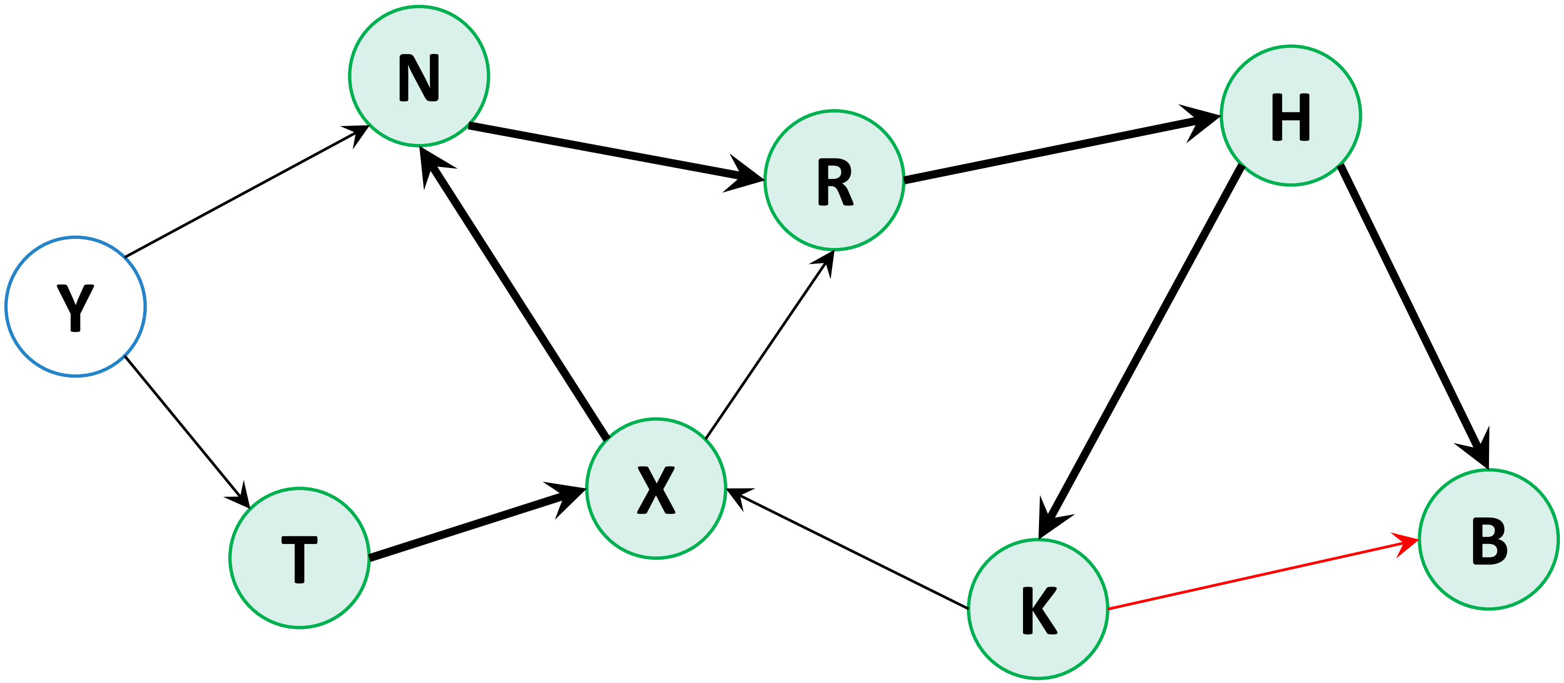
- todos los vértices son inicialmente *blancos*
- un vértice se pinta de *gris* cuando es descubierto
- un vértice se pinta de *negro* cuando su lista de adyacencias ha sido examinada exhaustivamente

Queremos distinguir este caso ...



¡Algoritmo, date cuenta de que esto es un ciclo!

… de este otro



¡Y que esto otro, no!

El algoritmo DFS

$dfs(V, E)$:

$for\ each\ u\ in\ V$:

$u.color\ =\ white$

$for\ each\ u\ in\ V$:

$if\ u.color\ ==\ white$:

$dfsVisit(u)$

$dfsVisit(u)$:

$u.color\ =\ gray$

$for\ each\ v\ in\ \alpha[u]$:

$if\ v.color\ ==\ white$:

$dfsVisit(v)$

$u.color\ =\ black$

¿Cuál es la complejidad de *dfs*?

Ahora agregamos tiempos de descubrimiento, o inicio, y de finalización

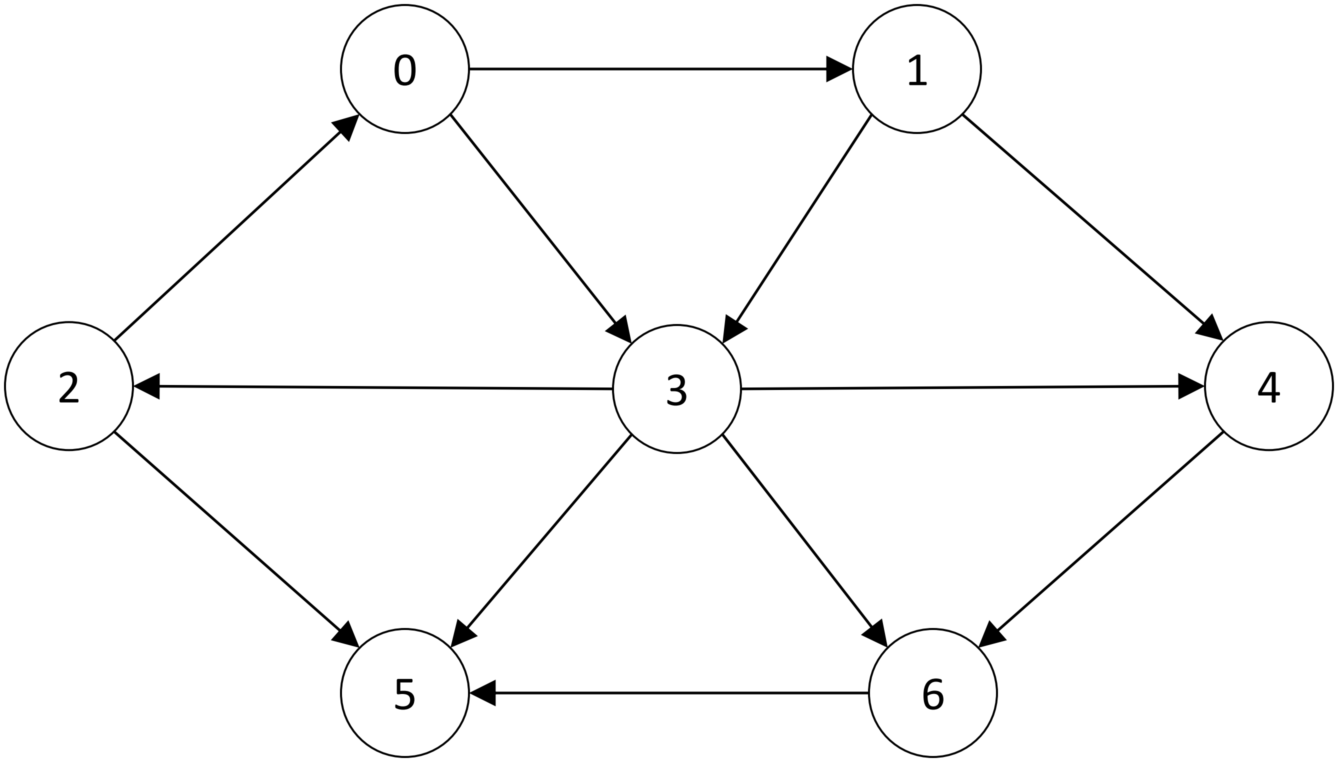
Cuando se visita un nodo blanco no solo se pinta de gris …

… además, se marca el tiempo (la hora) en que se pinta de gris

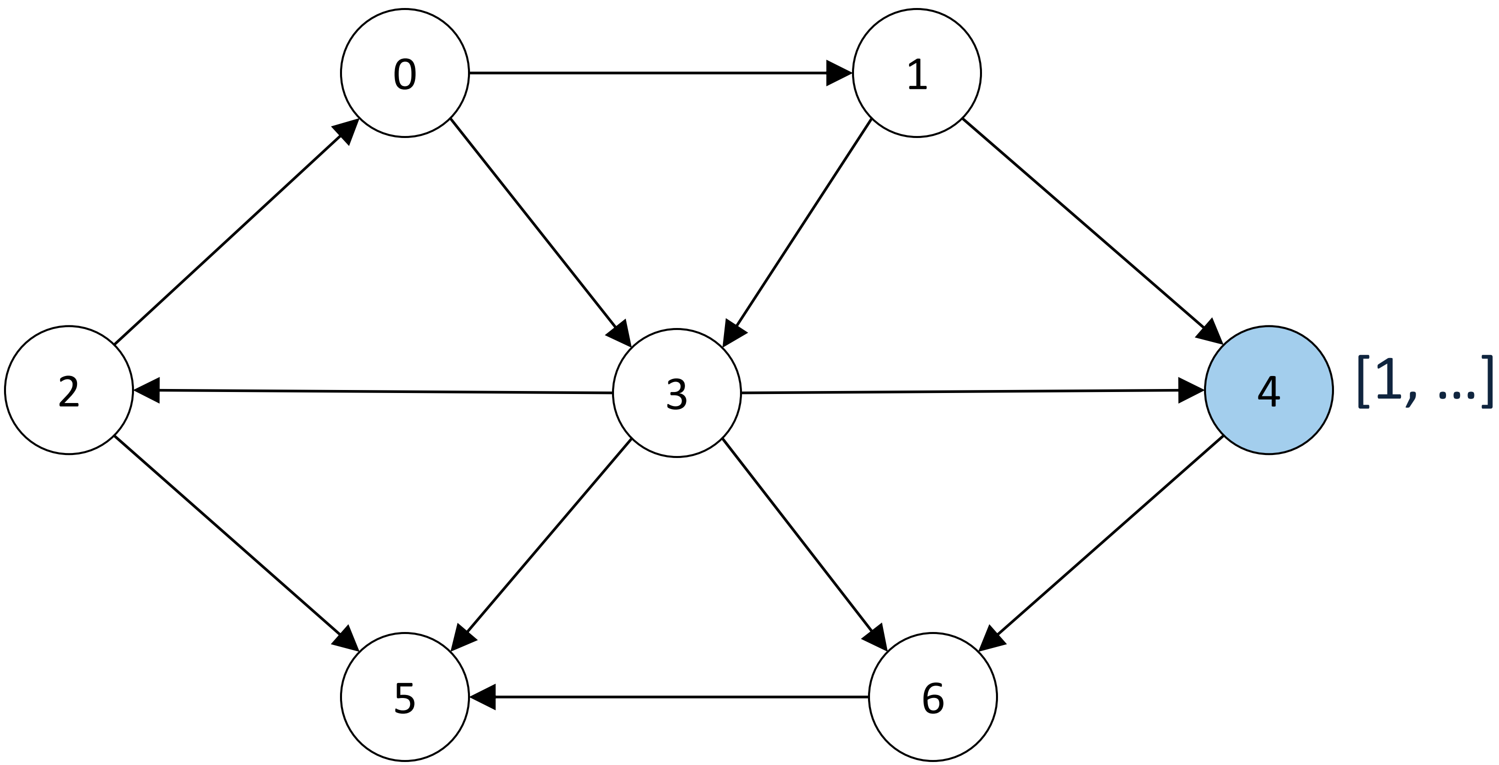
Similarmente, cuando se pinta de negro

Estos son, respectivamente, el *tiempo de inicio* y el *tiempo de finalización* de un nodo,

