ORDENACIÓN TOPOLÓGICA

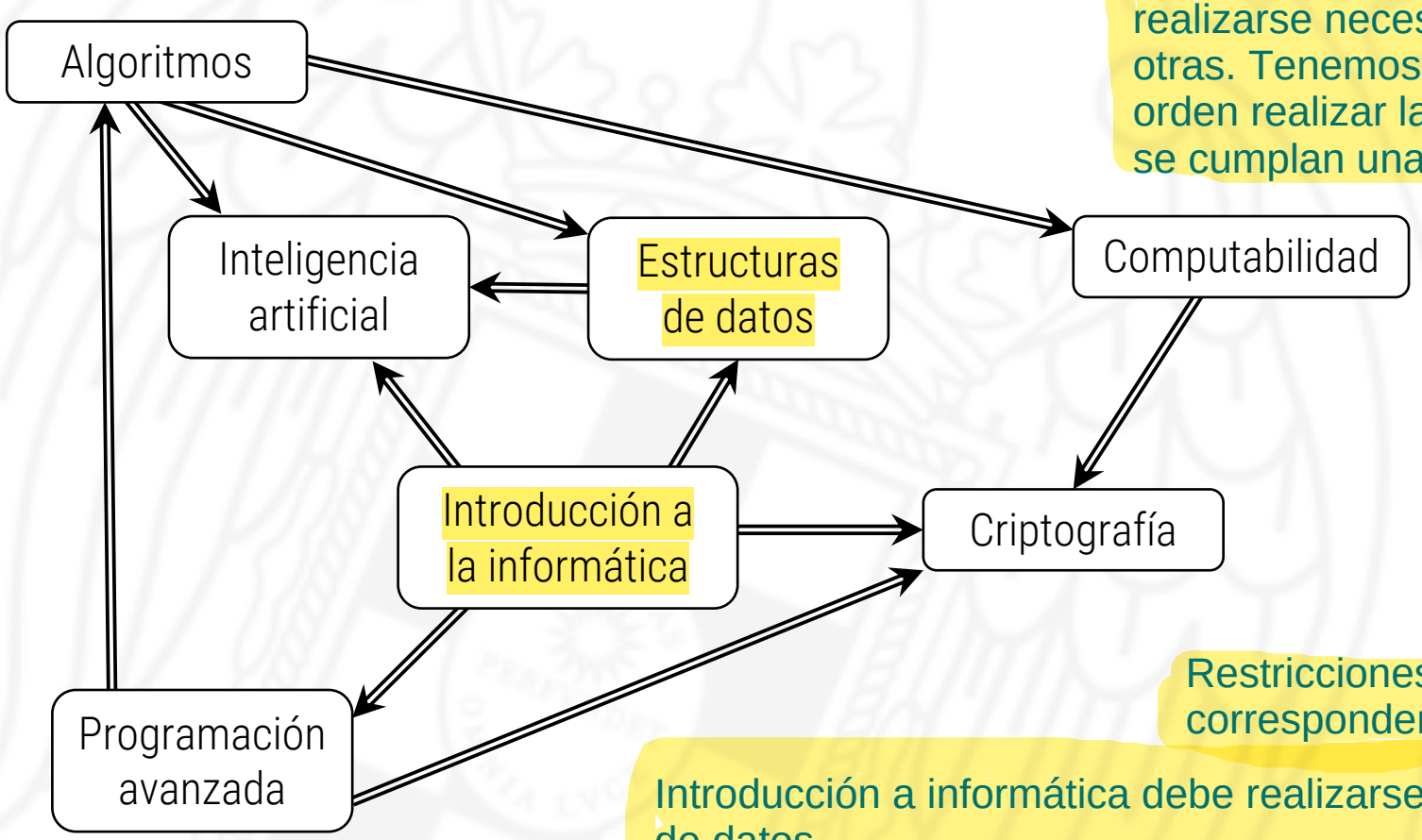
Consiste en ordenar los vértices de tal forma que todas las aristas vayan de un vértice a otro posterior de la secuencia.

UNIVERSIDAD COMPLUTENSE MADRID

ALBERTO VERDEJO

Planificación de tareas con precedencia

- Dado un conjunto de tareas a realizar con restricciones de precedencia, ¿en qué orden deberíamos planificarlas?



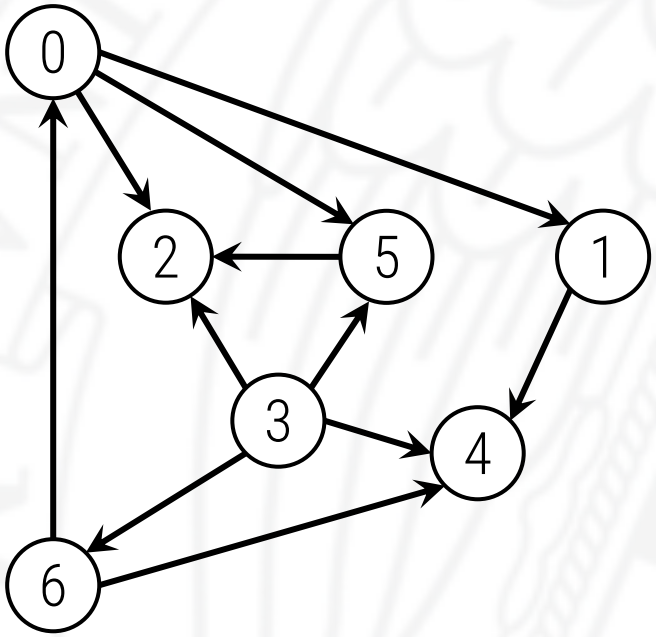
Restricciones de precedencia es = que existen tareas que deben realizarse necesariamente antes que otras. Tenemos que decidir en qué orden realizar las tareas de forma que se cumplan una serie de restricciones.

