Ejercicio 1.

Sea $G = (V, A)$ un grafo dirigido con pesos:

$V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$

$A = \{(v_0,v_1, 2), (v_0,v_3, 1), (v_1,v_3, 3), (v_1,v_4, 10), (v_3,v_4, 2),$
$(v_3,v_6, 4), (v_3,v_5, 8), (v_3,v_2, 2), (v_2,v_0, 4), (v_2,v_5, 5),$
$(v_4,v_6, 6), (v_6,v_5, 1)\}$

Se pide:

a) $|V|$ y $|A|$

b) Vértices adyacentes a cada uno de los vértices

c) Grado de cada vértice y del grafo

d) Caminos simples desde v_0 a v_6, y su longitud con y sin pesos

e) Vértices alcanzables desde v_0

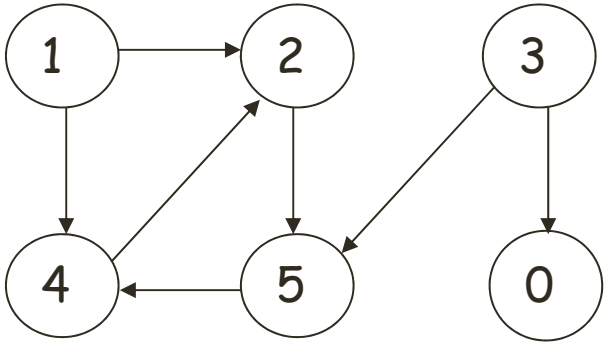
f) Caminos mínimos desde v_0 al resto de vértices

g) ¿Tiene ciclos?

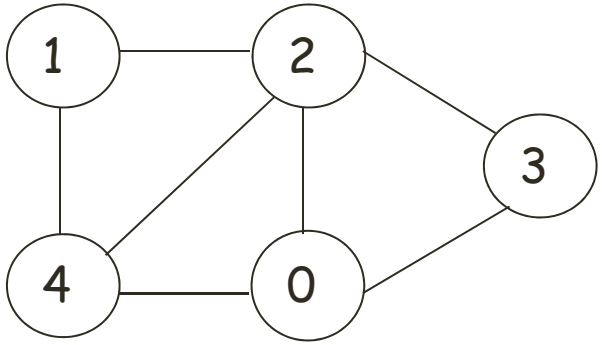
Ejercicio 2.

Representad los siguientes grafos mediante una matriz de adyacencia y mediante listas de adyacencia

a)

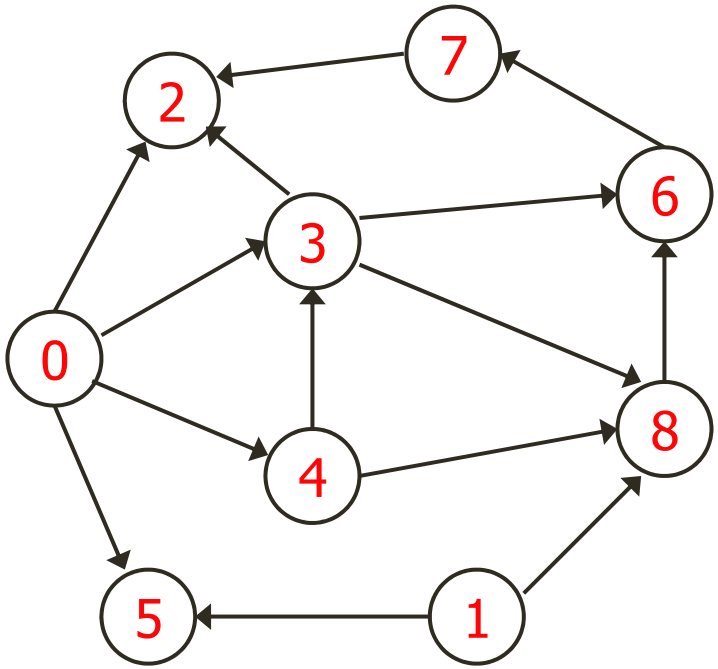


b)



Ejercicio 3.

Mostrar el resultado de imprimir por pantalla el recorrido en profundidad y en anchura del siguiente grafo:



Ejercicio 4.

Diseña un método en la clase *ForestUFSet* que muestre por pantalla los identificadores de cada uno de los conjuntos que hay en el *UFSet*.

Ejercicio 5.