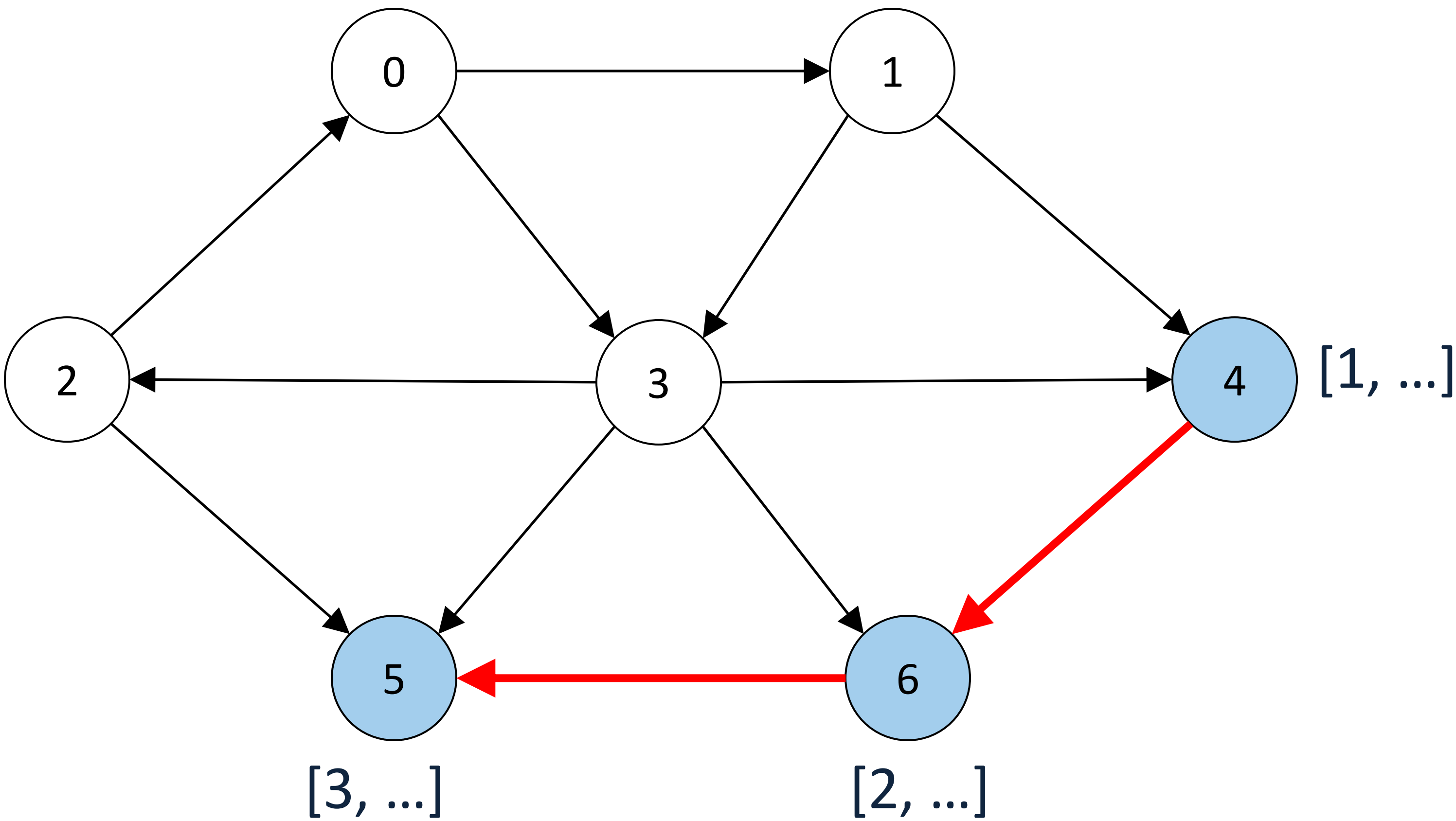
Ej.: un grafo *G* direccional



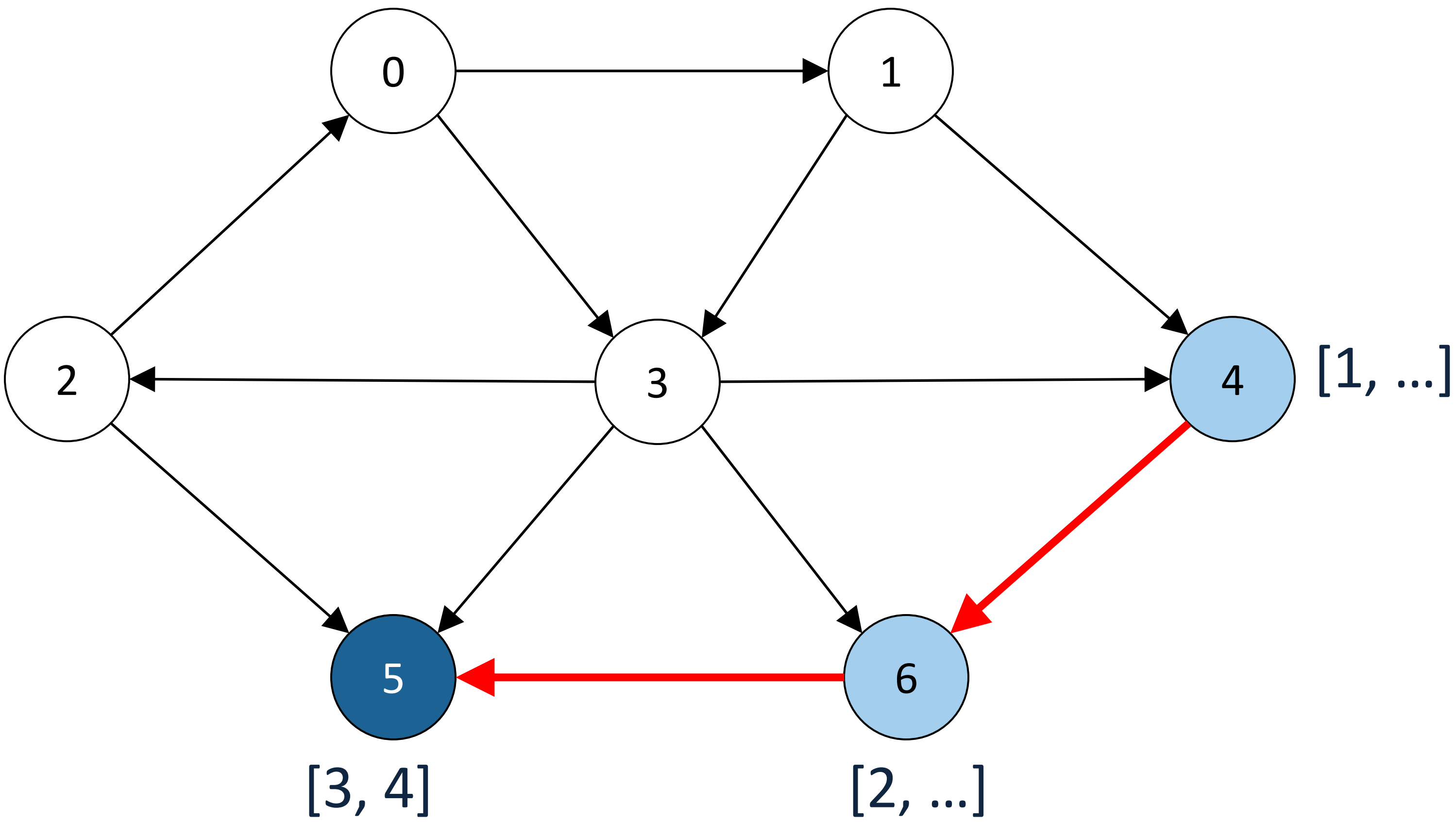
dfsVisit de *G* a partir del vértice 4



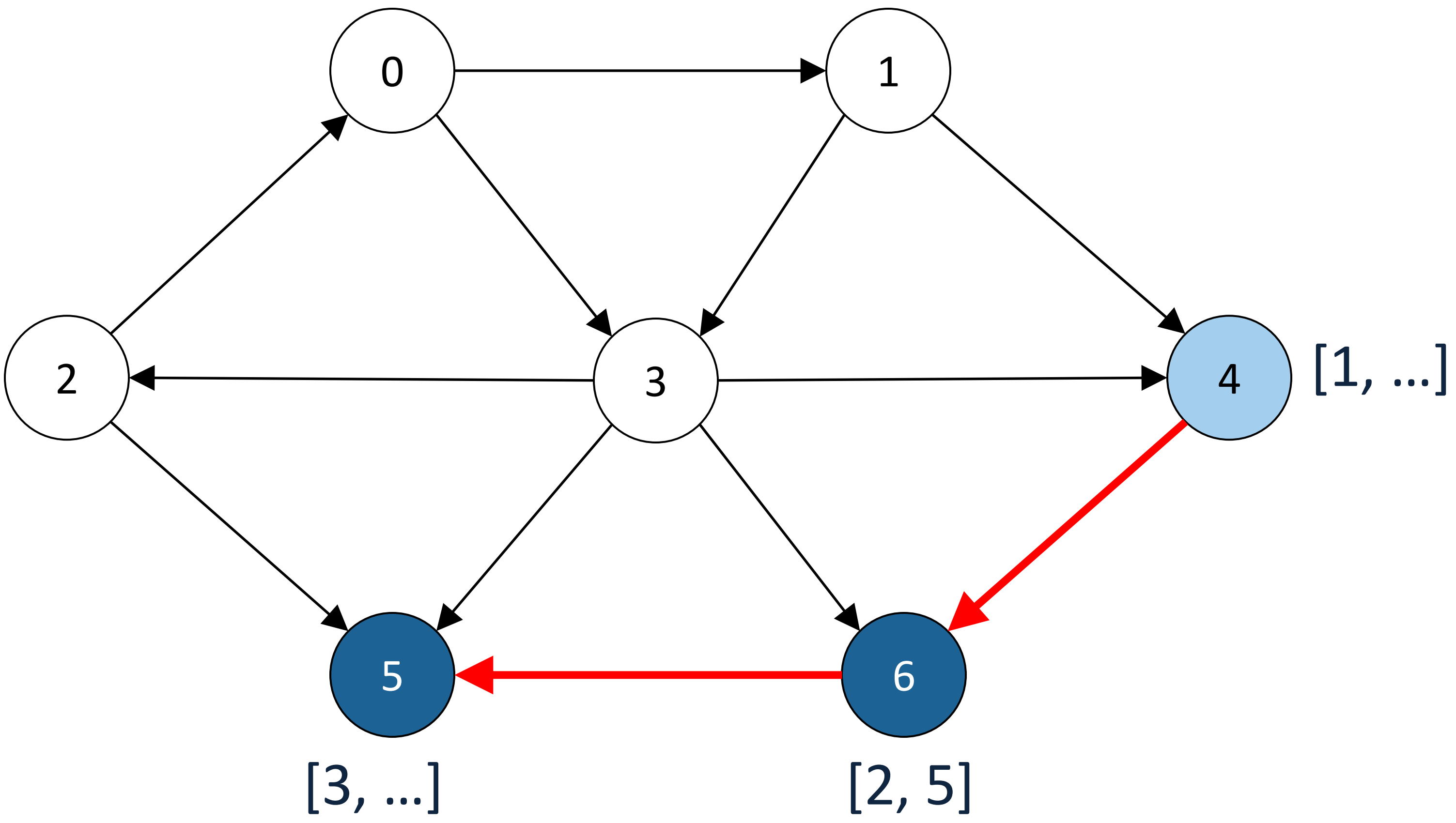
dfsVisit de *G*: del vértice 4 vamos al 6 y de ahí al 5



