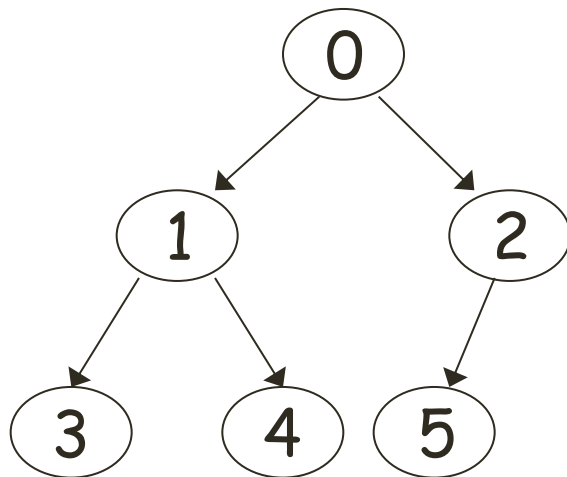


# 4. Recorridos sobre grafos

## *Recorrido en profundidad o DFS*

- Generalización del recorrido en *PreOrden* de un árbol:

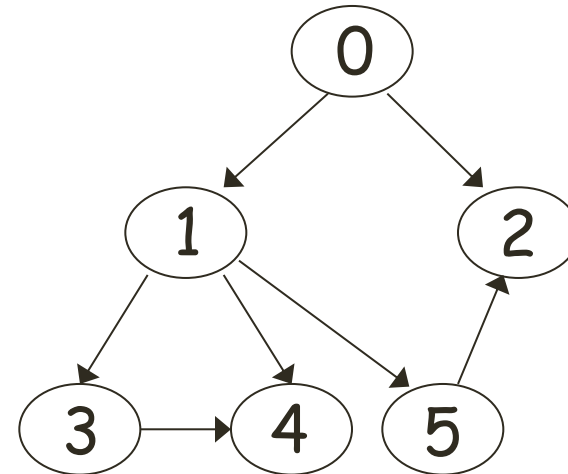
Árbol



*PreOrden*: Padre, Izq, Der

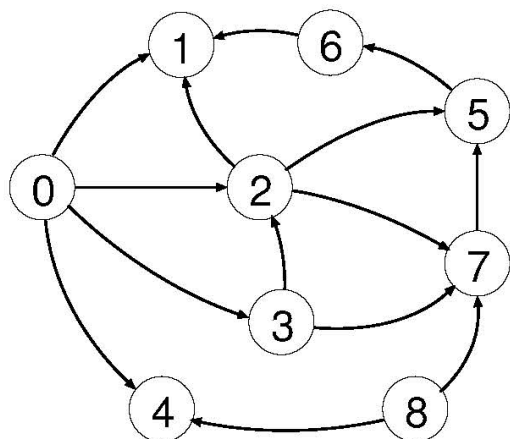
0, 1, 3, 4, 2, 5

Grafo



0, 1, 3, 4, 5, 2

Precaución para no  
repetir ningún vértice



nodos	<i>visita</i>								
<i>v/w</i>	0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0
<b>0/1,2,3,4</b>	1	-	-	-	-	-	-	-	-
<b>1/-</b>	-	2	-	-	-	-	-	-	-
<b>2/1,5,7</b>	-	-	3	-	-	-	-	-	-
<b>5/6</b>	-	-	-	-	-	4	-	-	-
<b>6/1</b>	-	-	-	-	-	-	5	-	-
<b>7/5</b>	-	-	-	-	-	-	-	6	-
<b>3/2,7</b>	-	-	-	7	-	-	-	-	-
<b>4/-</b>	-	-	-	-	8	-	-	-	-
<b>8/4,7</b>	-	-	-	-	-	-	-	-	9
<i>visita</i>	1	2	3	7	8	4	5	6	9