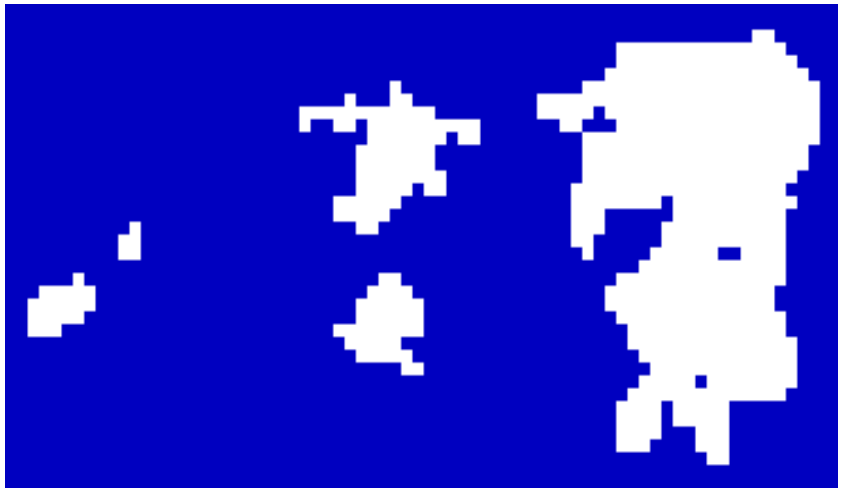
Diseña un método en la clase *ForestUFSet* que muestre por pantalla los elementos del conjunto al que pertenece un elemento dado *x*.

Ejercicio 6.

Diseña un método en la clase *ForestUFSet* que devuelva el número de elementos que habría en el conjunto resultante de unir los conjuntos a los que pertenecen dos elementos dados *x* e *y*.

Ejercicio 7.

Se dispone de una matriz de tipo *boolean* que representa un mapa de un archipiélago. Un valor *true* en una posición (x,y) indica que hay tierra en ese punto, mientras que un valor *false* indica que hay mar.



Ejemplo:

Los valores *true* y *false* se representan mediante los colores blanco y azul, respectivamente.

Implementa un método con el siguiente perfil que permita calcular el número de islas que hay en el archipiélago:

```
public static int numIslas(boolean[][] mapa);
```

Ejercicio 8.

Definir los siguientes métodos en la clase *GrafoDirigido*:

a) Consultar el **grado de salida** de un vértice dado

b) Consultar el **grado de entrada** de un vértice dado

c) Empleando los dos métodos anteriores, escribir un método que devuelva el **grado** del grafo

d) Comprobar si un vértice es **fuente**, es decir, si es un vértice del que solo salen aristas

e) Comprobar si un vértice es un **sumidero** (i.e. un vértice al que sólo llegan aristas) al que llegan aristas de todos los demás vértices del grafo

Ejercicio 9.

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

Ejercicio 10.

Un grafo transpuesto *T* de un grafo *G* tiene el mismo conjunto de vértices pero con las direcciones de las aristas en sentido contrario, es decir, que una arista (u, v) en *G* se corresponde con una arista (v, u) en *T*.

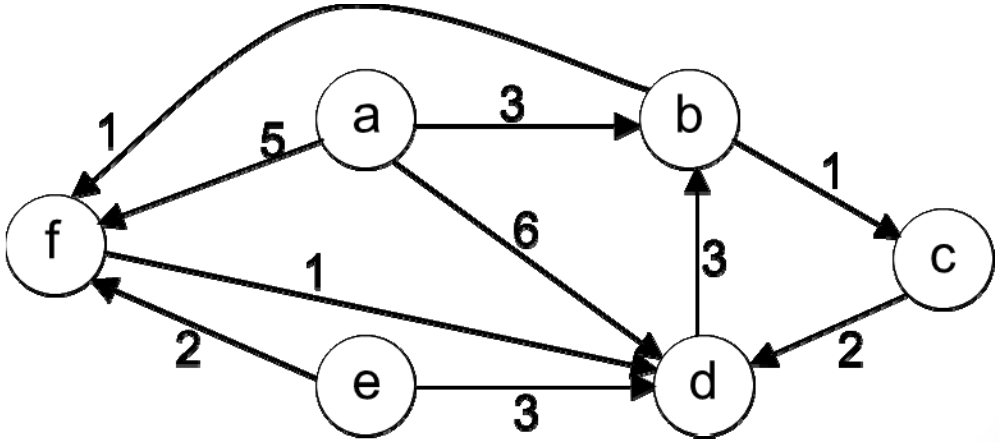
Diseña un método en la clase *GrafoDirigido* que permita obtener su grafo transpuesto:

```
public GrafoDirigido grafoTranspuesto();
```

Ejercicio 11.