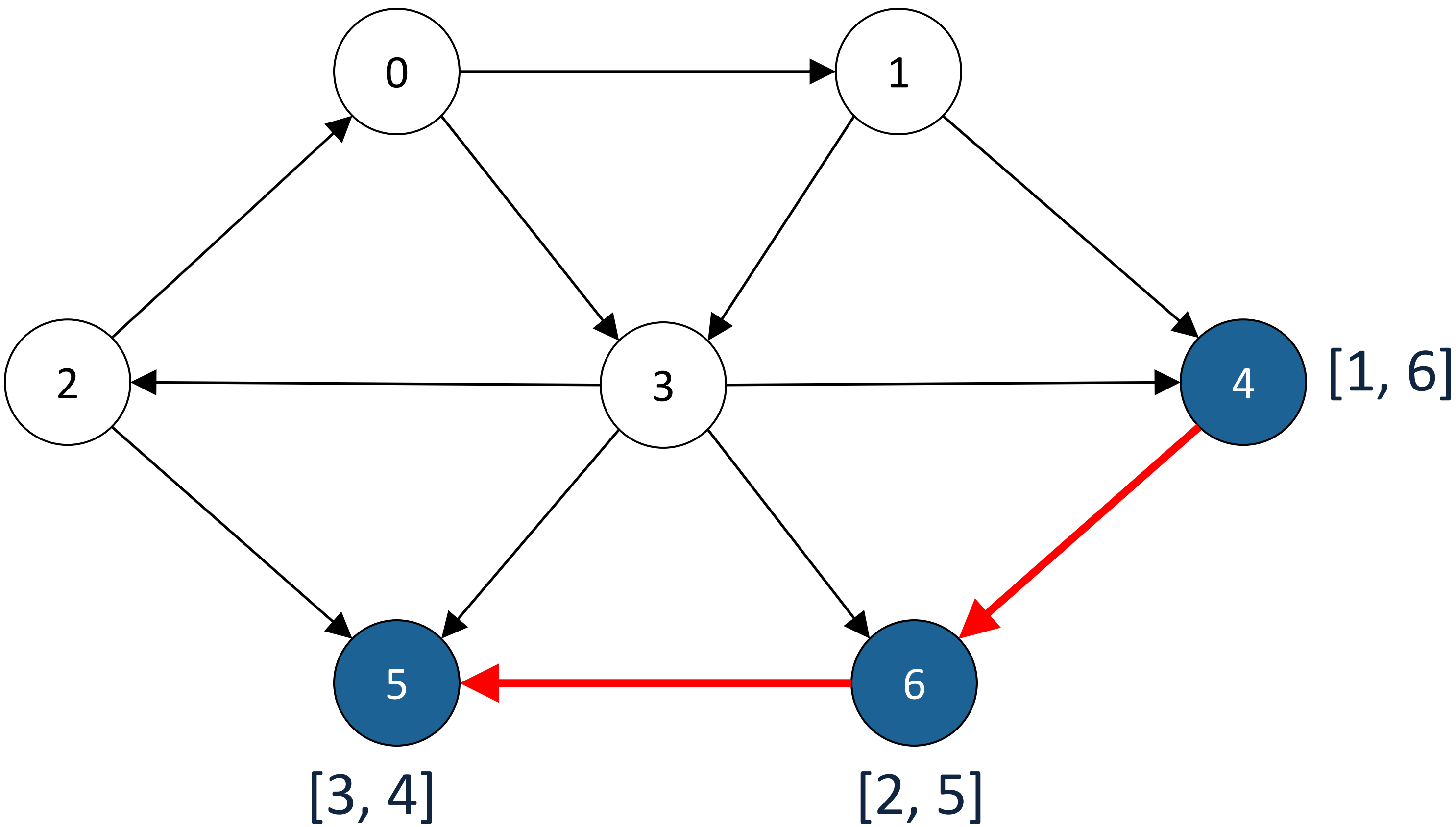
Como desde 5 no podemos seguir, lo terminamos y volvemos a 6



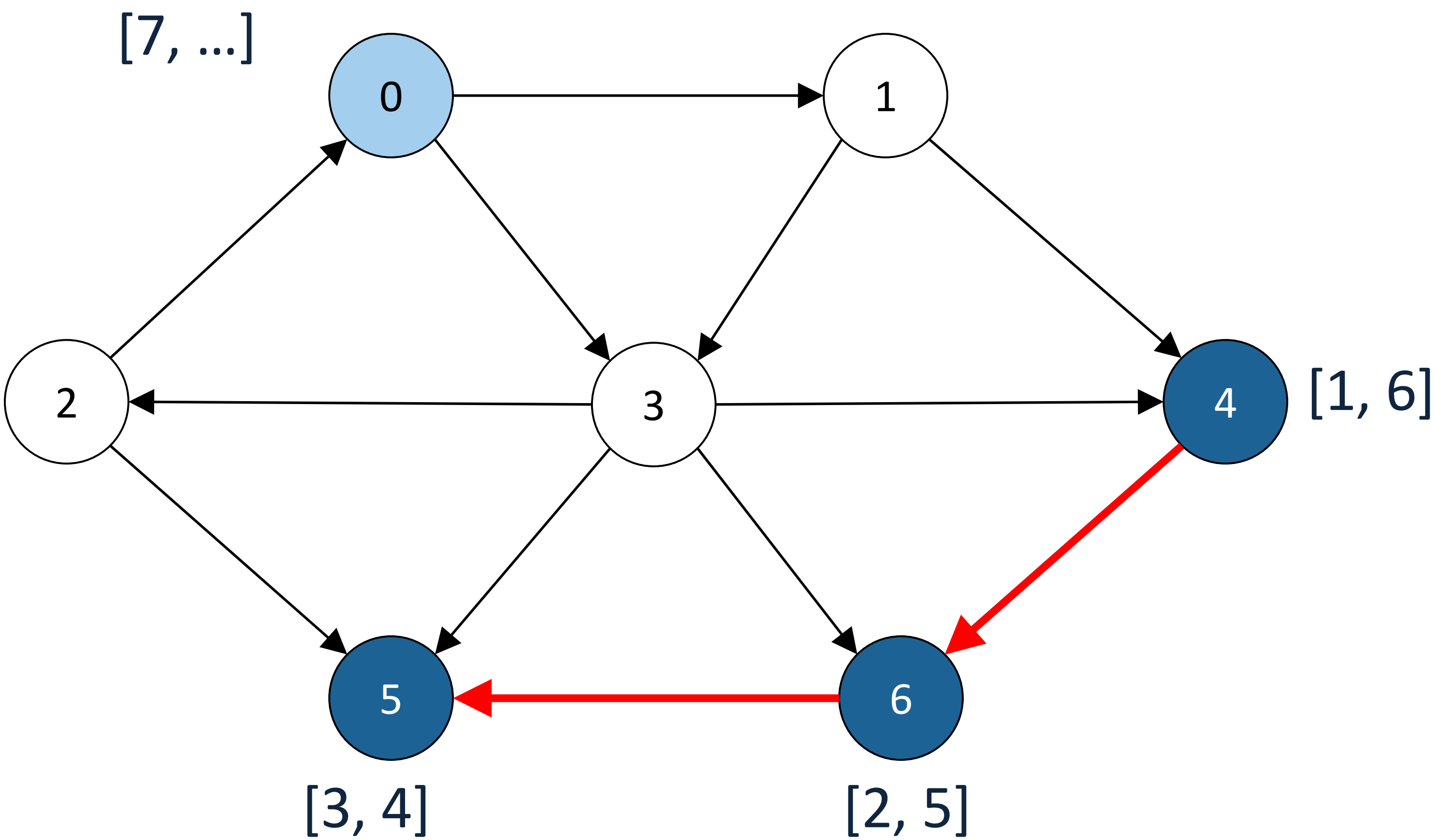
Como desde 6 no salen otras aristas, lo terminamos y volvemos a 4



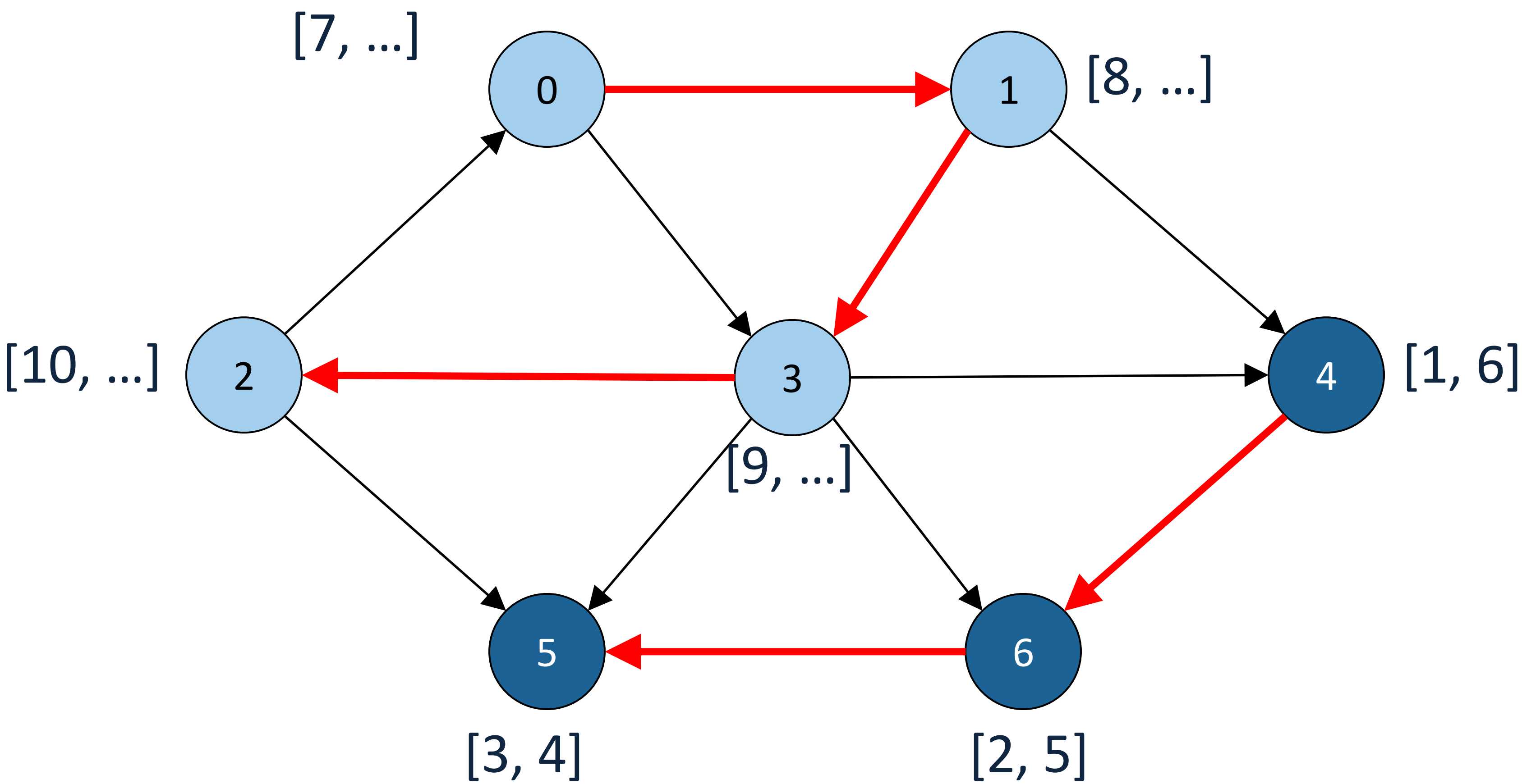
Como desde 4 no salen otras aristas → terminamos *dfsVisit* de *G* desde 4



