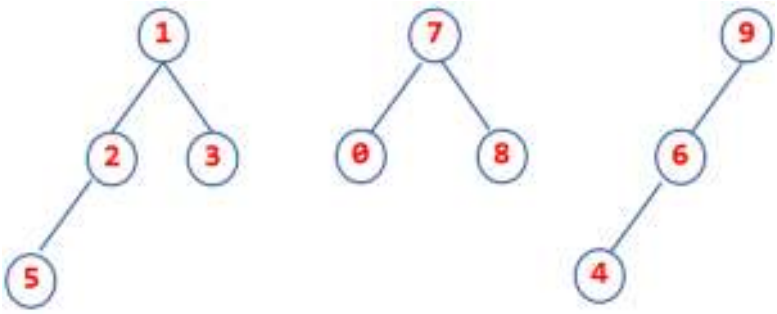
Pregunta 1 (UFSets). 1.5 puntos

1. Completa el contenido del array que representa el siguiente UF-set.



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

2. Teniendo en cuenta que se implementa la **fusión por rango** y la **compresión de caminos**, completa el contenido de la tabla que representa el estado del array después de ejecutar las operaciones que aparecen en la primera columna de forma consecutiva. **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(4)	7	-3	1	1	9	2	9	-2	7	-3
union(7, 9)	7	-3	1	1	9	2	9	9	7	-3
union(1, 9)	7	9	1	1	9	2	9	9	7	-4

Pregunta 2 (Traza Dijkstra). 1.5 puntos

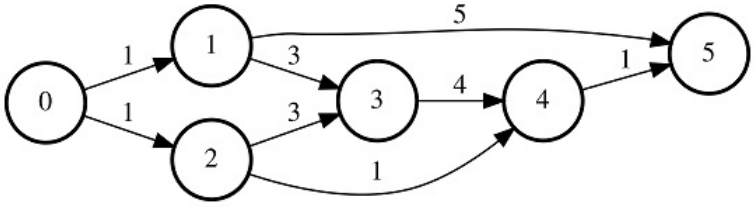
Dado el grafo dirigido de la figura, realiza la traza del algoritmo de **Dijkstra** tomando el vértice **0** del grafo como vértice origen. Para ello, completa la tabla que hay debajo de la figura, indicando cuál es el estado de las variables **v** (siguiente vértice a explorar), **qPrior** (cola de prioridad), **visitados** (vértices ya explorados), **distanciaMin** (distancia mínima desde el vértice origen hasta el momento) y **caminoMin** (predecesor del camino mínimo hasta el momento) **TRAS cada iteración del bucle de Dijkstra**. Rellena tantas filas como sea necesario. Contesta posteriormente a las cuestiones planteadas.

Es MUY IMPORTANTE que ...

****Escribas los elementos de cada fila de la tabla con el formato que se muestra para cada uno de ellos en su segunda fila (inicialización de Dijkstra). En concreto, escribe dos letras "o" minúsculas juntas ("oo") para indicar INFINITO.**

****Escribas los pares de qPrior EN ORDEN ASCENDENTE DE PESOS.**

****Escribas "NADA" para indicar que qPrior está vacía.**



v	qPrior	visitados						distanciaMin						caminoMin					
		0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
	(0, 0)	0	0	0	0	0	0	0	oo	oo	oo	oo	oo	-1	-1	-1	-1	-1	-1
0	**(1, 1) (2, 1)**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**oo**	**oo**	**oo**	**-1**	**0**	**0**	**-1**	**-1**	**-1**
1	**(2, 1) (3, 4) (5, 6)**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**4**	**oo**	**6**	**-1**	**0**	**0**	**1**	**-1**	**1**
2	**(4, 2) (3, 4) (5, 6)**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**4**	**2**	**6**	**-1**	**0**	**0**	**1**	**2**	**1**
4	**(5,3) (3, 4) (5, 6)**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**4**	**2**	**3**	**-1**	**0**	**0**	**1**	**2**	**4**
5	**(3,4) (5, 6)**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**4**	**2**	**3**	**-1**	**0**	**0**	**1**	**2**	**4**
3	**(5,6)**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**4**	**2**	**3**	**-1**	**0**	**0**	**1**	**2**	**4**
5	**NADA**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**4**	**2**	**3**	**-1**	**0**	**0**	**1**	**2**	**4**

Contesta a las siguientes cuestiones:

a) ¿Cuál es la distancia del vértice origen 0 al vértice 3? **4**

b) ¿Cuál es el camino mínimo del vértice origen 0 al vértice 5? **0 2 4 5**

Pregunta 3 (Costes). 1.5 puntos

Considerar la siguiente implementación de un método de instancia en la clase **MonticuloBinario**:

```
protected int listaMenoresQue(E e, ListaConPI<E> l, int i) {
    if (i > talla || elArray[i].compareTo(e) >= 0) return 0;
    l.insertar(elArray[i]);
    return 1 + listaMenoresQue(e, l, 2 * i) + listaMenoresQue(e, l, 2 * i + 1);
}
```

Siguiendo los pasos que se indican a continuación, se pide obtener el coste Temporal Asintótico del método **listaMenoresQue**.