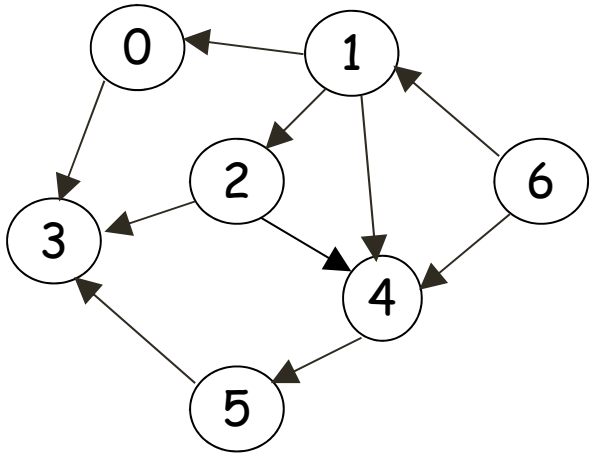
Los vértices del siguiente grafo representan personas (Ana, Begoña, Carmen, Daniel, Eliseo y Francisco) y las aristas indican si una persona tiene el número de móvil de otra. El peso de una arista es el coste de enviar un SMS (por ejemplo, Ana puede enviar un SMS a Begoña por 3 céntimos).

Haz una traza de Dijkstra para averiguar cuál sería la forma más barata de que Ana le haga llegar un SMS a Francisco.



Ejercicio 12.

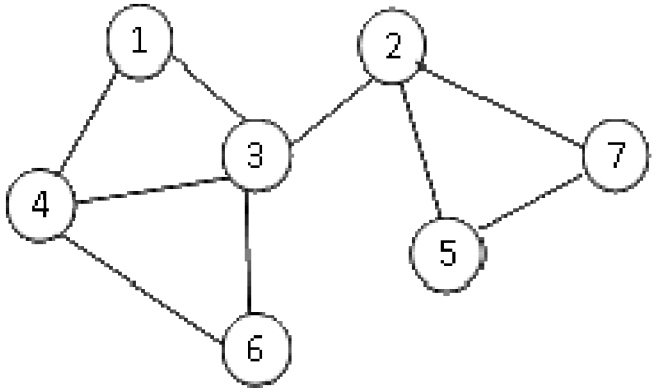
Siguiendo el método *ordenacionTopologica*, mostrar la ordenación topológica resultante para el siguiente grafo dirigido acíclico:



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

Ejercicio 13.

En teoría de grafos un *puente* es una arista tal que si fuese eliminada de un grafo incrementaría el número de componentes conexas de éste. Nótese entonces que un grafo conexo dejaría de serlo si se elimina una arista puente; por ejemplo, la arista (2, 3) del grafo de la siguiente figura es un puente.

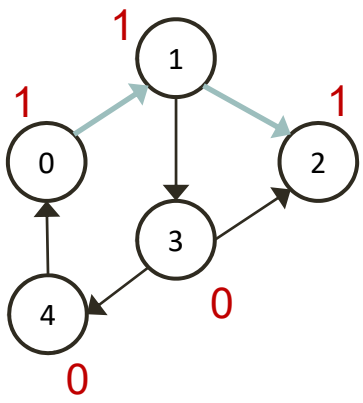


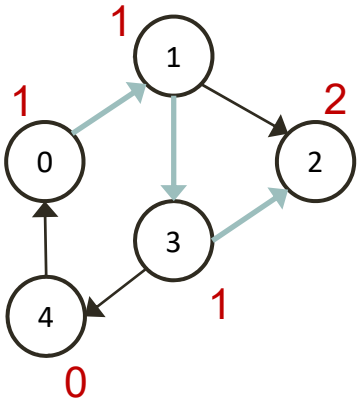
Se pide diseñar un método en la clase *Grafo* que compruebe si una arista (i, j) es una arista puente.

Ejercicio 14.

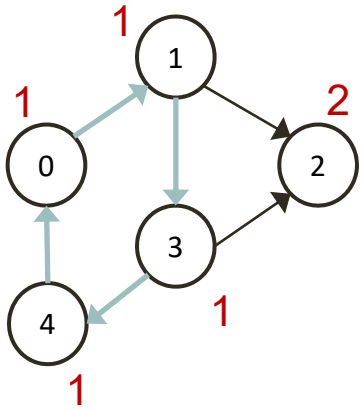
Diseña un método en la clase *Grafo* que compruebe si el grafo tiene ciclos.

Para ello no basta con saber si un vértice ha sido visitado o no, necesitamos tres posibles estados: **0** (no visitado), **1** (visitado en el camino actual), **2** (visitado por otro camino).





No es un ciclo, el vértice 2 fue visitado por otro camino



Ciclo detectado: una arista vuelve a un vértice visitado en el camino actual