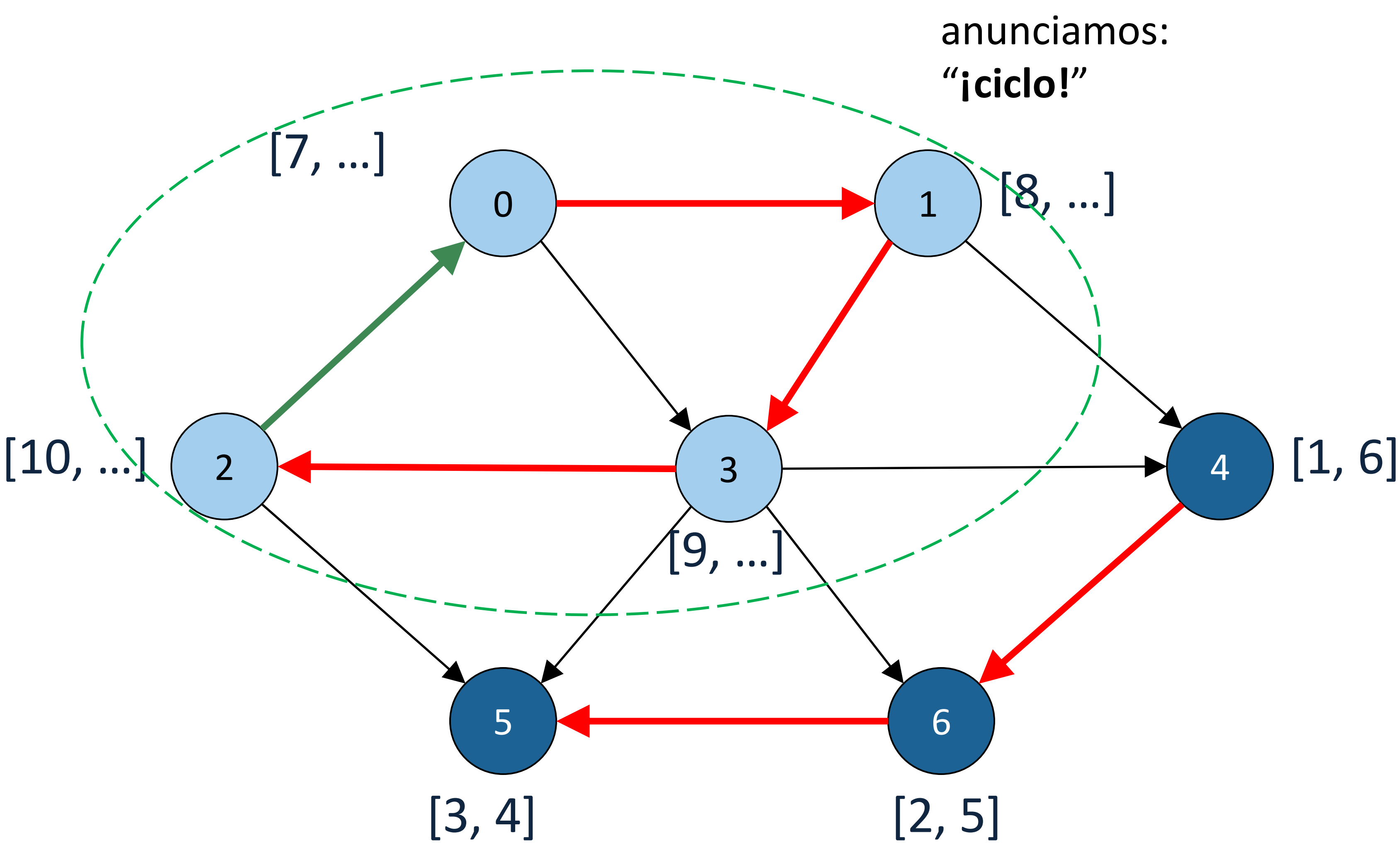
dfsVisit de *G* a partir del vértice 0



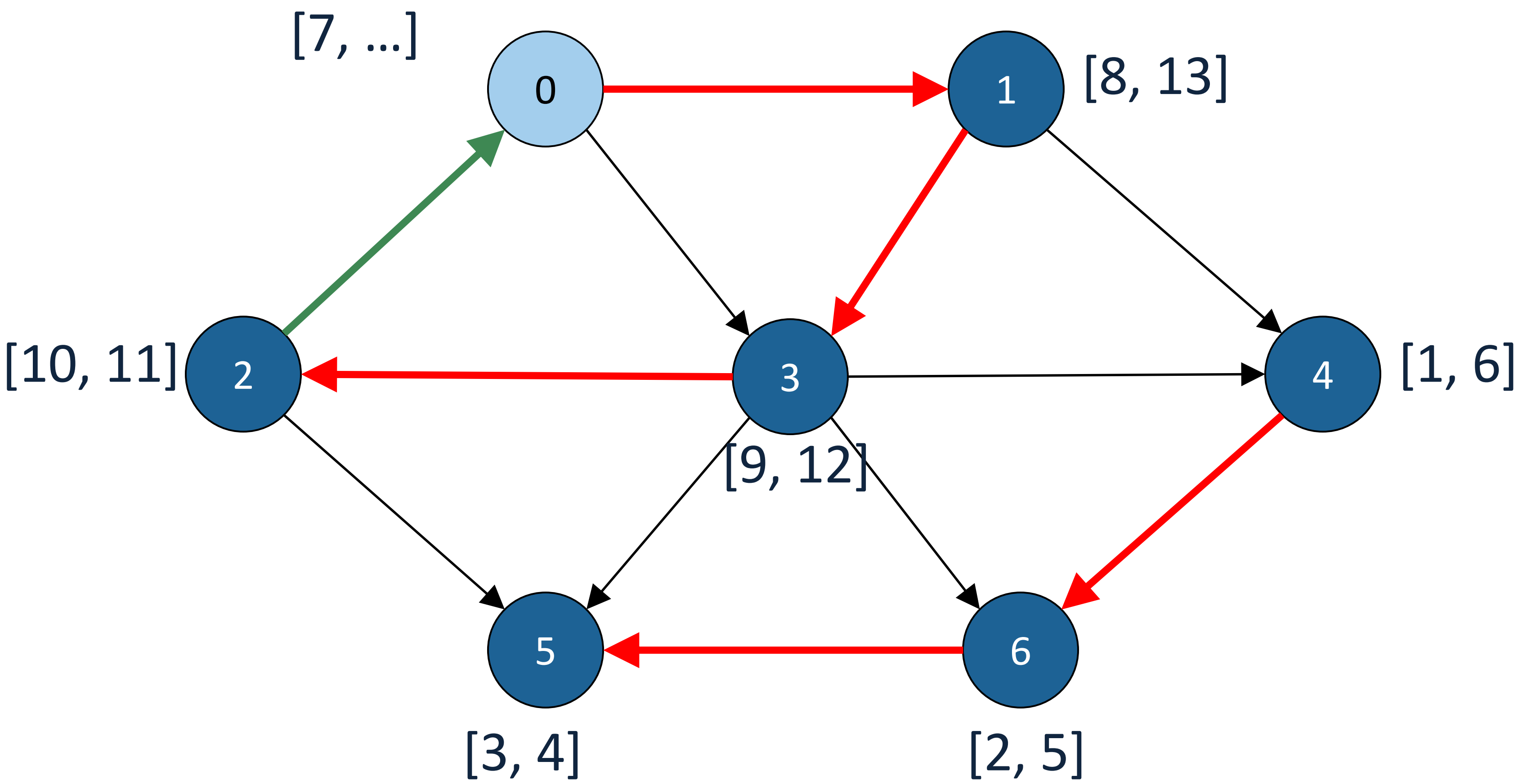
dfsVisit de *G*: de 0 vamos a 1, de ahí a 3 y de ahí a 2



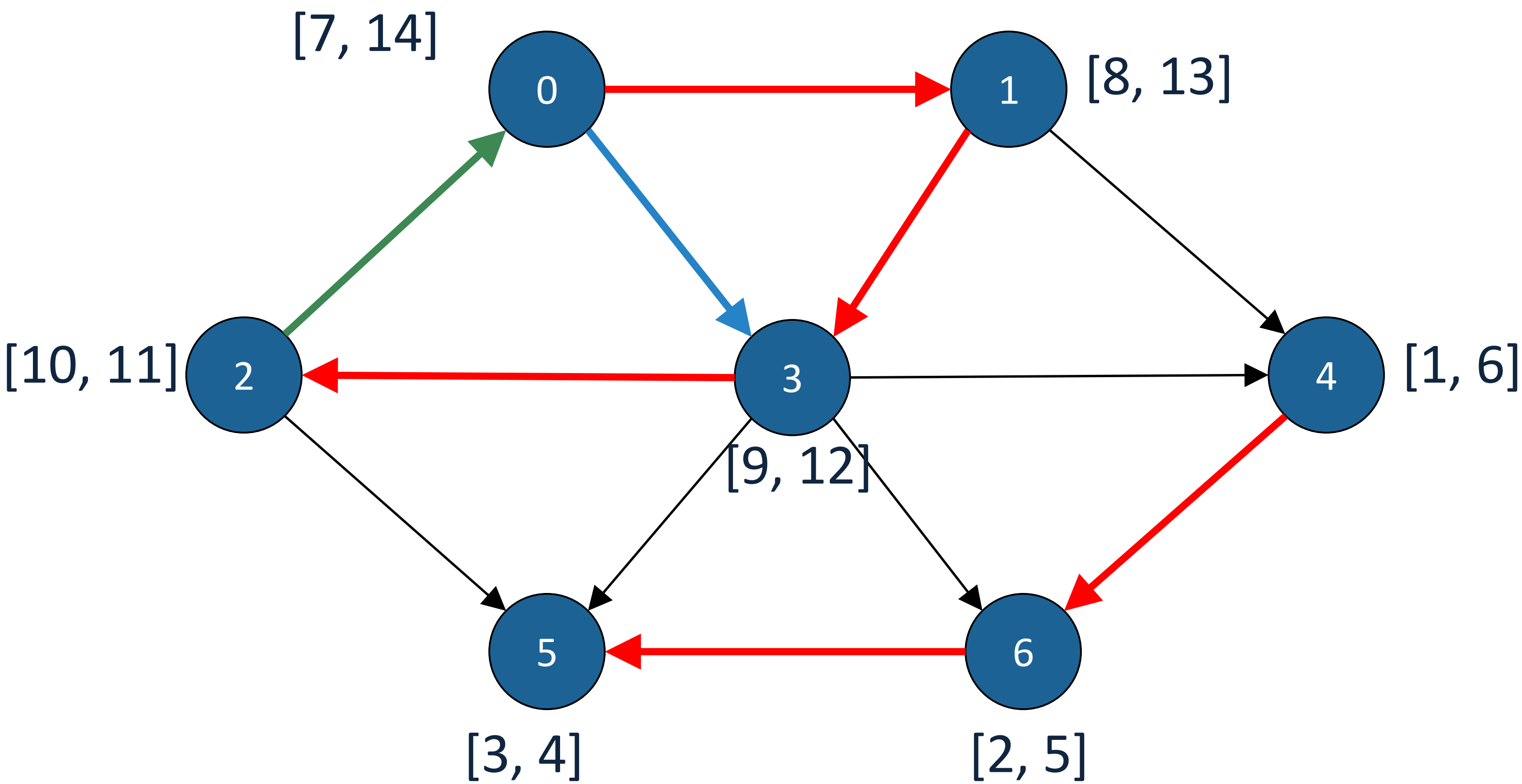
De 2 **no** vamos a 5 ni a 0; la diferencia es que 5 ya está terminado, pero 0 aún no



Terminamos 2, volvemos a 3 y (con 4, 5 y 6 terminados) terminamos 3 y luego 1



Volvemos a 0, **no** vamos a 3 → terminamos *dfsVisit* de *G* desde 0



```
dfs(V, E):
    time = 1
    for each u in V:
        u.color = white
    for each u in V:
        if u.color == white:
            time = dfsVisit(u, time)
```

```
dfsVisit(u, time):
    u.color = gray
    u.start = time
    time += 1
    for each v in α[u]:
        if v.color == white:
            time = dfsVisit(v, time)
    u.color = black
    u.end = time
    time += 1
    return time
```

Propiedades de los intervalos $[u.start, u.end]$

Dados dos vértices *u* y *v,* sus intervalos cumplen una de las siguientes relaciones:

- [*u.start*, *u.end*] y [*v.start*, *v.end*] son *disjuntos*, y ni *u* ni *v* es descendiente del otro en el bosque DFS
- [*u.start*, *u.end*] *está contenido* en el intervalo [*v.start*, *v.end*], y *u es descendiente de v* en un árbol DFS
- [*v.start*, *v.end*] *está contenido* en el intervalo [*u.start*, *u.end*], y *v es descendiente de u* en un árbol DFS

Tipos de aristas luego de DFS

Aristas de árbol: la arista (u, v) es una arista de árbol si v fue descubierto por primera vez al transitar (u, v)