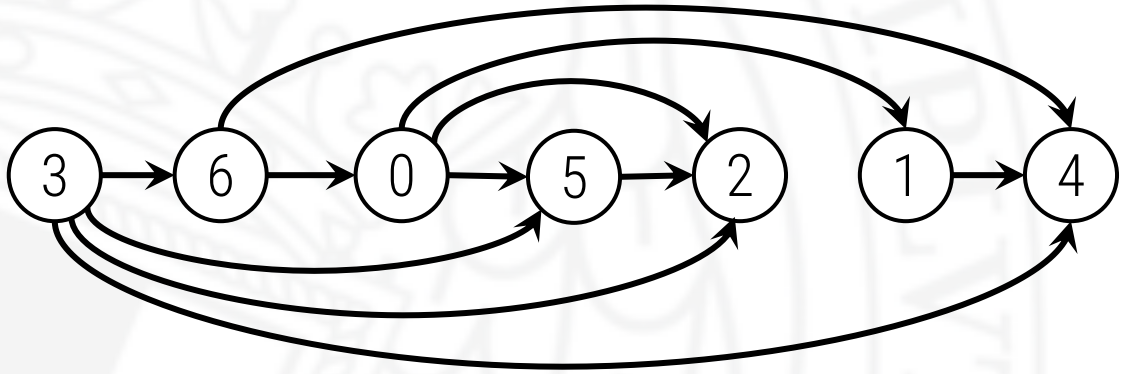
Restricciones de precedencia corresponden a las aristas.

Introducción a informática debe realizarse antes que estructuras de datos.

Ordenación topológica

- Ordenar los vértices de forma que todas las aristas vayan de un vértice a otro posterior en la ordenación.





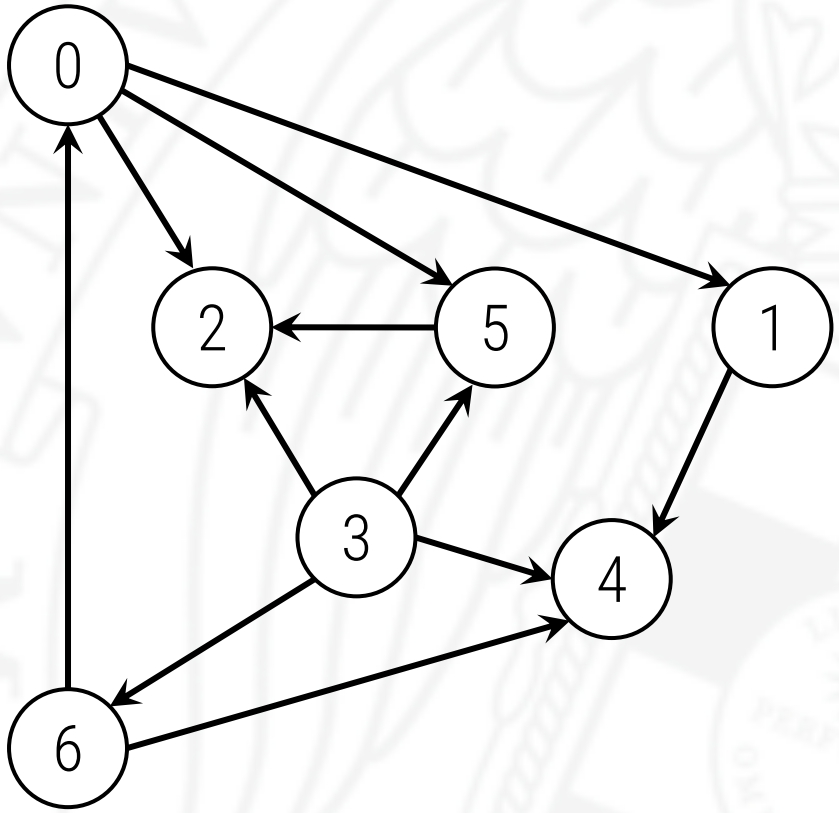
Vamos a asumir que el grafo es acíclico, y detectaremos cuando un grafo dirigido tiene algún ciclo.

- Imprescindible que no haya ciclos (DAG).

DAG = GRAFO DIRIGIDO ACICLICO.

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.



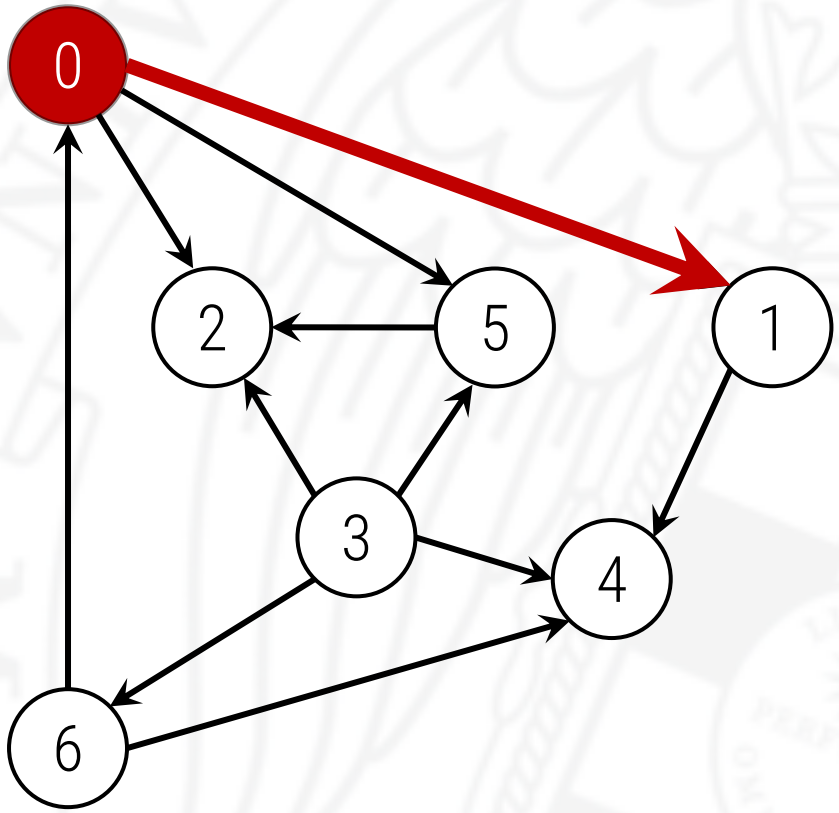
Esto es para el recorrido en profundidad.

