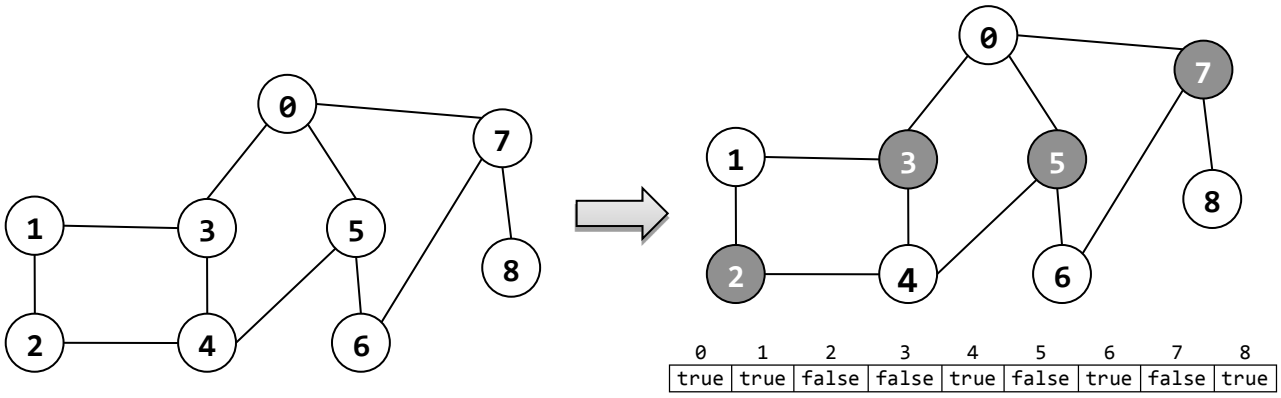
Resolución del Segundo Parcial de EDA (31 de Mayo de 2019)

1) ***(3.5 puntos)*** Dado un grafo conexo y no dirigido, se desea saber si se pueden colorear sus vértices con dos colores de forma que no haya aristas entre los vértices de un mismo color (equivalentemente, que todos los vértices adyacentes a uno dado tengan un color distinto al de este). Si los colores son blanco (**`true`**) y gris (**`false`**), dado el siguiente grafo, se pueden colorear los vértices de la forma siguiente:



Se pide implementar un método de instancia en la clase **`Grafo`** que, partiendo del vértice 0, devuelva **`true`** si el grafo se puede colorear, y **`false`** en caso contrario.

Para el grafo de la figura anterior, el método devuelve **`true`**. Para otro grafo, como el de la figura pero con una arista más entre los vértices 0 y 1, el método devolvería **`false`**.

```
public boolean colorDFS0() {
    visitados = new int[numVertices()];
    boolean[] color = new boolean[numVertices()];
    visitados[0] = 1;
    color[0] = true;
    return colorDFS(0, color);
}
protected boolean colorDFS(int v, boolean[] color) {
    ListaConPI<Adyacente> l = adyacentesDe(v);
    for (l.inicio(); !l.esFin(); l.siguiente()) {
        int w = l.recuperar().getDestino();
        if (visitados[w] == 0) {
            visitados[w] = 1;
            color[w] = !color[v];
            if (!colorDFS(w, color)) { return false; }
        }
        else if (color[w] == color[v]) { return false; }
    }
    return true;
}
```

2) ***(3 puntos)*** Se pide diseñar un método estático e iterativo que devuelva el número de elementos de un array genérico **v** que incumplen la propiedad de orden de un Montículo Binario. Para ello, considérese lo siguiente:

- El número de **compareTo** que el método debe realizar ha de ser el menor posible.
- Todas las posiciones de **v** están ocupadas, incluida la cero.

Algunos ejemplos, considerando que **v** es un array de enteros:

- Si `v = {6, 2, 7, 4, 5}`, entonces el resultado es 1.
- Si `v = {9, 15, 8, 16, 12, 10}`, entonces el resultado es 2.

```
public static <E extends Comparable<E>> int incumplenPOrden(E[] v) {
    int res = 0;
    for (int i = 1; i < v.length; i++) {
        if (v[i].compareTo(v[(i - 1) / 2]) < 0) { res++; }
    }
    return res;
}
```

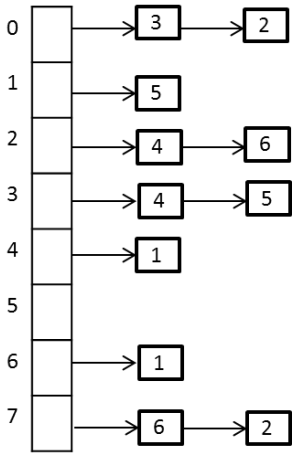
Alternativamente:

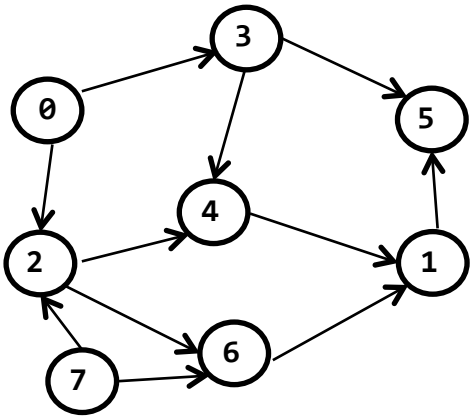
```
public static <E extends Comparable<E>> int incumplenPOrden(E[] v) {
    int res = 0;
    for (int i = 0; i < v.length / 2; i++) {
        if (v[i].compareTo(v[i * 2 + 1]) > 0) {
            res++;
        }
        if (i * 2 + 2 < v.length && v[i].compareTo(v[i * 2 + 2]) > 0) {
            res++;
        }
    }
    return res;
}
```

O bien:

```
public static <E extends Comparable<E>> int incumplenPOrden(E[] v) {
    int res = 0;
    for (int i = 0; i < v.length; i++) {
        if (i * 2 + 1 < v.length && v[i].compareTo(v[i * 2 + 1]) > 0) {
            res++;
        }
        if (i * 2 + 2 < v.length && v[i].compareTo(v[i * 2 + 2]) > 0) {
            res++;
        }
    }
    return res;
}
```

3) ***(1.5 puntos)*** Sea el siguiente grafo dirigido y acíclico representado con listas de adyacencia.





Se pide obtener los resultados de...

a) Su recorrido DFS.

```
0 3 4 1 5 2 6 7
```

b) Su recorrido BFS.

```
0 3 2 4 5 6 1 7
```

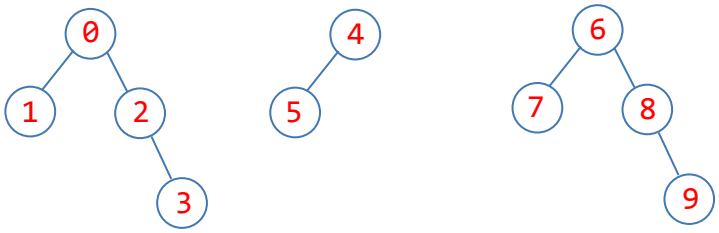
c) Su Ordenación Topológica.

```
7 0 2 6 3 4 1 5
```

4) ***(2 puntos)*** Sea la siguiente representación de un **`UF-Set`**:

0	1	2	3	4	5	6	7	8	9
-3	0	0	2	-2	4	-3	6	6	8

a) Dibujar el bosque de árboles que contiene.



b) Teniendo en cuenta que se implementa la fusión por rango y la compresión de caminos, indicar cómo irá evolucionando dicha representación tras la ejecución de las instrucciones de la siguiente tabla.

Nota: al unir dos árboles con la misma altura (o rango), el primer árbol deberá colgar del segundo.

	0	1	2	3	4	5	6	7	8	9
`find(9)`	-3	0	0	2	-2	4	-3	6	6	**6**
`union(4, 6)`	-3	0	0	2	**6**	4	-3	6	6	6
`union(0, 6)`	**6**	0	0	2	6	4	**-4**	6	6	6

ANEXO

Las clases Grafo y Adyacente del paquete grafos.

```
public abstract class Grafo {
    protected static final double INFINITO = Double.POSITIVE_INFINITY;
    protected boolean esDirigido;
    protected int[] visitados;
    protected int ordenVisita;
    protected Cola<Integer> q;
    ...
    public abstract int numVertices();
    public abstract int numAristas();
    public abstract ListaConPI<Adyacente> adyacentesDe(int i);
    ...
}

public class Adyacente {
    protected int destino;
    protected double peso;
    public Adyacente(int d, double p) { … }
    public int getDestino() { … }
    public double getPeso() { … }
    public String toString() { … }
}
```

La clase ForestUFSet del paquete jerarquicos.

```
public class ForestUFSet implements UFSet {
    protected int[] elArray;
    public ForestUFSet(int n) { … }
    public int find(int i) { … }
    public void union(int claseI, int claseJ) { … }
}
```