Orden de Visita de Nodos: 0, 1, 2, 5, 6, 7, 3, 4, 8

# 4. Recorridos sobre grafos

## *Implementación del recorrido DFS (1/2)*

```
public abstract class Grafo {  
    // El recorrido en profundidad necesita dos atributos  
    protected boolean visitados[]; // Para no repetir vértices  
    protected int ordenVisita; // Orden de visita de los  
                                // vértices  
  
    // Recorrido en profundidad (DFS): devuelve un array con  
    // los códigos de los vértices recorridos según DFS  
  
    public int[] toArrayDFS() {  
        int res[] = new int[numVertices()];  
        visitados = new boolean[numVertices()];  
        ordenVisita = 0;  
        for (int i = 0; i < numVertices(); i++)  
            if (!visitados[i]) toArrayDFS(i, res);  
        return res;  
    }  
}
```

Se inicializa  
automáticamente  
a false



# 4. Recorridos sobre grafos

## *Implementación del recorrido DFS (2/2)*

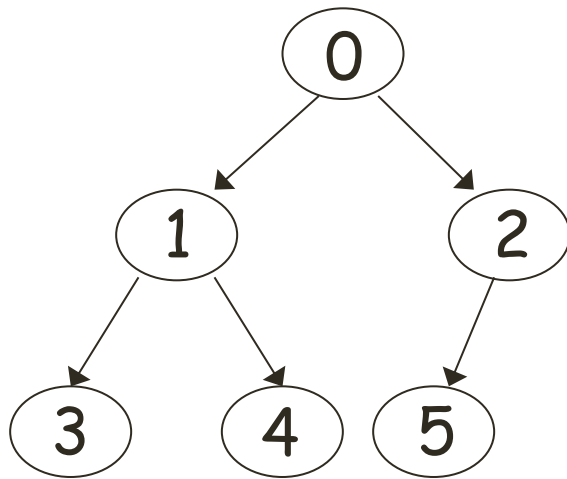
```
// Método recursivo para el recorrido en profundidad
protected void toArrayDFS(int origen, int res[]) {
    // Añadimos el vértice origen y lo marcamos como visitado
    res[ordenVisita++] = origen;
    visitados[origen] = true;
    // Recorremos los adyacentes del vértice origen
    ListaConPI<Adyacente> l = adyacentesDe(origen);
    for (l.inicio(); !l.esFin(); l.siguiente()) {
        Adyacente a = l.recuperar();
        if (!visitados[a.destino]) toArrayDFS(a.destino, res);
    }
}
```

# 4. Recorridos sobre grafos

## *Recorrido en anchura o BFS*

- Generalización del recorrido por niveles de un árbol:

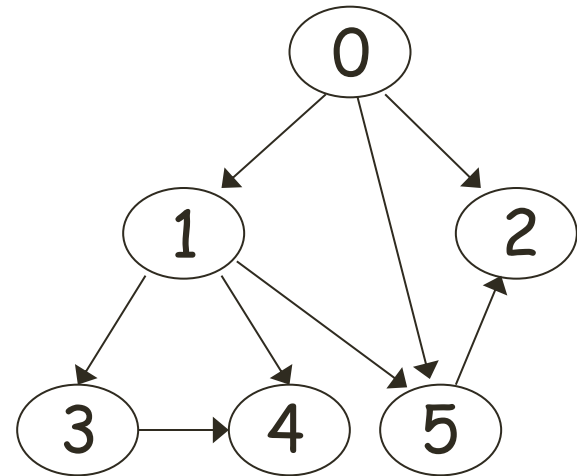
Árbol



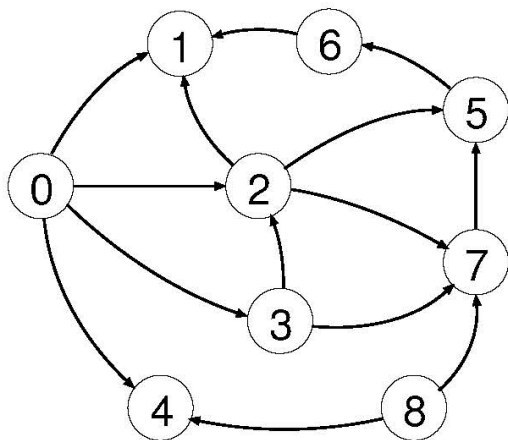
*Por niveles*

0, 1, 2, 3, 4, 5

Grafo



0, 1, 5, 2, 3, 4



		0	1	2	3	4	5	6	7	8	Q
0		1	0	0	0	0	0	0	0	0	< 0 >
	1	-	-	-	-	-	-	-	-	-	< >
	1	-	2	-	-	-	-	-	-	-	< 1 >
	2	-	-	3	-	-	-	-	-	-	< 1, 2 >
	3	-	-	-	4	-	-	-	-	-	< 1, 2, 3 >
	4	-	-	-	-	5	-	-	-	-	< 1, 2, 3, 4 >
	1	-	-	-	-	-	-	-	-	-	< 2, 3, 4 >
	2	-	-	-	-	-	-	-	-	-	< 3, 4 >
	1	-	-	-	-	-	-	-	-	-	—
	5	-	-	-	-	-	6	-	-	-	< 3, 4, 5 >
	7	-	-	-	-	-	-	-	7	-	< 3, 4, 5, 7 >
	3	-	-	-	-	-	-	-	-	-	< 4, 5, 7 >
	2	-	-	-	-	-	-	-	-	-	—
	7	-	-	-	-	-	-	-	-	-	—
	4	-	-	-	-	-	-	-	-	-	< 5, 7 >
	5	-	-	-	-	-	-	-	-	-	< 7 >
	6	-	-	-	-	-	-	8	-	-	< 7, 6 >
	7	-	-	-	-	-	-	-	-	-	< 6 >
	5	-	-	-	-	-	-	-	-	-	—
	6	-	-	-	-	-	-	-	-	-	< >
	1	-	-	-	-	-	-	-	-	-	< >
8		-	-	-	-	-	-	-	-	9	< 8 >
	8	-	-	-	-	-	-	-	-	-	< >
	4	-	-	-	-	-	-	-	-	-	—
	7	-	-	-	-	-	-	-	-	-	—
		1	2	3	4	5	6	8	7	9	

Orden de Visita de Nodos 0, 1, 2, 3, 4, 5, 7, 6, 8

# 4. Recorridos sobre grafos

## *Implementación del recorrido BFS (1/2)*

```
public abstract class Grafo {  
    ... // Además de los atributos visitados y ordenVisita, el  
        // recorrido BFS requiere una Cola auxiliar pues el  
        // recorrido es iterativo  
    protected Cola<Integer> q;  
  
    // Recorrido en anchura (BFS)  
    public int[] toArrayBFS() {  
        int res[] = new int[numVertices()];  
        visitados = new boolean[numVertices()];  
        ordenVisita = 0;  
        q = new ArrayCola<Integer>();  
        for (int i = 0; i < numVertices(); i++)  
            if (!visitados[i]) toArrayBFS(i, res);  
        return res;  
    }  
}
```

# 4. Recorridos sobre grafos

## *Implementación del recorrido BFS (2/2)*

```
protected void toArrayBFS(int origen, int res[]) {  
    res[ordenVisita++] = origen;  
    visitados[origen] = true;  
    q.encolar(origen);  
    while (!q.esVacia()) {  
        int u = q.desencolar().intValue();  
        ListaConPI<Adyacente> l = adyacentesDe(u);  
        for (l.inicio(); !l.esFin(); l.siguiente()) {  
            Adyacente a = l.recuperar();  
            if (!visitados[a.destino]) {  
                res[ordenVisita++] = a.destino;  
                visitados[a.destino] = true;  
                q.encolar(a.destino);  
            }  
        }  
    }  
}
```



# 4. Recorridos sobre grafos

## *Ejercicios*

**Ejercicio.** Implementa un método en la clase *Grafo* que compruebe si un vértice es alcanzable desde otro vértice dado.