Para un grafo podemos definir el recorrido en pre-orden y post-orden.

Preorden = añade vértices a la ordenación según van siendo alcanzados al marcarlos como visitados.
En postorden los añade después de haber hecho las llamadas recursivas con sus hijos.

Postorden

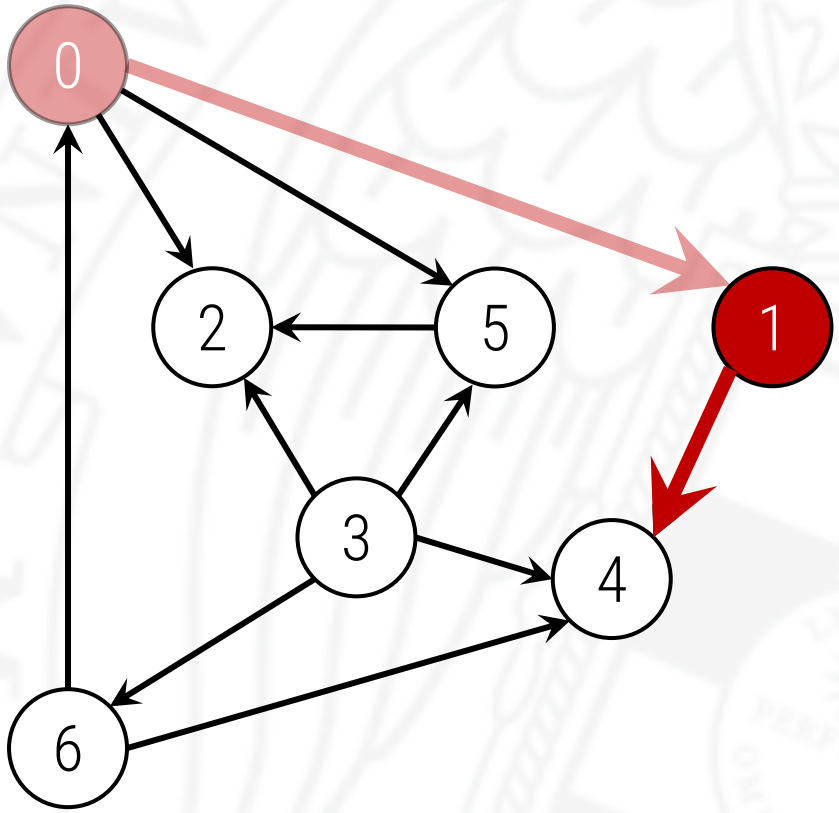
- Añadir el vértice al orden tras las llamadas recursivas.



Postorden

Postorden

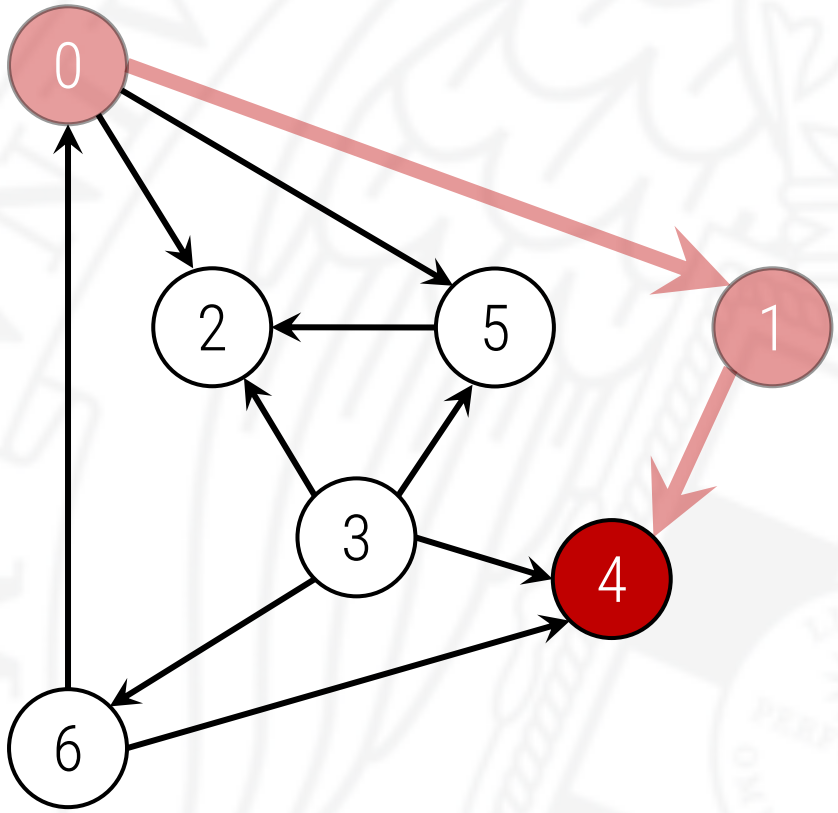
- Añadir el vértice al orden tras las llamadas recursivas.