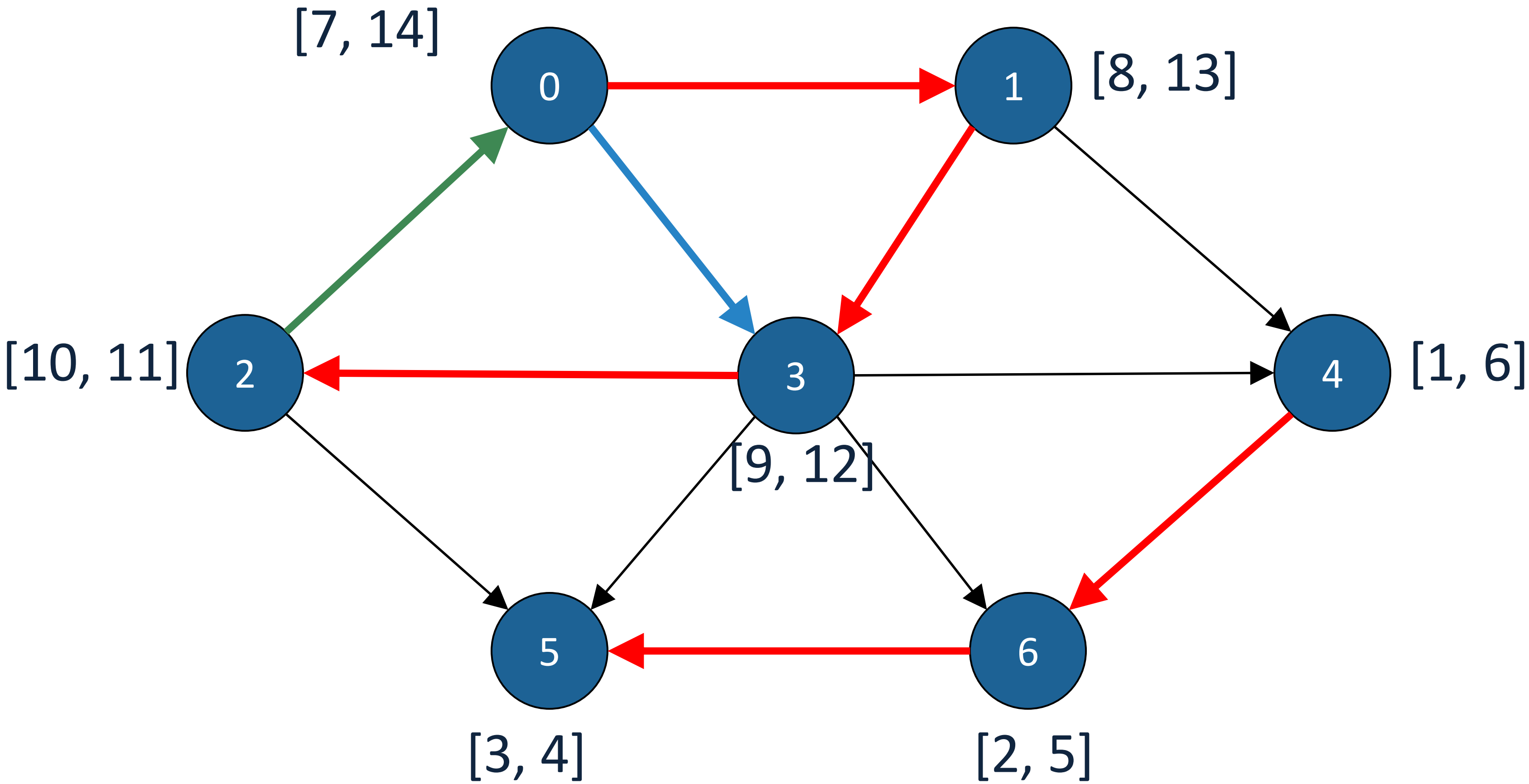
Aristas hacia atrás: aristas (u, v) que conectan un nodo u a un ancestro v en un árbol DFS:

- **el grafo es acíclico si y solo si DFS no produce aristas hacia atrás**

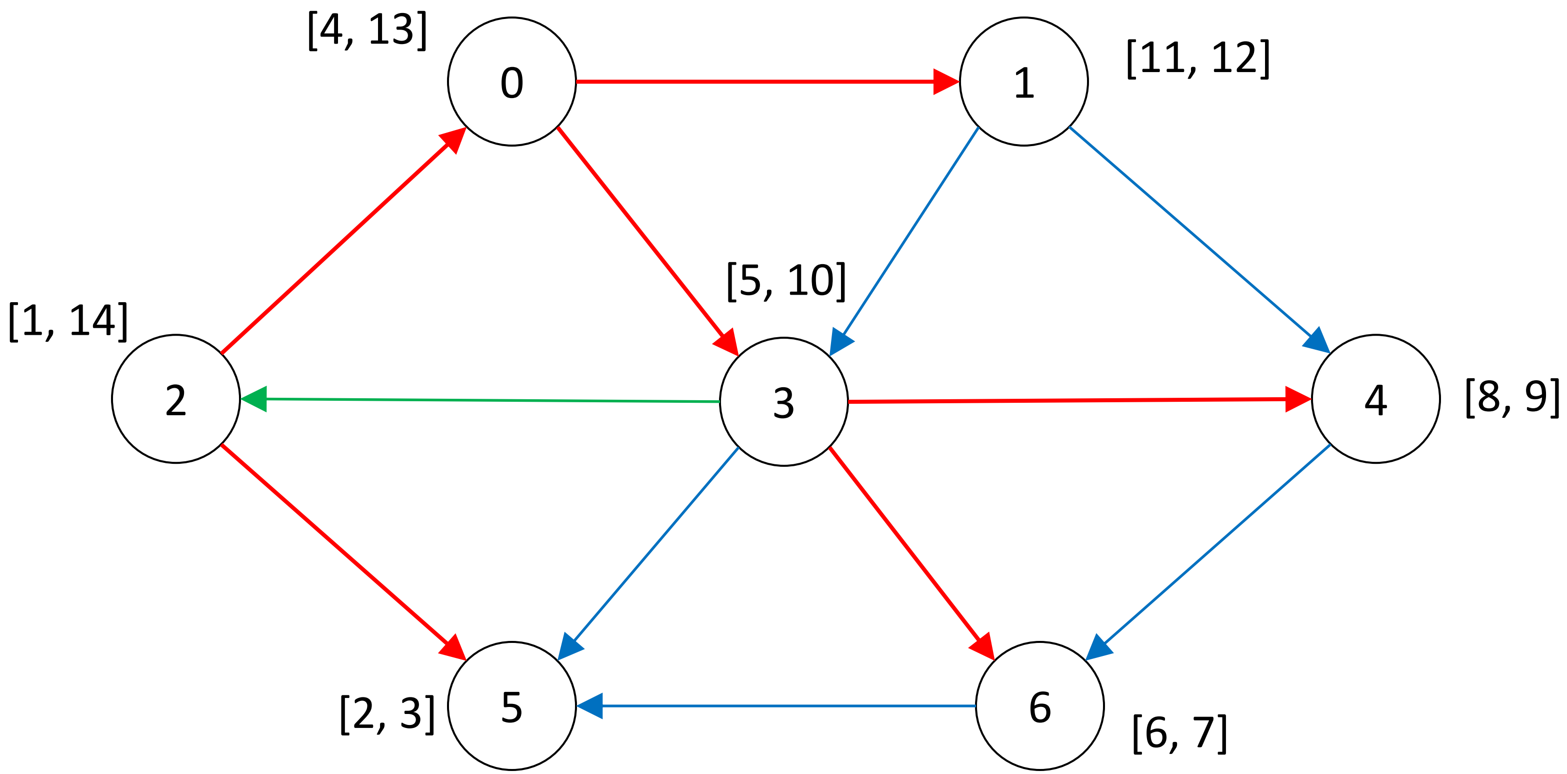
Aristas hacia adelante: aristas (u, v) que no son de árbol y conectan un nodo u a un descendiente v en un árbol DFS; *no aparecen en grafos no direccionales*

Aristas cruzadas: todas las otras aristas; *no aparecen en grafos no direccionales*

Las propiedades de los intervalos de tiempo y los tipos de arista



Ej.: *dfs* de *G* a partir del nodo 2



¿Qué usos le podemos dar a DFS
+ los tiempos de (inicio y) finalización?

En grafos acíclicos: ordenación topológica

En grafos con ciclos: componentes fuertemente conectadas

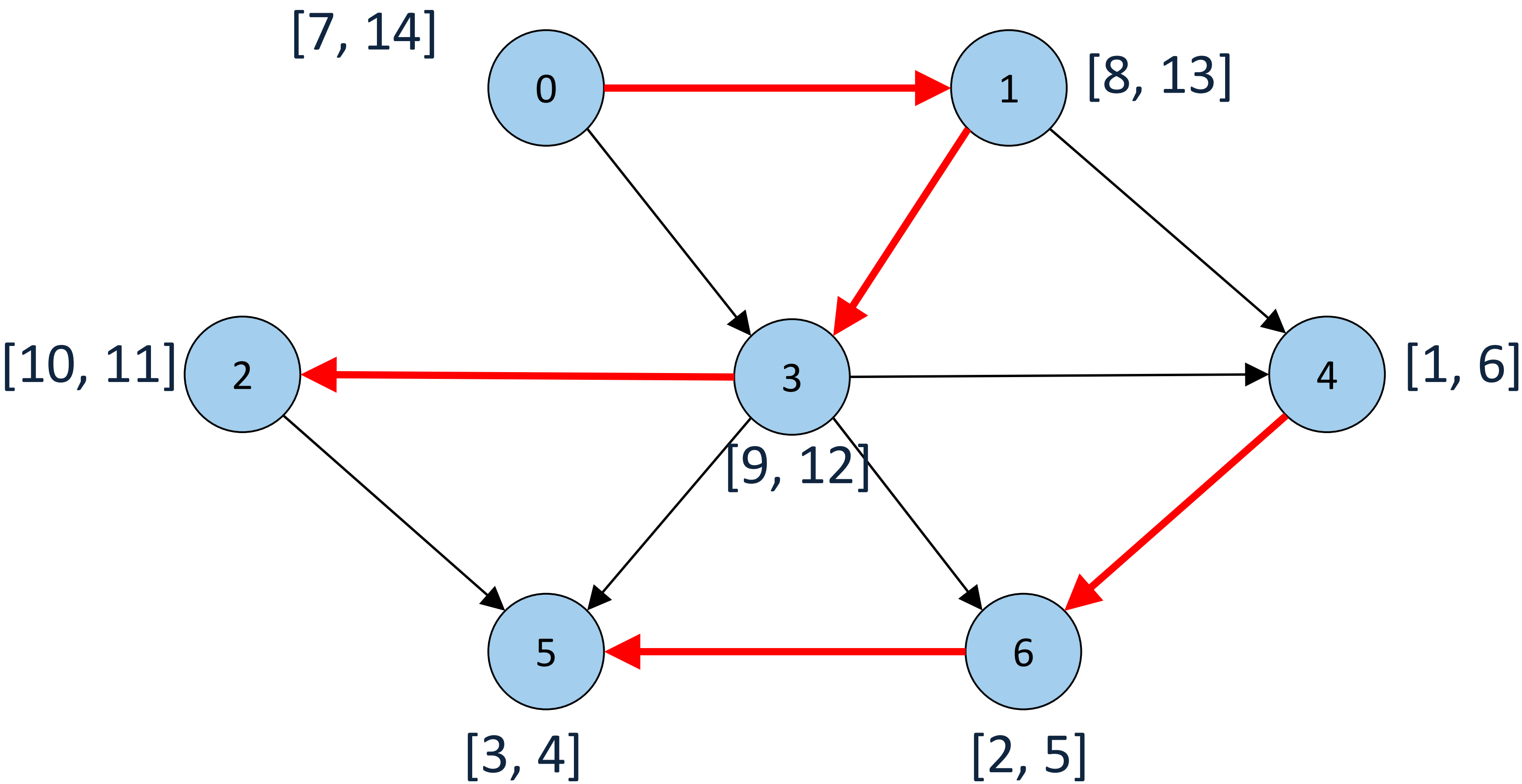
Un grafo direccional acíclico G se puede *ordenar topológicamente*

La **ordenación topológica** de G es una ordenación lineal de todos los nodos

… tal que si G contiene la arista direccional (u, v), entonces u aparece antes que v en la ordenación

Si G tiene ciclos, entonces no existe un orden topológico de G

Ej.: grafo después de ejecutar DFS



Lista *L*: 0 1 3 2 4 6 5

El algoritmo de ordenación topológica

$\boldsymbol{topSort}(\boldsymbol{G})$

Crear lista $\boldsymbol{L}$ vacía

Ejecutar $\boldsymbol{dfs}(\boldsymbol{G})$ con tiempos

Insertar nodos en $\boldsymbol{L}$ en orden descendiente de tiempos $\boldsymbol{end}$