## SOLUCIÓN:

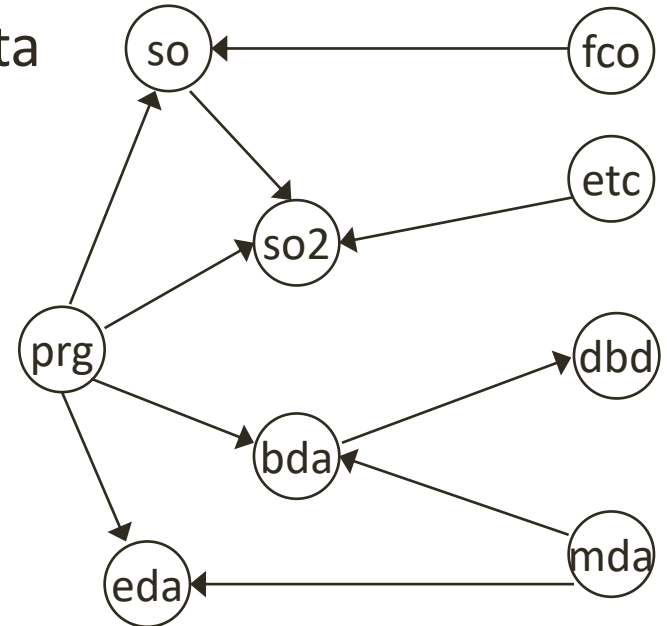
```
public boolean esAlcanzable(int vOrigen, int vDestino) {  
    visitados = new boolean[numVertices()];  
    return esAlcanzableRec(vOrigen, vDestino);  
}  
  
private boolean esAlcanzableRec(int vActual, int vDestino) {  
    if (vActual == vDestino) return true;  
    visitados[vActual] = true;  
    ListaConPI<Adyacente> ady = adyacentesDe(vActual);  
    for (ady.inicio(); !ady.esFin(); ady.siguiente()) {  
        int vSiguiente = ady.recuperar().destino;  
        if (visitados[vSiguiente] == 0 && esAlcanzableRec(vSiguiente, vDestino))  
            return true;  
    }  
    return false;  
}
```

# 5. Órdenes topológicos

## Introducción

- **Ejemplo:** el siguiente grafo representa los prerequisites entre asignaturas.

Una arista  $(u, w)$  indica que la asignatura  $u$  debe ser aprobada para poder matricularse en  $w$



- $\langle \text{prg}, \text{so}, \text{so2} \rangle, \langle \text{prg}, \text{bda}, \text{dbd} \rangle, \langle \text{mda}, \text{bda}, \text{dbd} \rangle, \langle \text{mda}, \text{eda} \rangle, \text{etc.}$

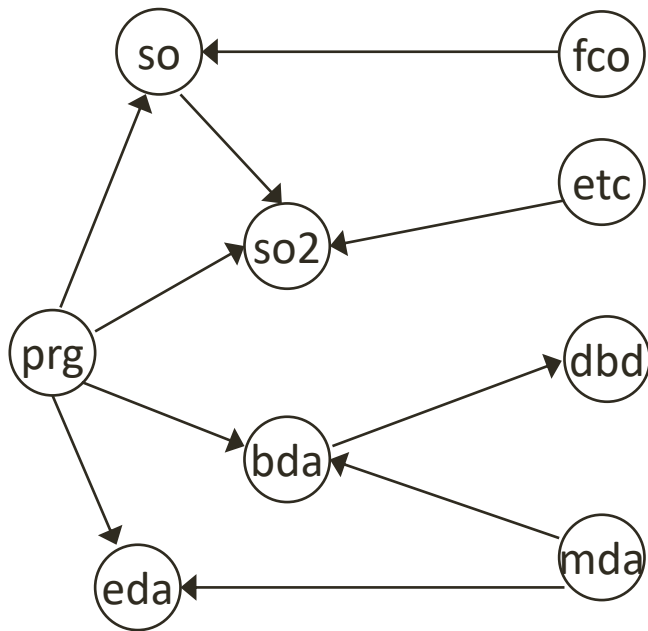
son **órdenes topológicos**

- Una **ordenación topológica** es una ordenación lineal de los vértices de un grafo acíclico dado, conservando la ordenación parcial original