Postorden

Postorden

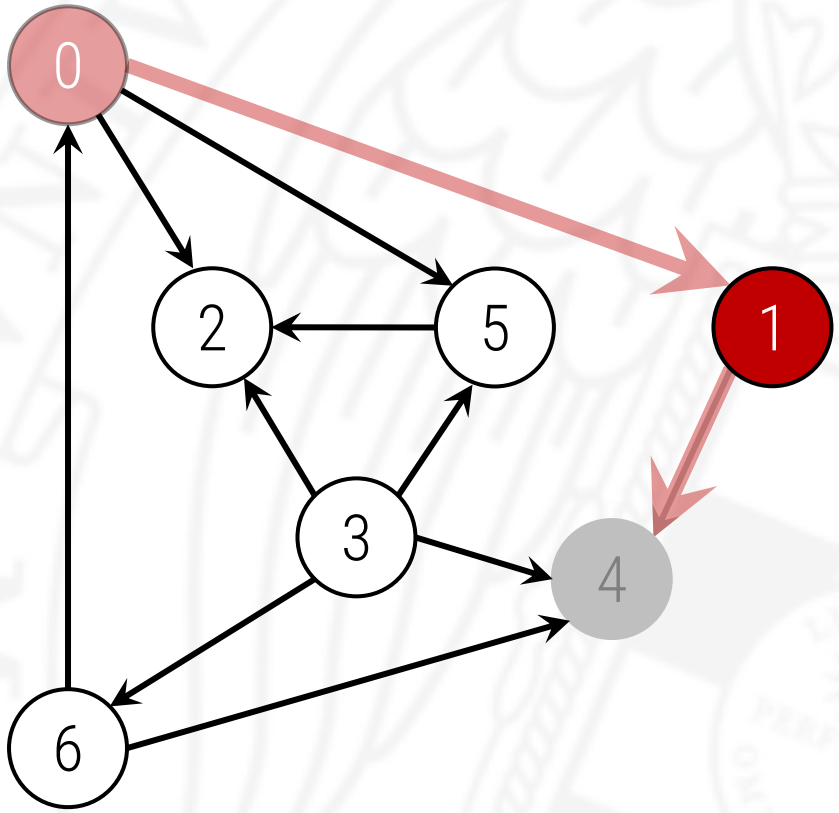
- Añadir el vértice al orden tras las llamadas recursivas.