$\boldsymbol{return}\ \boldsymbol{L}$

El algoritmo de ordenación topológica

$\boldsymbol{topSort}(\boldsymbol{G})$

Crear lista $\boldsymbol{L}$ vacía

Ejecutar $\boldsymbol{dfs}(\boldsymbol{G})$ con tiempos:

- cada vez que calculamos el tiempo $\boldsymbol{end}$ para un nodo, insertamos ese nodo al frente de $\boldsymbol{L}$

$\boldsymbol{return}\ \boldsymbol{L}$

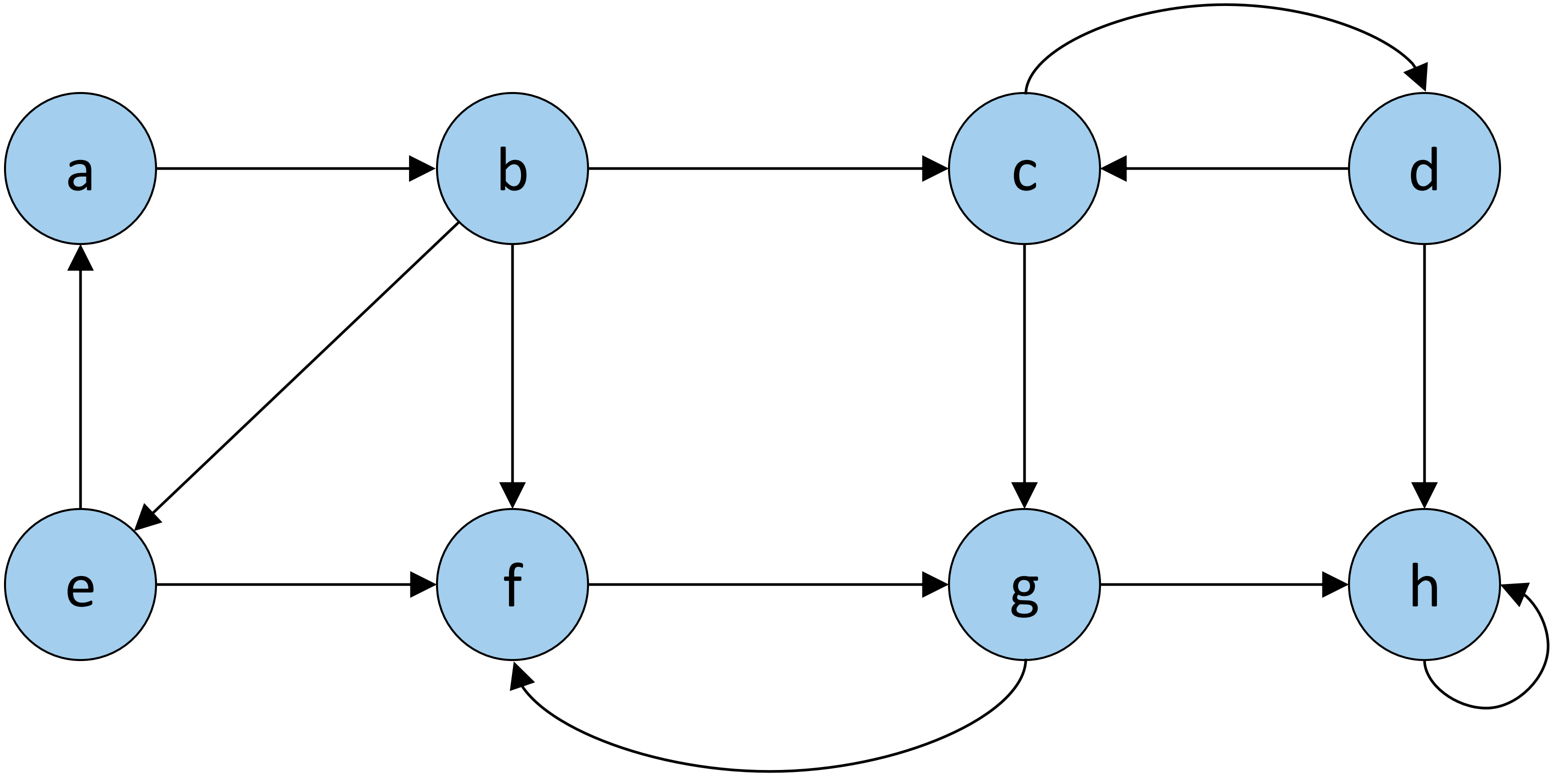
Grafos direccionales con ciclos y sus *componentes fuertemente conectadas*

En un grafo con ciclos no es posible encontrar un orden topológico ya que dos nodos de un ciclo pueden alcanzarse mutuamente

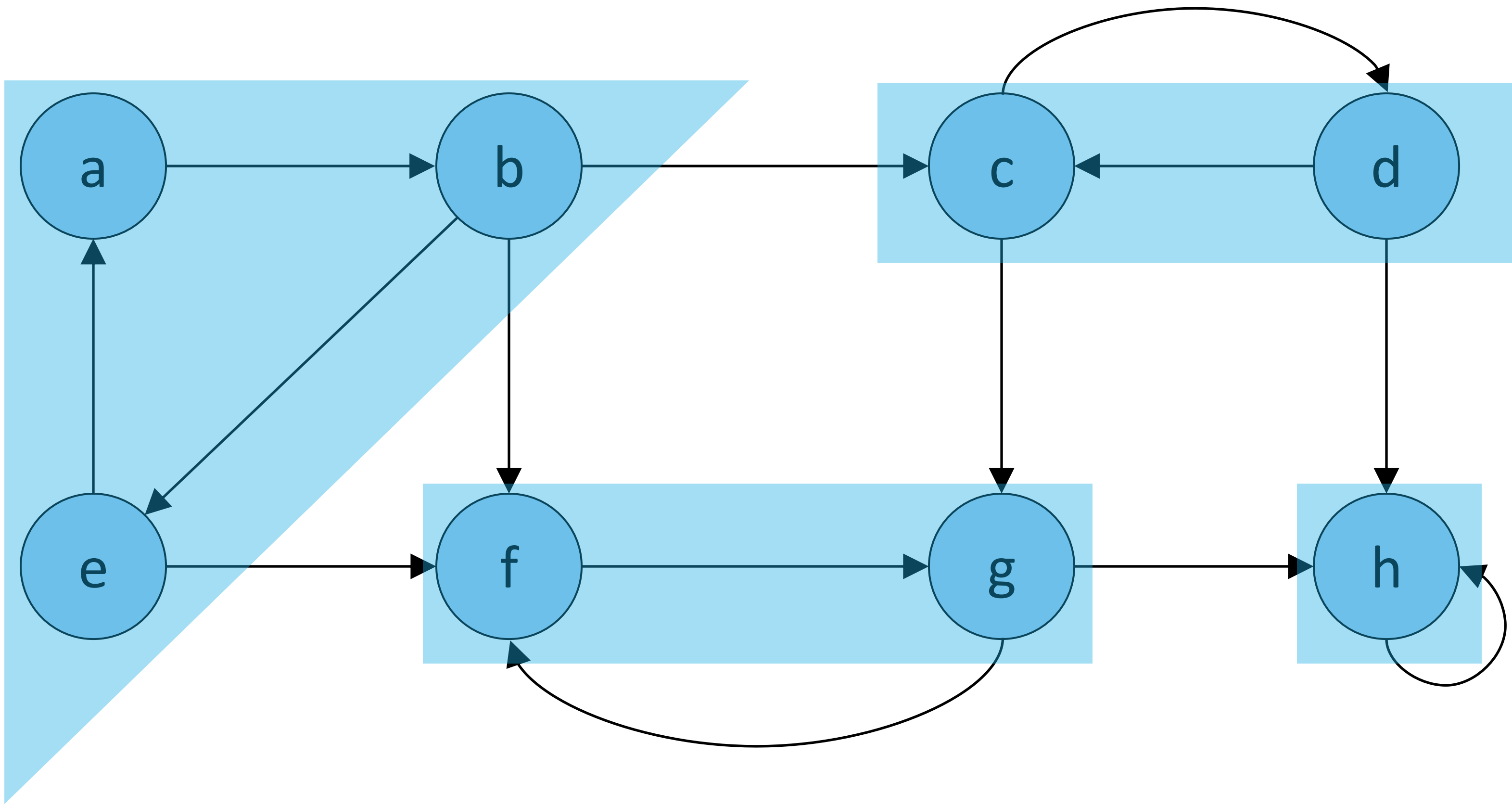
Los nodos de un grafo que se pueden alcanzar mutuamente son miembros de una misma *componente fuertemente conectada* (CFC) del grafo

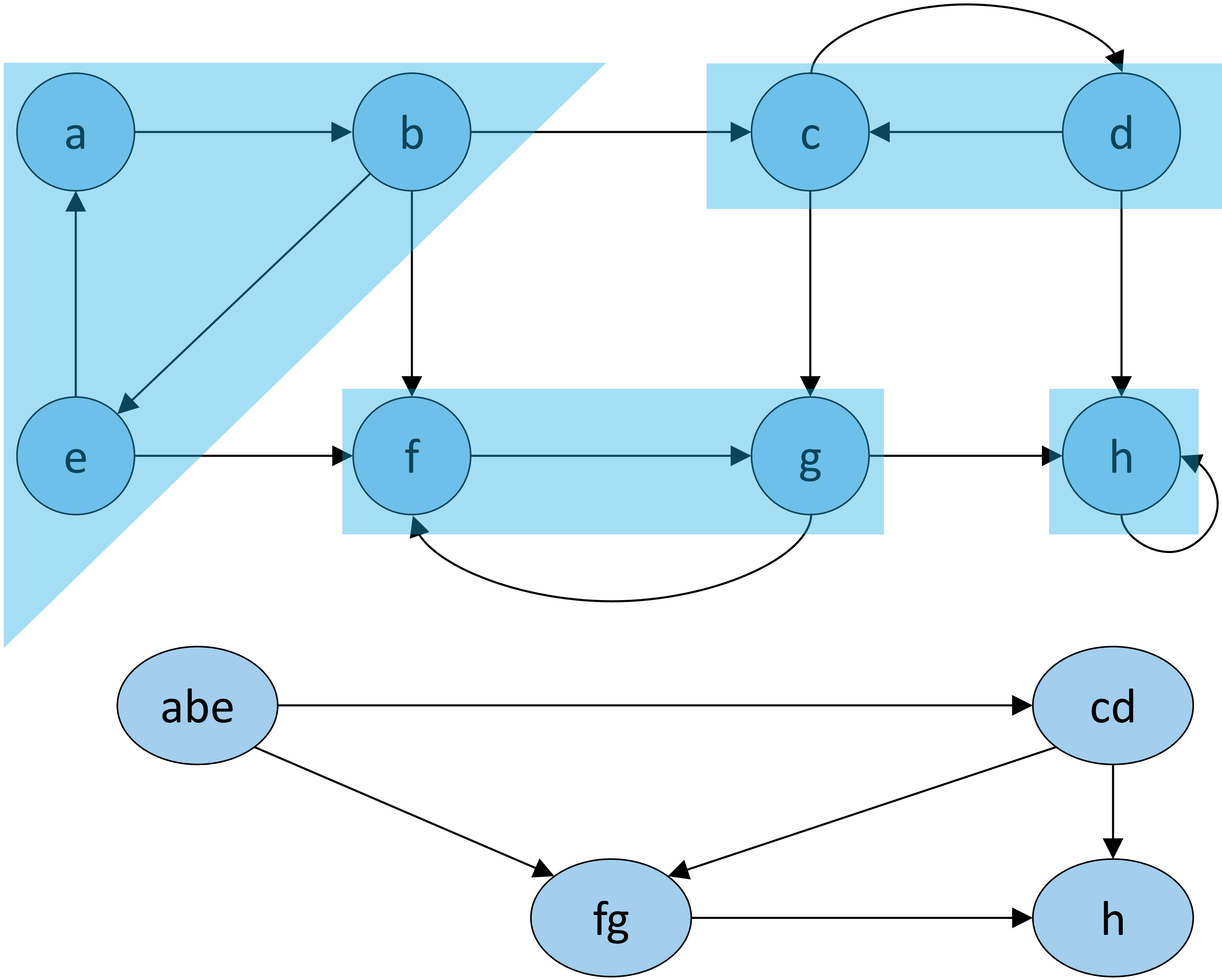
Las CFCs de un grafo direccional G son conjuntos máximos de nodos $C \subseteq V$ tales que para todo par de nodos u y v en C, u y v son mutuamente alcanzables