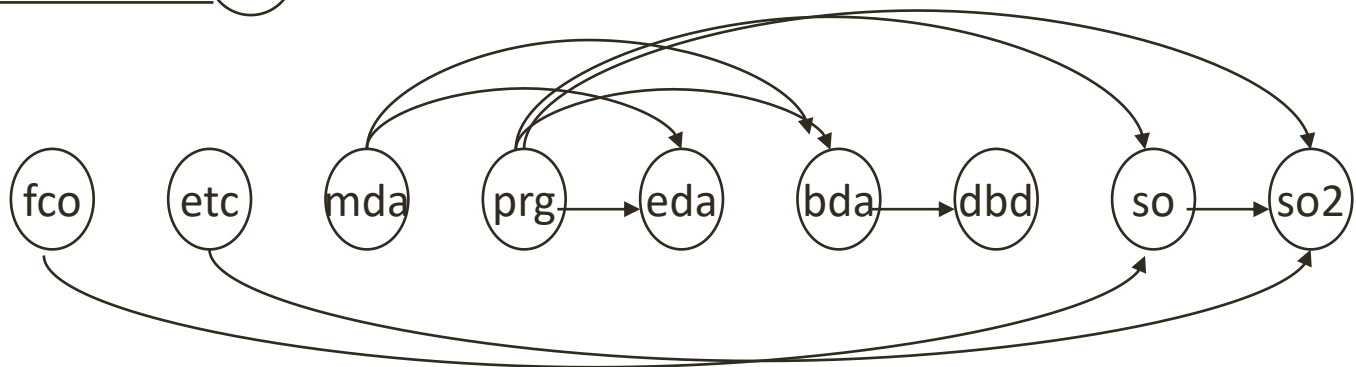
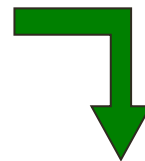
# 5. Órdenes topológicos

## *Introducción*

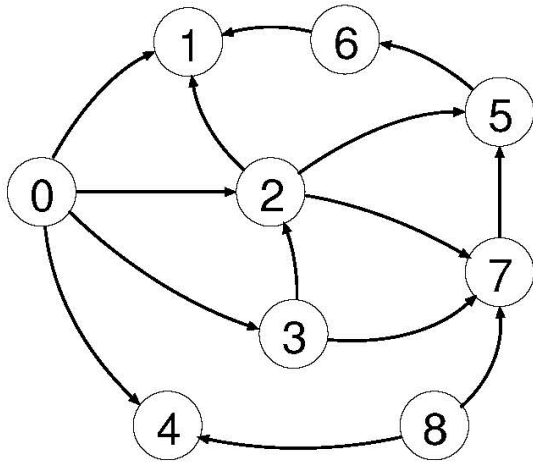
**Ejemplo:** encontrar un orden para poder estudiar TODAS las asignaturas:



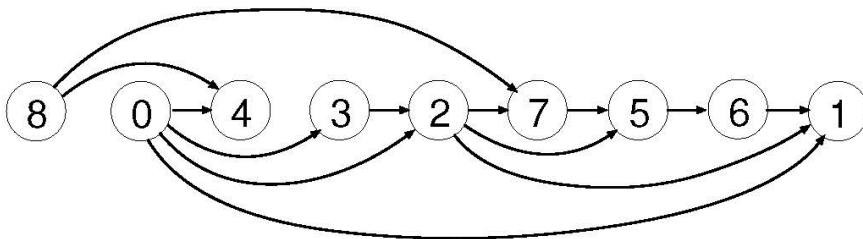
Recorrido en profundidad  
+  
uso de una pila de vértices