Postorden

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

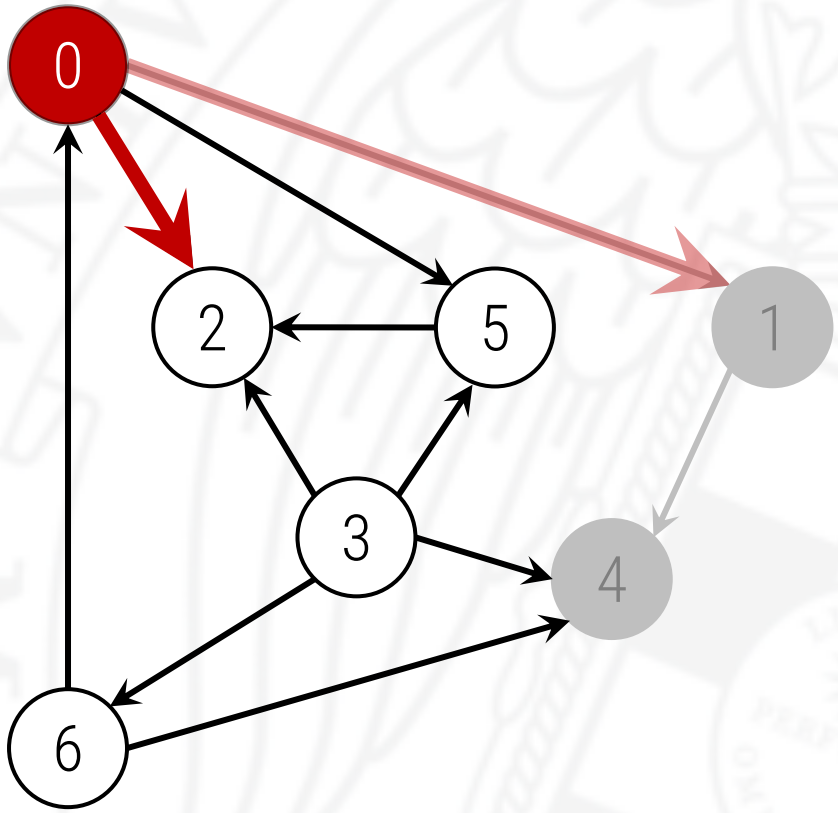


Postorden

4

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

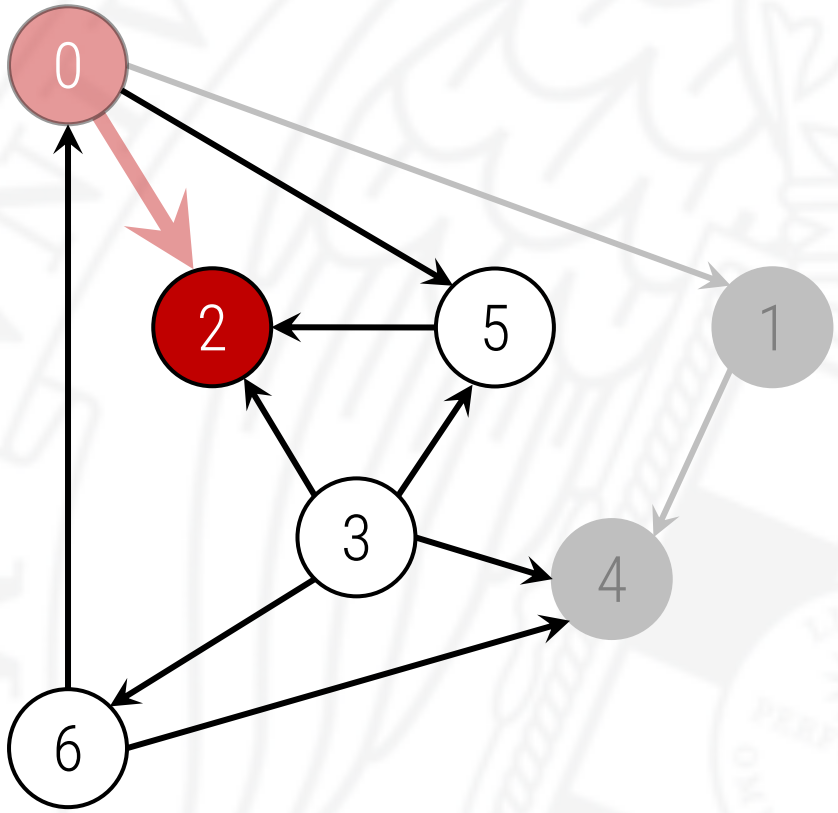


Postorden

4 1

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

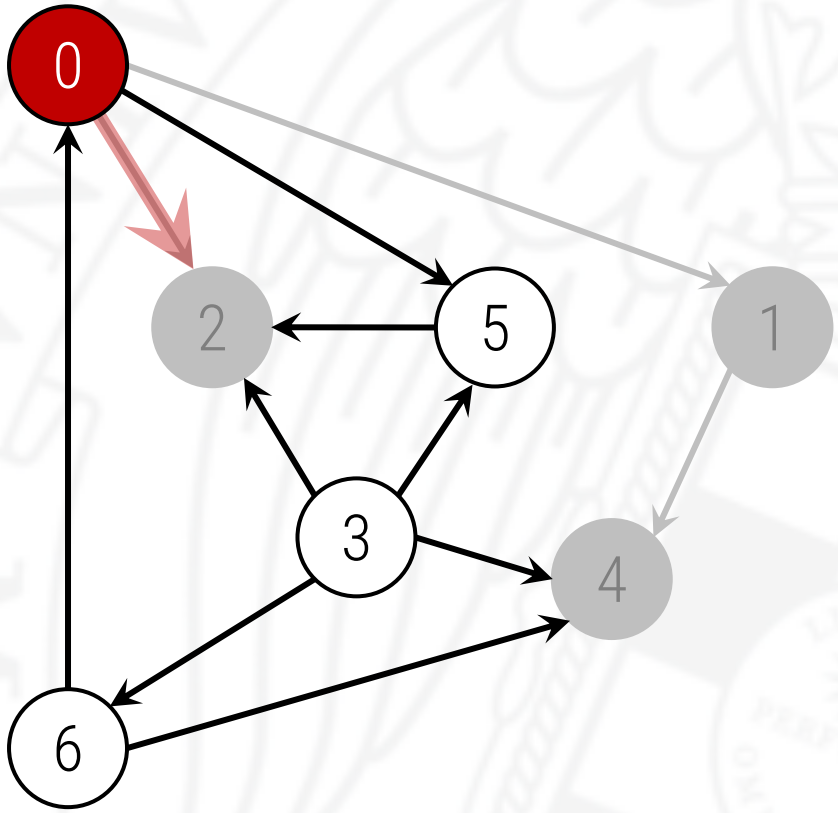


Postorden

4 1

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

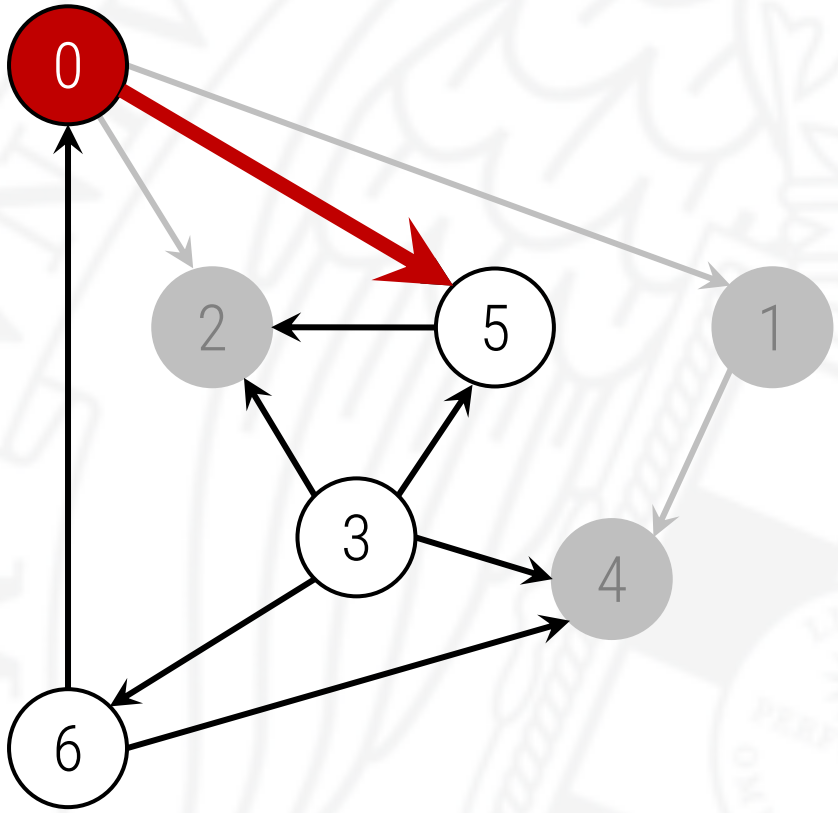