Ej.: grafo *G* con ciclos



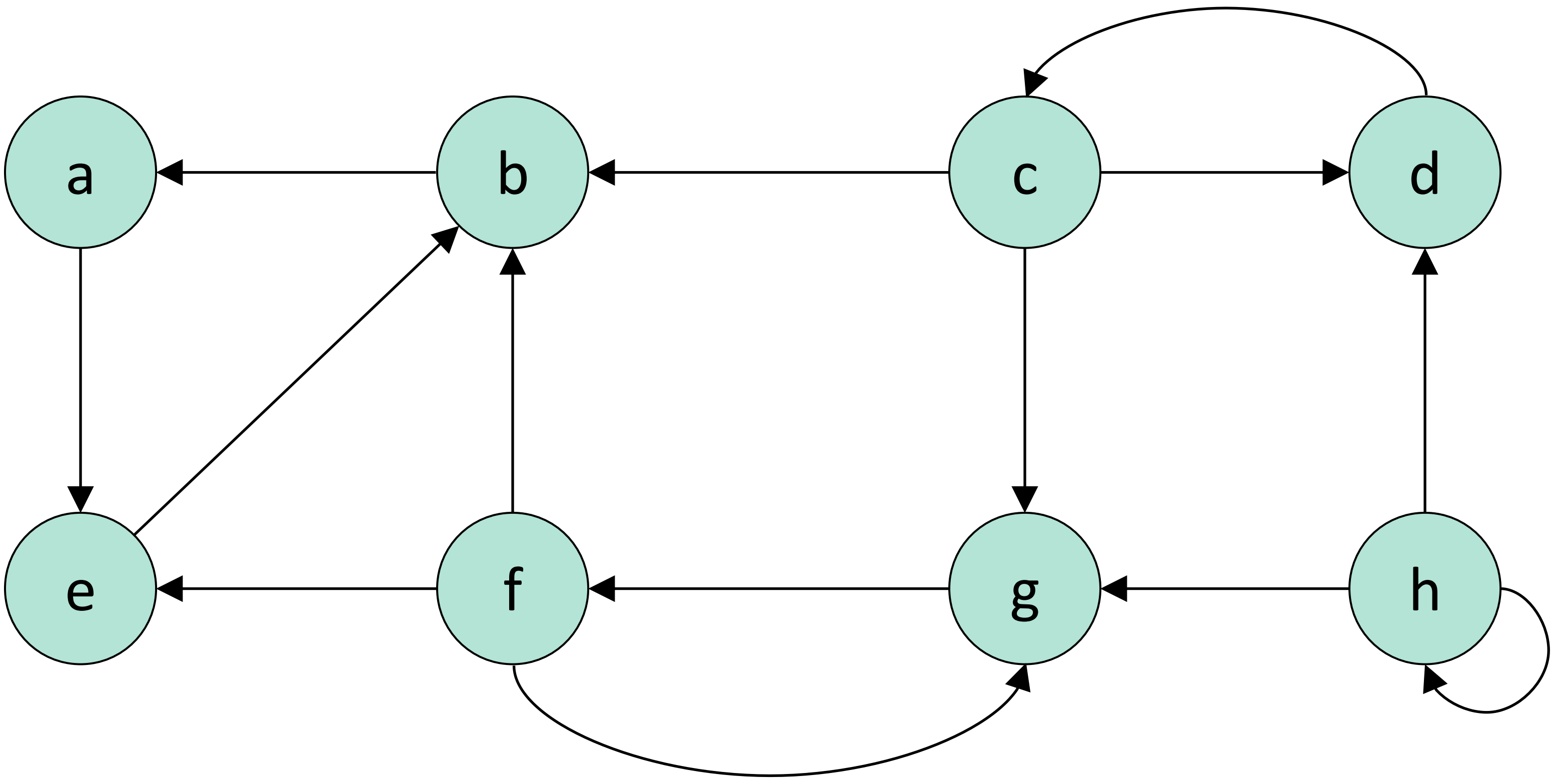
Las CFCs de *G*



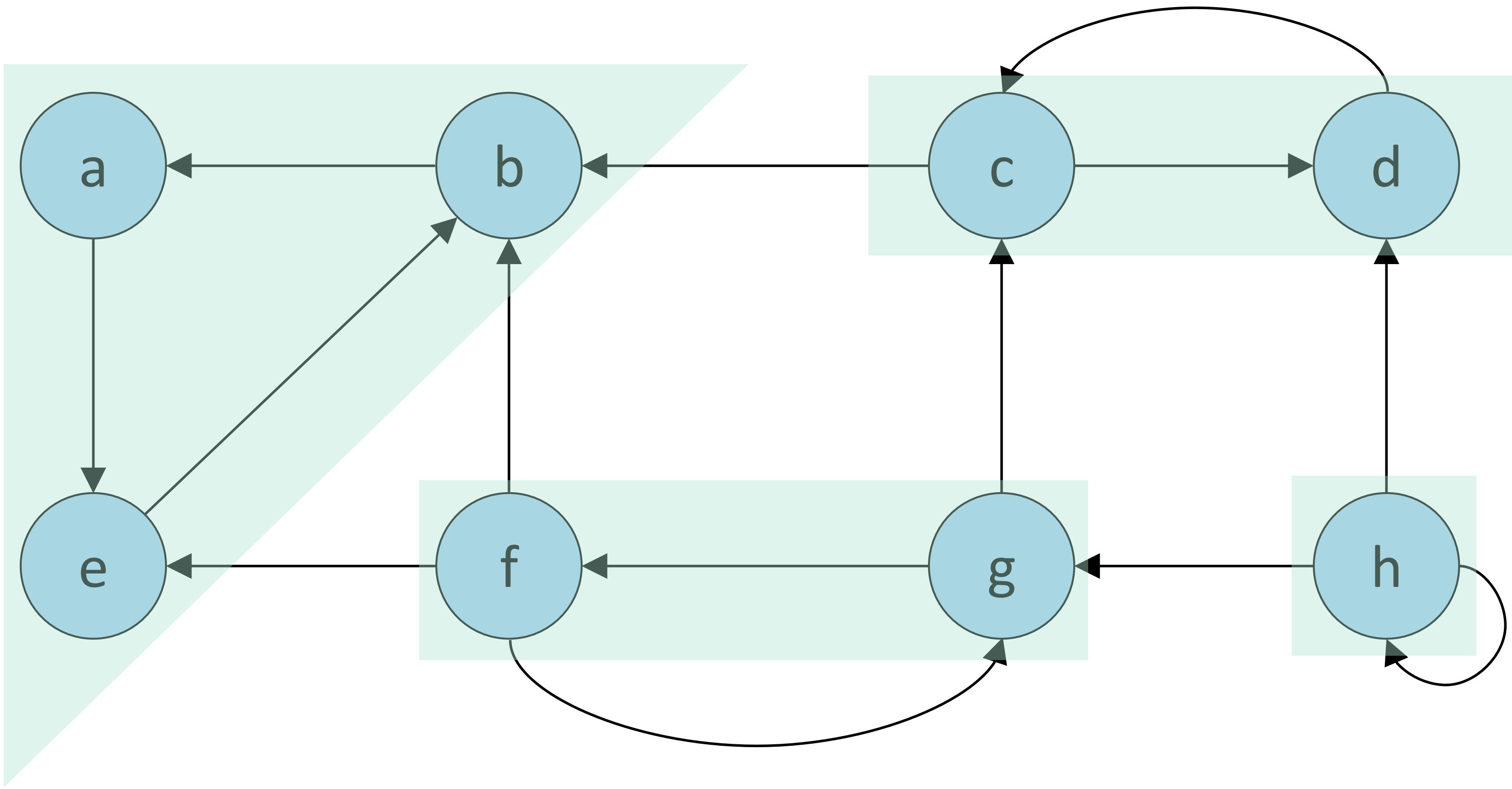

a
b
c
d
e
f
g
h
abe
cd
fg
h

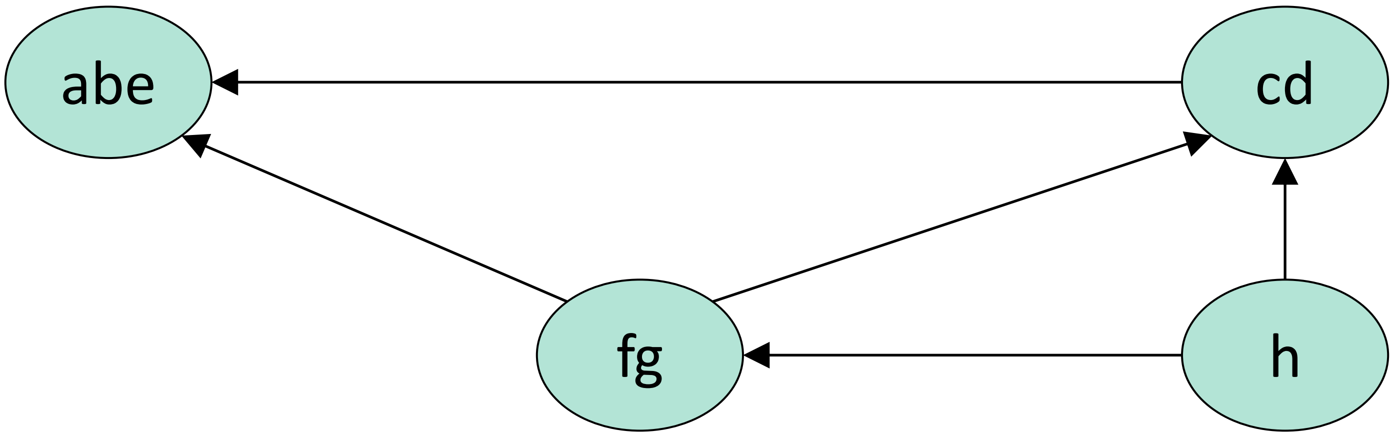
El algoritmo usa el *grafo transpuesto G'* de *G*

G' es *G* pero con la dirección de las aristas invertida: sea $\alpha'[\boldsymbol{u}]$ la lista de aristas de $\boldsymbol{u}$ en *G'*