## Orden topológico en grafos acíclicos: Ejemplo



nodos	<i>visita</i>										
<i>v/w</i>	0	1	2	3	4	5	6	7	8		<i>Q</i>
—	0	0	0	0	0	0	0	0	0		<>
0/1,2,3,4	1	-	-	-	-	-	-	-	-		<>
1/-	-	2	-	-	-	-	-	-	-		< 1 >
2/1,5,7	-	-	3	-	-	-	-	-	-		< 1 >
5/6	-	-	-	-	-	4	-	-	-		< 1 >
6/1	-	-	-	-	-	-	5	-	-		< 6, 1 >
—	-	-	-	-	-	-	-	-	-		< 5, 6, 1 >
7/5	-	-	-	-	-	-	-	6	-		< 7, 5, 6, 1 >
—	-	-	-	-	-	-	-	-	-		< 2, 7, 5, 6, 1 >
3/2,7	-	-	-	7	-	-	-	-	-		< 3, 2, 7, 5, 6, 1 >
4/-	-	-	-	-	8	-	-	-	-		< 4, 3, 2, 7, 5, 6, 1 >
—	-	-	-	-	-	-	-	-	-		< 0, 4, 3, 2, 7, 5, 6, 1 >
8/4,7	-	-	-	-	-	-	-	-	9		< 8, 0, 4, 3, 2, 7, 5, 6, 1 >



Grafo ordenado topológicamente

# 5. Órdenes topológicos

## *Método lanzadera*

```
// Devuelve un array con los códigos de los vértices en orden
// topológico

public int[] toArrayTopologico() {
    visitados = new boolean[numVertices()];
    Pila<Integer> pVRecorridos = new ArrayPila<Integer>();
    // Recorrido de los vértices
    for (int vOrigen = 0; vOrigen < numVertices(); vOrigen++)
        if (!visitado[vOrigen])
            ordenacionTopologica(vOrigen, pVRecorridos);
    // Copia el resultado de la ordenación a un array
    int res[] = new int[numVertices()];
    for (int i = 0; i < numVertices(); i++)
        res[i] = pVRecorridos.desapilar();
    return res;
}
```

# 5. Órdenes topológicos

## *Método recursivo*

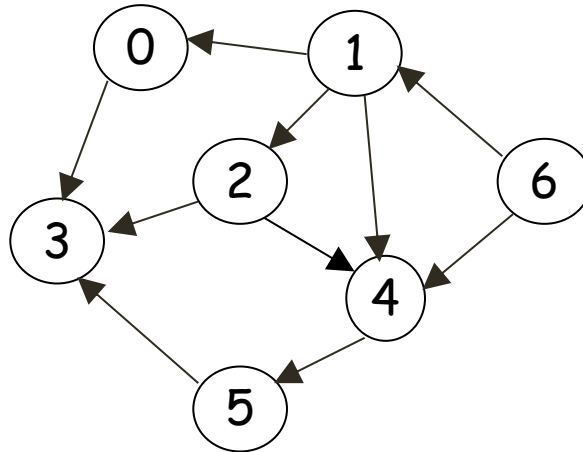
```
protected void ordenacionTopologica(int origen,  
                                   Pila<Integer> pVRecorridos) {  
    visitados[origen] = true;  
    // Recorremos los vértices adyacentes  
    ListaConPI<Adyacente> aux = adyacentesDe(origen);  
    for (aux.inicio(); !aux.esFin(); aux.siguiente()) {  
        int destino = aux.recuperar().destino;  
        if (!visitados[destino])  
            ordenacionTopologica(destino, pVRecorridos);  
    }  
    // Apilamos el vértice  
    pVRecorridos.apilar(origen);  
}
```

$$T_{\text{ordenacionTopologica}}(|V|, |A|) \in O(|V| + |A|)$$

# 5. Órdenes topológicos

## *Ejercicios*

**Ejercicio.** Siguiendo el método *ordenacionTopologica*, mostrar la ordenación topológica resultante para el siguiente grafo dirigido acíclico:



¿La ordenación obtenida es única? En caso negativo mostrar otra ordenación válida.





## SOLUCIÓN:

Ordenación obtenida:  $6 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 0 \rightarrow 3$

No es la única ordenación posible. Ejemplo:

$6 \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3$