Postorden

4 1 2

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

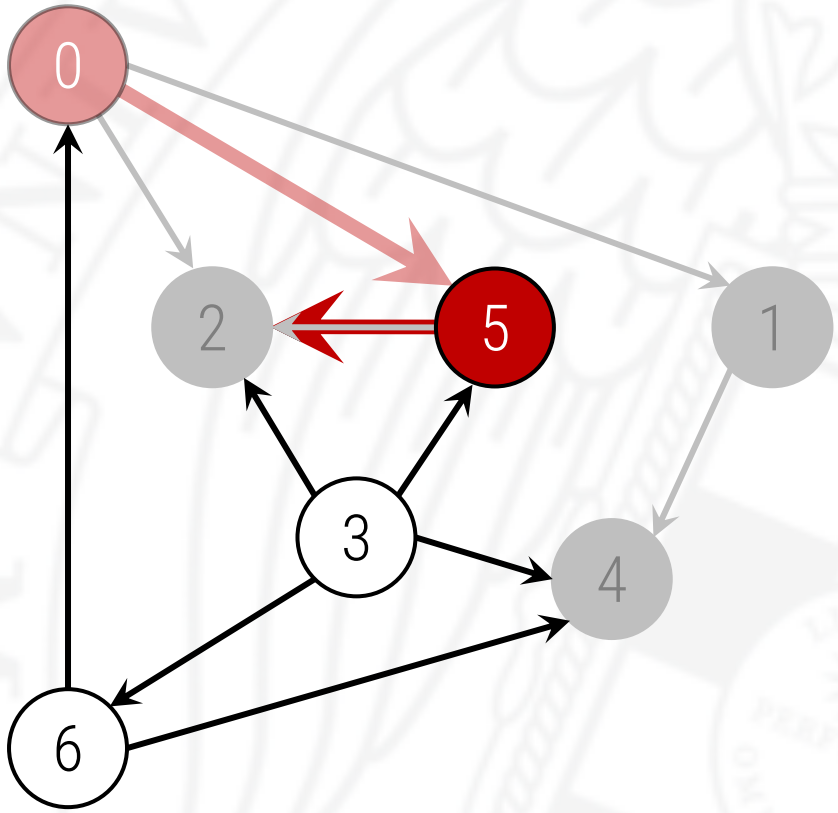


Postorden

4 1 2

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

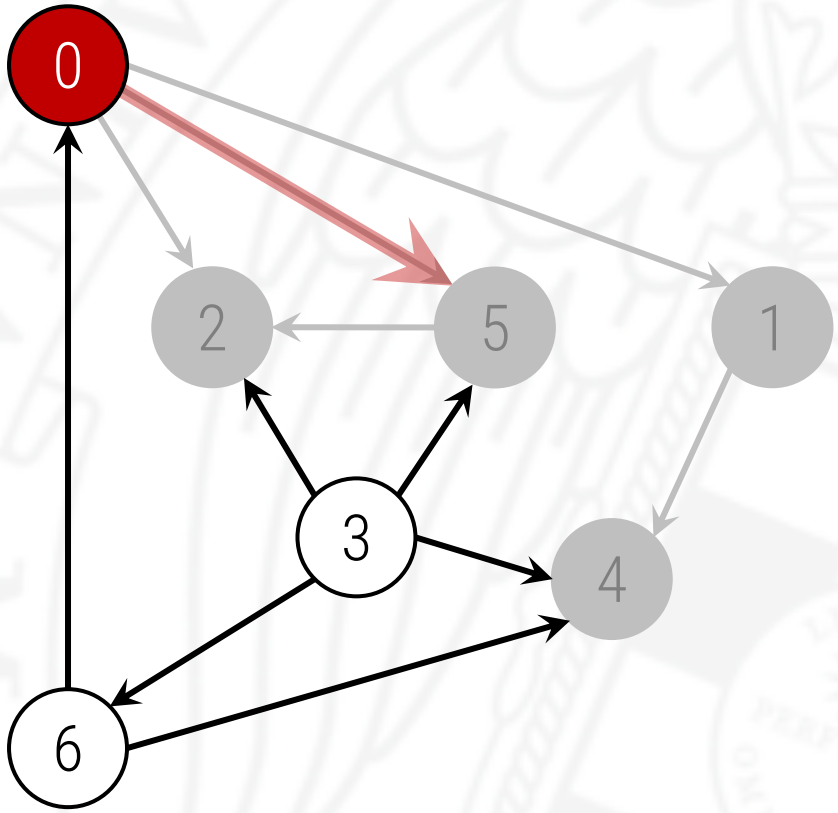


Postorden

4 1 2

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

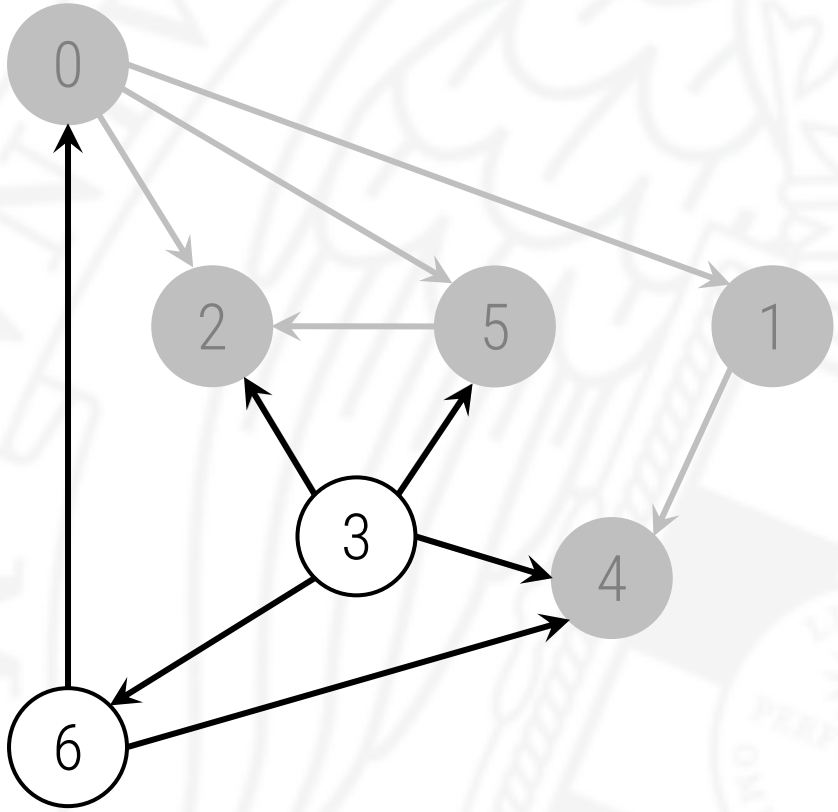


Postorden

4 1 2 5

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.