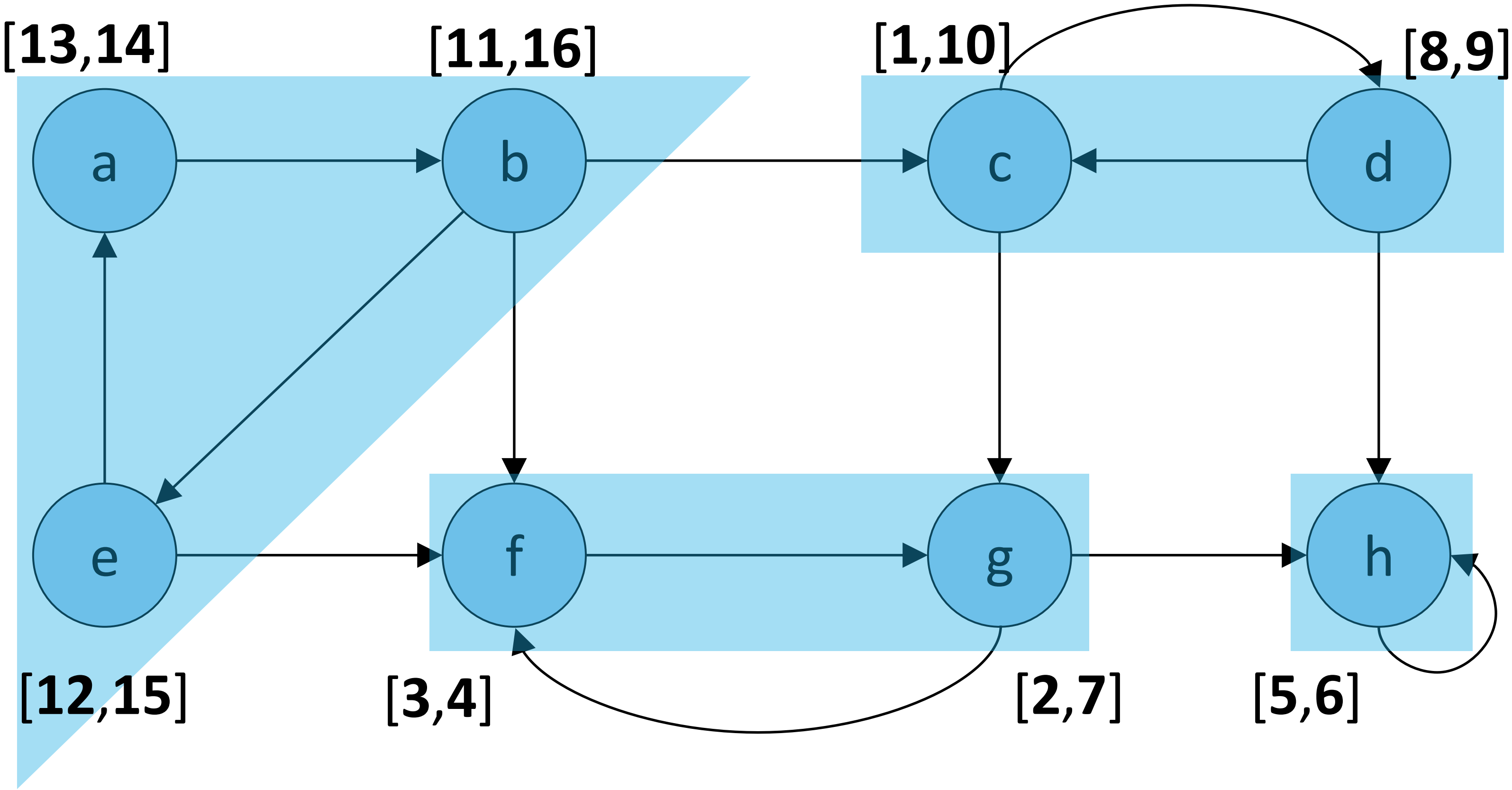


G' tiene las mismas CFCs que *G*

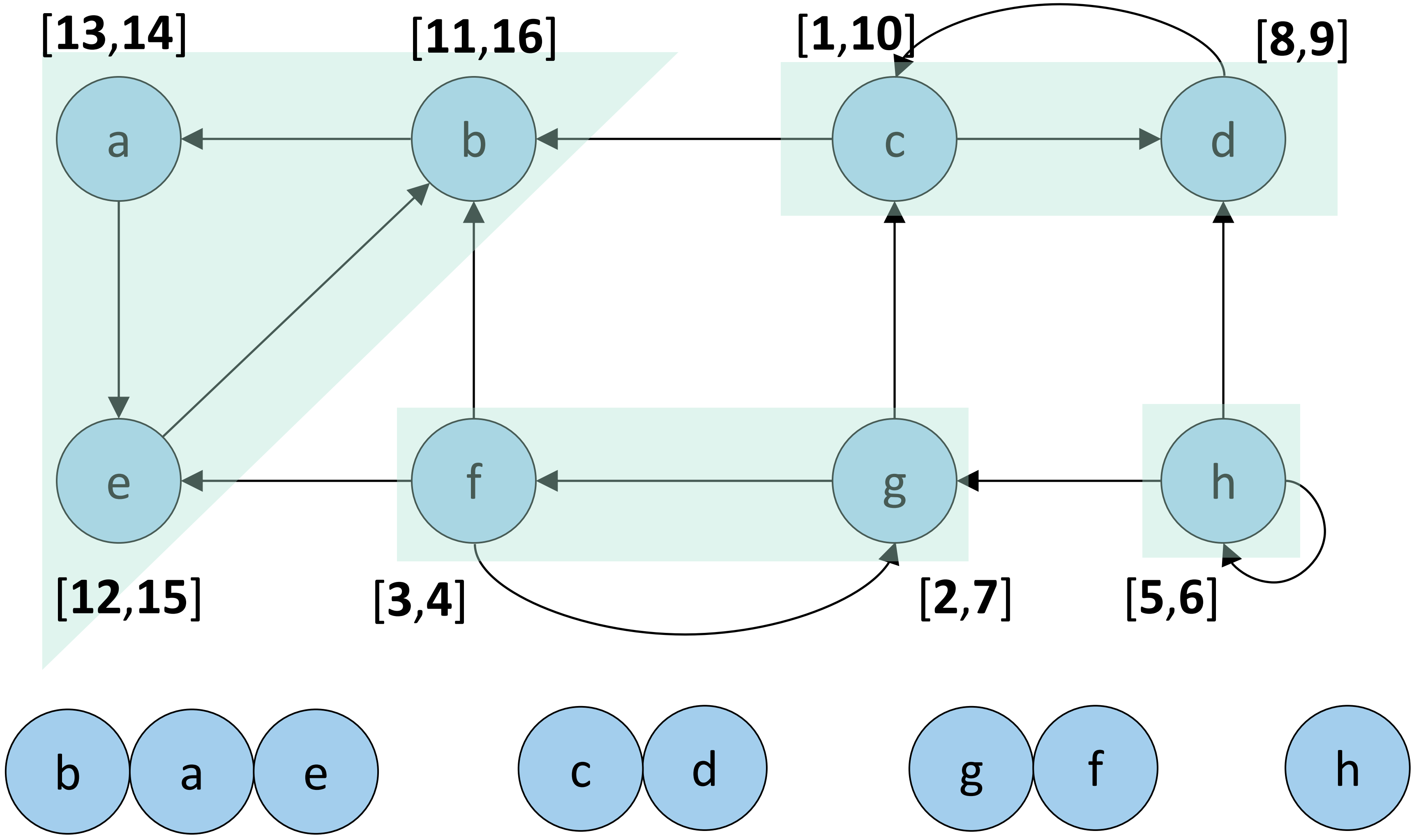



abe
cd
fg
h

Hagamos un recorrido DFS de *G*, anotando los tiempos de finalización de cada nodo



DFS sobre *G'*, pero en orden decreciente de tiempos ***end*** (según el recorrido anterior)



Algoritmo de Kosaraju para CFCs

Cada CFC tiene un nodo ***representante:***

si el representante de dos nodos es el mismo, entonces los nodos pertenecen a la misma CFC

$\boldsymbol{assign}(\boldsymbol{u}, \boldsymbol{rep})$:

$\quad \boldsymbol{if}\ \boldsymbol{u}.\boldsymbol{rep} = \emptyset$:

$\quad\quad \boldsymbol{u}.\boldsymbol{rep} = \boldsymbol{rep}$

$\quad\quad \boldsymbol{foreach}\ \boldsymbol{v}\ \boldsymbol{in}\ \alpha'[\boldsymbol{u}]$:

$\quad\quad\quad \boldsymbol{assign}(\boldsymbol{v}, \boldsymbol{rep})$

Algoritmo de Kosaraju para CFCs

$\boldsymbol{kosaraju}(\boldsymbol{G})$

Crear lista $\boldsymbol{L}$ vacía

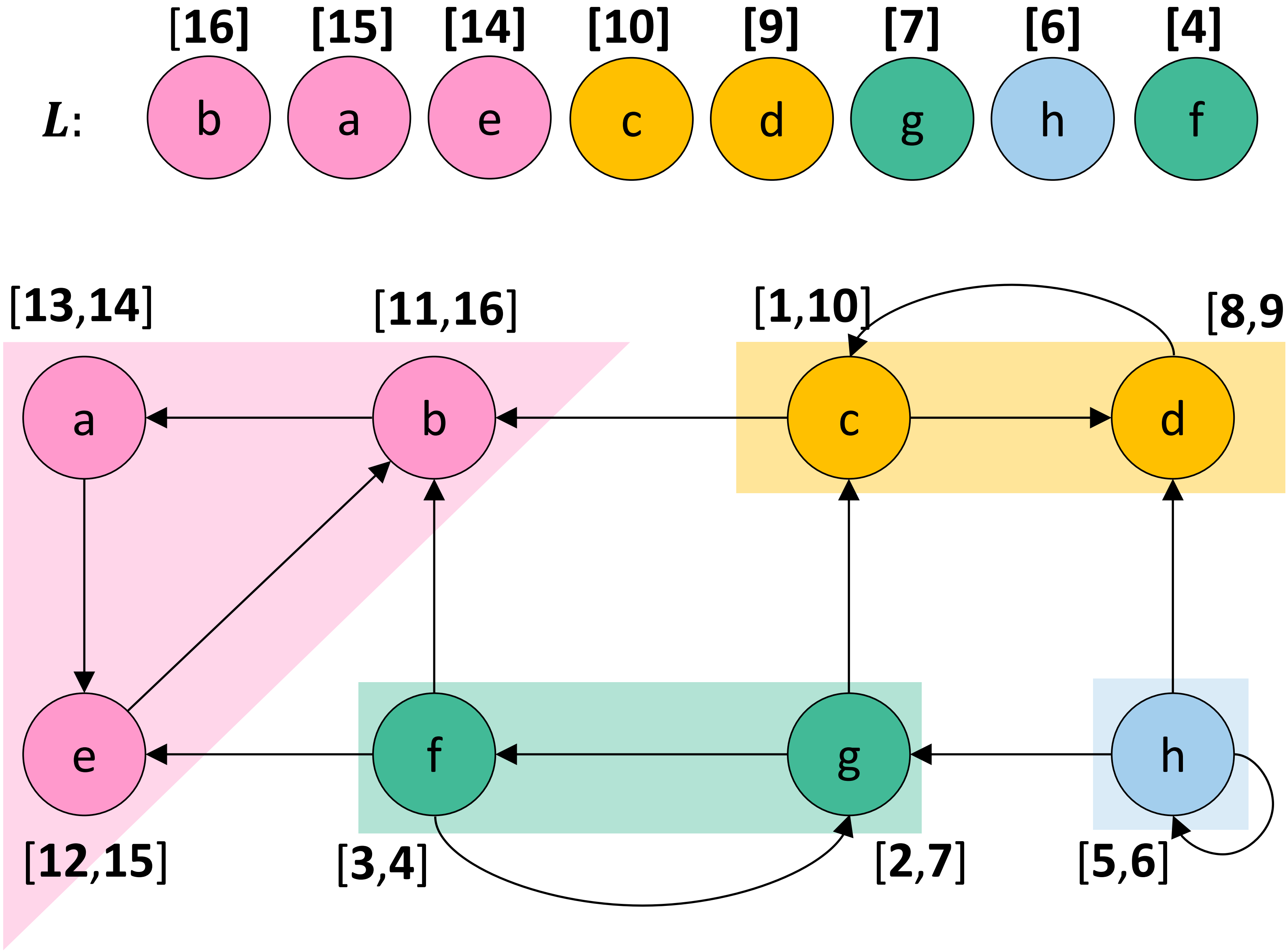
Ejecutar $\boldsymbol{dfs}(\boldsymbol{G})$ con tiempos

Insertar vértices en $\boldsymbol{L}$ en orden descendiente de tiempos $\boldsymbol{end}$

$\boldsymbol{for\ each\ u\ in\ L}$:

$\boldsymbol{assign}(\boldsymbol{u}, \boldsymbol{u})$

Igual que en el orden topológico



Nótese que los nodos de una misma CFC no necesariamente son contiguos en L

Definamos el *grafo de componentes* G^{CFC}

Supongamos que G tiene las componentes fuertemente conectadas C_1, C_2, …, C_k

V^{CFC} es $\{C_1, C_2, \ldots, C_k\}$

Hay una arista $(C_i, C_j) \in E^{CFC}$ si G tiene una arista direccional (x, y) para algún $x \in C_i$ y algún $y \in C_j$

El grafo G^{CFC} tiene un orden topológico

G^{CFC} es un grafo direccional acíclico

Esto, ya que si existiera un ciclo en G^{CFC}, este tendría CFCs, lo cual no es posible por construcción del grafo

Por lo que podemos encontrar un orden topológico en G^{CFC}

Resumen

- Podemos guardar los tiempos de inicio y fin de cada nodo al hacer DFS
- Usando los tiempos podemos encontrar un orden topológico en un grafo acíclico
- En un grafo cíclico podemos encontrar las componentes fuertemente conectadas
- Podemos encontrar el orden topológico de las componentes fuertemente conectadas en un grafo cualquiera