Paso 1. Determinar la talla del problema, **x**.
Se debe contestar **escribiendo V o F en los huecos que figuran delante de las afirmaciones**.

- **F** La talla del problema, **x**, es la talla de **l**, la ListaConPI segundo argumento del método.
- **V** La talla del problema, **x**, es el tamaño del nodo **i** (o talla del Heap con raíz en **i)**, del objeto **this** invocador del método.

Paso 2. Determinar las instancias significativas.
Se debe contestar **escribiendo V o F en los huecos que figuran delante de las afirmaciones**.

- **F** El Mejor de los Casos se da cuando el nodo **i** es una Hoja.
- **F** El Mejor de los Casos se da cuando todos los elementos del nodo **i** son menores que el argumento **e**.
- **V** El Mejor de los Casos se da cuando todos los elementos del nodo **i** son mayores que el argumento **e**.
- **F** El Peor de los Casos se da cuando el nodo **i** es una Hoja.
- **V** El Peor de los Casos se da cuando todos los elementos del nodo **i** son menores que el argumento **e**.
- **F** El Peor de los Casos se da cuando todos los elementos del nodo **i** son mayores que el argumento **e**.

Paso 3. Escribir las Relaciones de Recurrencia en el caso general.

Para ello, completa los huecos de las que aparecen a continuación, teniendo en cuenta que:

- **Se debe expresar la talla** del problema mediante **x**
- **Se debe expresar la sobrecarga** (coste de las instrucciones del caso general que no son llamadas recursivas) **como sigue**:
  - sobrecarga **constante:** escribe **k**
  - sobrecarga **lineal:** escribe **k*x**
  - sobrecarga **cuadrática:** escribe **k*x*x**

Relaciones de Recurrencia en el caso general, cuando x > **1**.

**Para el Mejor de los Casos: $T^{M}_{listaMenoresQue}(x) = \mathbf{0} * T^{M}_{listaMenoresQue}(\mathbf{x/2}) + \mathbf{k}$

**Para el Peor de los Casos: $T^{P}_{listaMenoresQue}(x) = \mathbf{2} * T^{P}_{listaMenoresQue}(\mathbf{x/2}) + \mathbf{k}$

Paso 4. Indicar su coste Temporal Asintótico.

Para ello, completa cada hueco de la siguiente línea con aquellas de las siguientes opciones que consideres adecuadas: **Theta**, **Omega**, **O**, **(1)**, **(logx)**, **(x)**, **(x*x)**, **(x*logx)** o **(2^x)**.

- **Se debe respetar la notación, incluidos los paréntesis que aparecen en algunas de las opciones**.

Como mínimo (cota inferior), $T_{listaMenoresQue}(x)$ € **Omega(1)

Como máximo (cota superior), $T_{listaMenoresQue}(x)$ € **O(x)

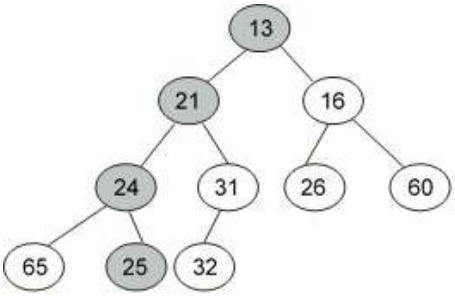
Pregunta 4 (Implementación. Heaps). 2.5 puntos

Completa la implementación del método de instancia de la clase **MonticuloBinario** que se muestra a continuación para que, con el menor coste posible, devuelva una **ListaConPI** con los elementos del camino que va de la menor de sus hojas a su raíz.

Importante. Considerar las siguientes precondiciones para el método:

- El montículo **this** contiene al menos un elemento (no es vacío).
- En el montículo **this** existe una única hoja de valor menor que todas las demás.

Ejemplo. Sea el montículo de la siguiente figura, la lista resultado sería [25, 24, 21, 13].



La implementación parcial del método es:

```
public ListaConPI<E> caminoDesdeLaMenorHoja() {
    ListaConPI<E> lpi = new LEGListaConPI<E>();
    // 1º paso, obtención de la posición de la hoja menor:
    int posMenor = /** hueco 1 **/;
    for (int i = /** hueco 2 **/; i <= /** hueco 3 **/; i++) {
        if (/** hueco 4 **/) {
            /** hueco 5 **/;
        }
    }
    // 2º paso, construcción del camino desde la hoja menor:
    /** hueco 6: puede usar instrucciones de control (while o if) */

    return lpi;
}
```