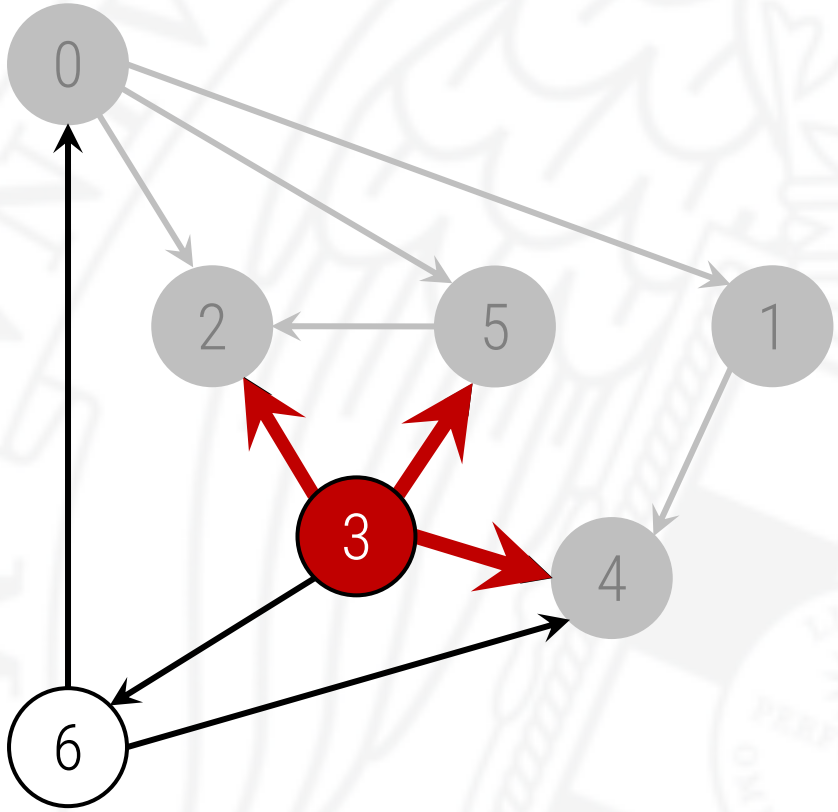
Postorden

4 1 2 5 0

Buscamos el siguiente vértice NO visitado.

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

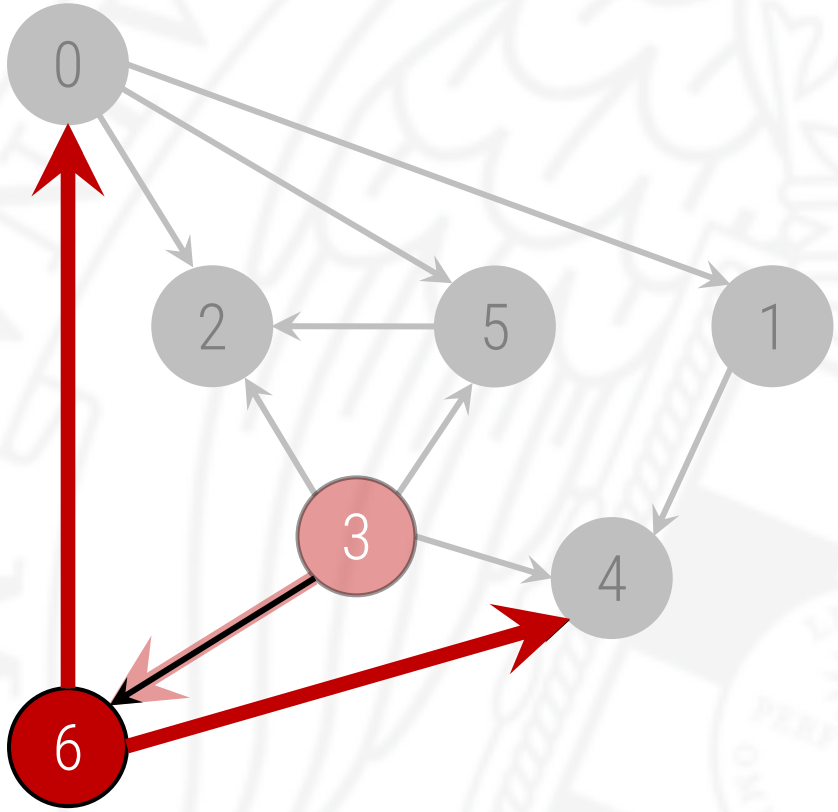


Postorden

4 1 2 5 0

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

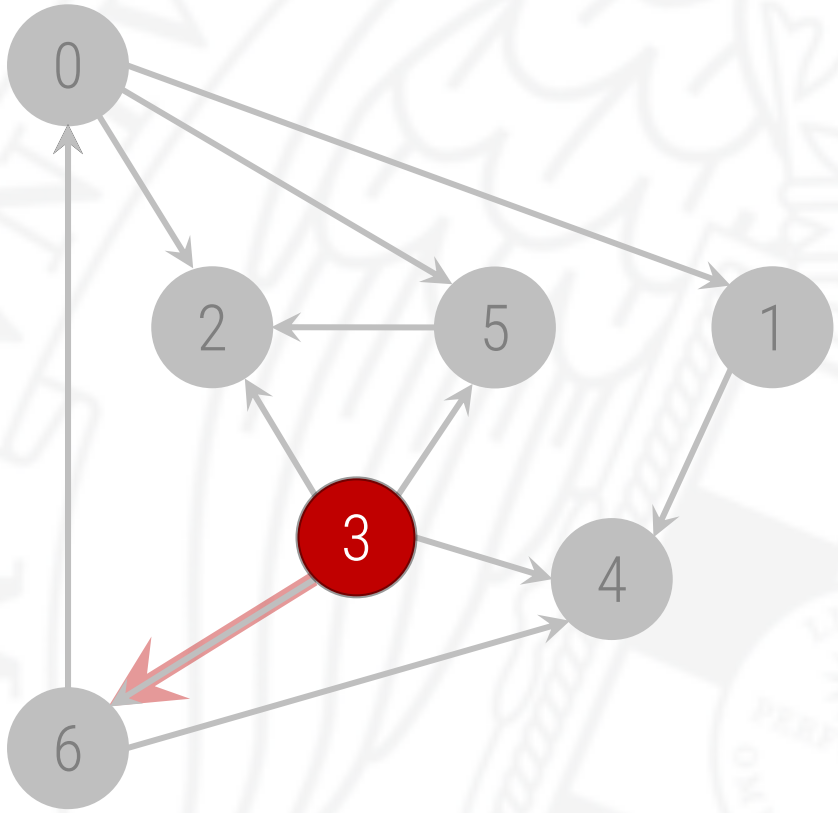


Postorden

4 1 2 5 0

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.

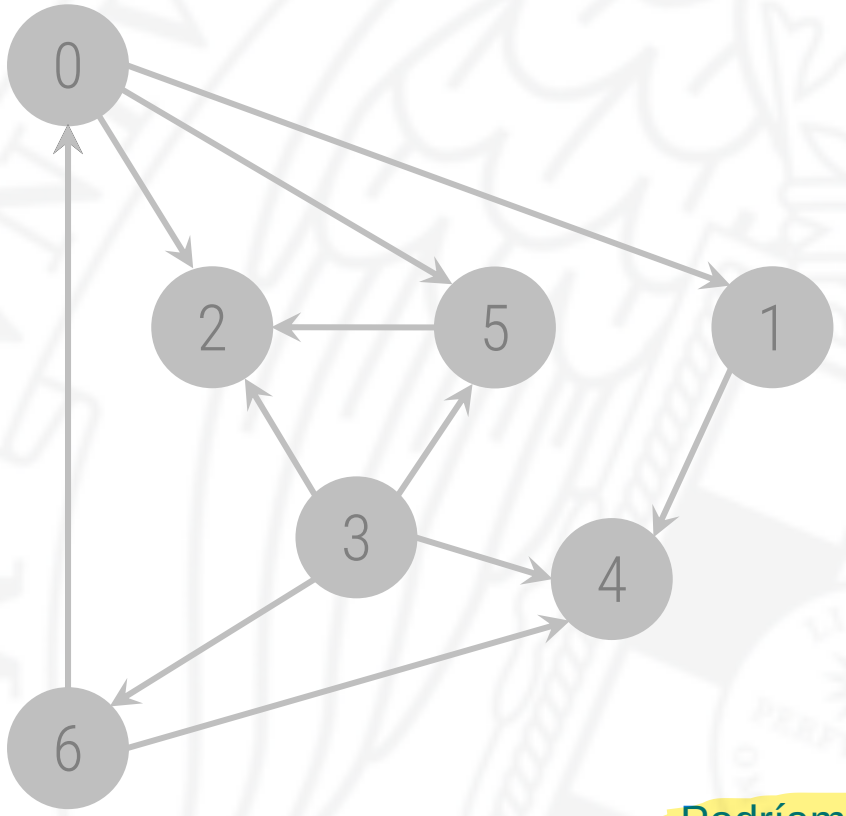


Postorden

4 1 2 5 0 6

Postorden

- Añadir el vértice al orden tras las llamadas recursivas.



Postorden

4 1 2 5 0 6 3

Ordenación topológica (postorden inverso)

3 6 0 5 2 1 4

Podríamos construir el postorden inverso empezando a insertar elementos desde atrás.

Corrección

- El postorden inverso de un DAG es una ordenación topológica.
- Demostración: Arista cualquiera $v \longrightarrow w$. Hacemos la llamada `dfs(v)`
  - Caso 1: `dfs(w)` ya se hizo y terminó.

    v estará antes que w en el postorden-inverso
  - Caso 2: `dfs(w)` aún no se ha hecho. La arista $v \longrightarrow w$ provocará que se haga `dfs(w)` y que termine antes de `dfs(v)`.
  - Caso 3: `dfs(w)` comenzó pero aún no ha terminado. Esto es imposible. La cadena de llamadas implica que existe un camino de w a v y la arista $v \longrightarrow w$ completaría un ciclo dirigido en un DAG.

ESTO ES IMPOSIBLE PORQUE EL GRAFO NO TIENE CICLOS.

Implementación

```
class OrdenTopologico {
public:
   // g es DAG
   OrdenTopologico(Digrafo const& g) : visit(g.V(), false) {
      for (int v = 0; v < g.V(); ++v)
         if (!visit[v])
            dfs(g, v);
   }

   // devuelve la ordenación topológica
   std::deque<int> const& orden() const {
      return _orden;
   }
```

Supone como precondición que es acíclico

Para cada vértice no marcada hace su recorrido en DFS.

Usamos una cola doble orden con el orden topológico.

Implementación

```
private:
    std::vector<bool> visit;
    std::deque<int> _orden; // ordenación topológica

    void dfs(Digrafo const& g, int v) {
        visit[v] = true;
        for (int w : g.ady(v))
            if (!visit[w])
                dfs(g, w);
        _orden.push_front(v);
    }
};
```

O(V+A)

Estamos haciendo el postorden inverso.