Indica el código a completar en cada hueco a continuación:

hueco 1: **talla / 2 + 1**

hueco 2: **posMenor + 1**

hueco 3: **talla**

hueco 4: **elArray[posMenor].compareTo(elArray[i]) > 0**

hueco 5: **posMenor = i**

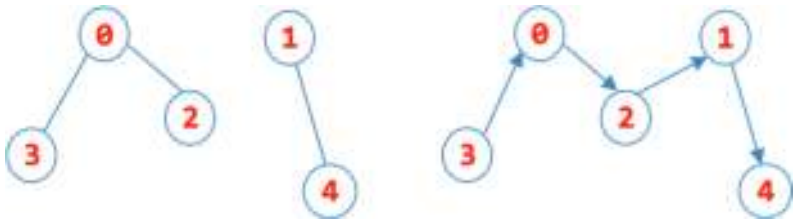
hueco 6. Recuerda que en este hueco DEBES implementar lo necesario para insertar en la **ListaConPI** resultado los valores en los nodos que forman el camino. Si para hacerlo quieres usar una o más instrucciones de control (while o if) debes escribir en huecos distintos su "cabecera" y cada una de las instrucciones de su cuerpo. MUY IMPORTANTE: NO puedes usar llaves de inicio y/o final de bloque en ningún hueco Y debes escribir NADA en los huecos que no utilices.

hueco 6:

- **while (posMenor >= 1)**
- **lpi.insertar(elArray[posMenor]);**
- **posMenor = posMenor / 2;**

Pregunta 5 (Implementación. Grafos). 3.0 puntos

Definición: un vértice **v** es una raíz de un grafo Acíclico **G** SI cada uno de los vértices de **G** es alcanzable desde **v**. Por ejemplo, dados los dos grafos Acíclicos de la siguiente figura, en el No Dirigido NO hay ningún vértice raíz y en el Dirigido solo hay uno, el vértice 3.



A partir de esta definición, se dispone de la siguiente versión incompleta, con 7 huecos, de un método de la clase Grafo que ha sido diseñado para devolver, en el menor **tiempo de ejecución** posible, el primer vértice raíz de un grafo Acíclico, o -1 si no existe tal vértice. Así, para los grafos de la figura anterior, el método devuelve -1 para el grafo No Dirigido y 3 para el Dirigido.

```
/** PRECONDICIÓN: this.Grafo es Aciclico */
public int primeraRaiz() {

    // Inicialización del DFS de this.Grafo
    /* hueco 1 */;
    /* hueco 2 */;

    // Ejecución del DFS de this.Grafo, en la que SE
    // DEBE devolver su 1er vértice raiz tan pronto
    // se encuentre (vía return) y SE DEBEN actualizar
    // las variables inicializadas en los huecos 1 y 2
    for (v = 0; v < numVertices(); v++) {

        /* hueco 3: puede usar instrucciones de control (if o for) */
    }

    // Fin del DFS de this.Grafo
    return -1;
}
protected void primeraRaiz(int v) {

    // Inicialización del DFS del vértice v
    /* hueco 4 */;
    /* hueco 5 */;
    ListaConPI<Adyacente> l = /* hueco 6 */;

    // Ejecución DFS del vértice v, en la que SE
    // DEBEN actualizar las variables del DFS y w
    int w;
    /* hueco 7: puede usar instrucciones de control (if o for) */

    // Fin del DFS del vértice v y del método
}
```

Completa los huecos del método **primeraRaiz** que figuran a continuación. Al hacerlo, **es MUY IMPORTANTE que respetes ESCRUPULOSAMENTE la sintáxis Java Y uses SOLAMENTE...**

****Los atributos protected int[] visitados e int ordenVisita de la clase Grafo .**

****Los métodos públicos numVertices() y adyacentesDe(int v) de la clase Grafo .**

****El atributo protected int destino de la clase Adyacente .**

****Las variables locales ya definidas en el código.**

- hueco 1: **visitados = new int[numVertices()]**
- hueco 2: **ordenVisita = 0**
- hueco 3: Recuerda que en este hueco DEBES devolver el primer vértice raíz del grafo tan pronto lo encuentres (vía return) Y actualizar las variables que hayas inicializado en los huecos 1 y 2 . Si para hacerlo quieres usar una o más instrucciones de control (if o for) debes escribir en huecos distintos su "cabecera" y cada una de las instrucciones que tuviera su cuerpo. **MUY IMPORTANTE: NO puedes usar llaves de inicio y/o final de bloque en ningún hueco Y debes escribir NADA en los huecos que no utilices.**
    - **primeraRaiz(v);**
    - **if (ordenVisita == numVertices())**
    - **return v;**
    - **visitados = new int[numVertices()];**
    - **ordenVisita = 0;**
- hueco 4: **visitados[v] = 1**
- hueco 5: **ordenVisita++**
- hueco 6: **adyacentesDe(v)**
- hueco 7: Recuerda que en este hueco DEBES actualizar tanto las variables del DFS como la variable **w**. Si para hacerlo quieres usar una o más instrucciones de control (if o for) debes escribir en huecos distintos su "cabecera" y cada una de las instrucciones de su cuerpo. **MUY IMPORTANTE: NO puedes usar llaves de inicio y/o final de bloque en ningún hueco Y debes escribir NADA en los huecos que no utilices.**
    - **for (l.inicio(); !l.esFin(); l.siguiente())**
    - **w = l.recuperar().destino;**
    - **if (visitados[w] == 0)**
    - **primeraRaiz(w);**