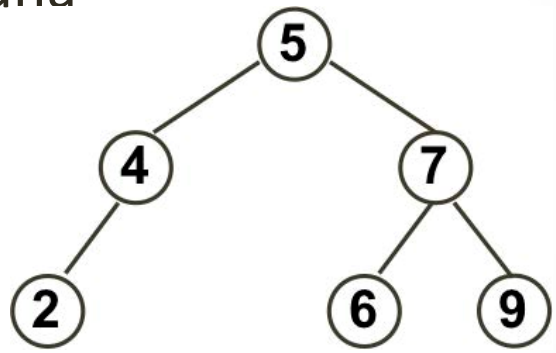
Ejercicio 1

Sea la clase ***ABBInteger*** una derivada de ***ABB***, cuyos elementos son todos enteros positivos. Diseña en ella un método *caminoQueSuma* que, con el menor coste posible y recursivamente, compruebe si existe una secuencia de nodos (camino) desde su raíz a uno de sus descendientes cuyos datos sumen ***s***, con ***s*** > 0.

Así, por ejemplo, dado el ABB de la derecha, el método devuelve ***true*** si el valor de ***s*** es 9, 11 o 12; sin embargo, devuelve ***false*** si el valor de ***s*** es 10 o 19.



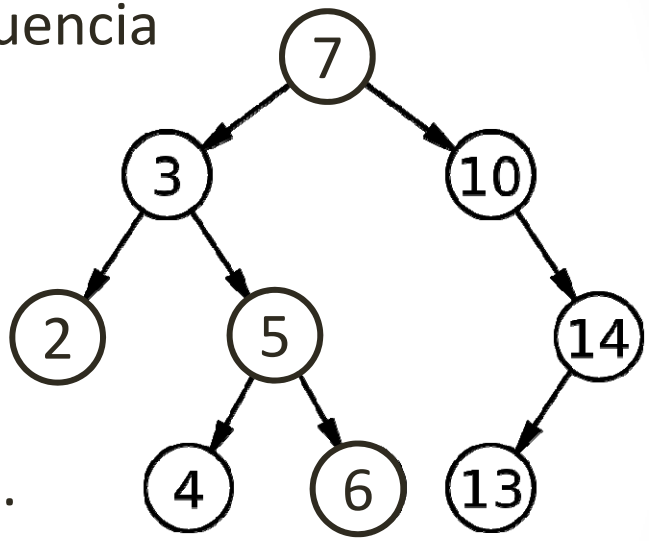

```
public class ABBDeInteger extends ABB<Integer> {
  …
}
```

Ejercicio 2

Escribe una función en la clase ABB que devuelva una Lista con PI con todos los elementos del árbol, ordenados de menor a mayor, que sean menores que uno dado.

Ejercicio 3

Escribe una función en la clase ABBInteger, que extiende de un ABB<Integer>, que devuelva el primer valor que falta en el árbol (suponiendo que la secuencia de valores empieza por el valor mínimo del árbol). En el árbol de la derecha, por ejemplo, el primer elemento que falta en la secuencia es el 8. La función devolverá ***null*** si el árbol está vacío.



Para que el coste en el peor caso sea lineal respecto al número de nodos, se podrá utilizar un Pila como estructura auxiliar donde almacenar los nodos conforme se van visitando.

Ejercicio 4

En la clase ABB, se pide implementar un método público *contarEnRango* que, con el menor coste posible, devuelva el número de elementos del ABB comprendidos en el intervalo `[eI, eS]`, donde `eI` y `eS` son dos elementos dados de tipo genérico `E` tales que `eI` es menor o igual que `eS`.

Ejercicio 5

La sucesión de Fibonacci es una sucesión de números naturales que comienza con los números 0 y 1 y, a partir de estos, cada término es la suma de los dos anteriores:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, …

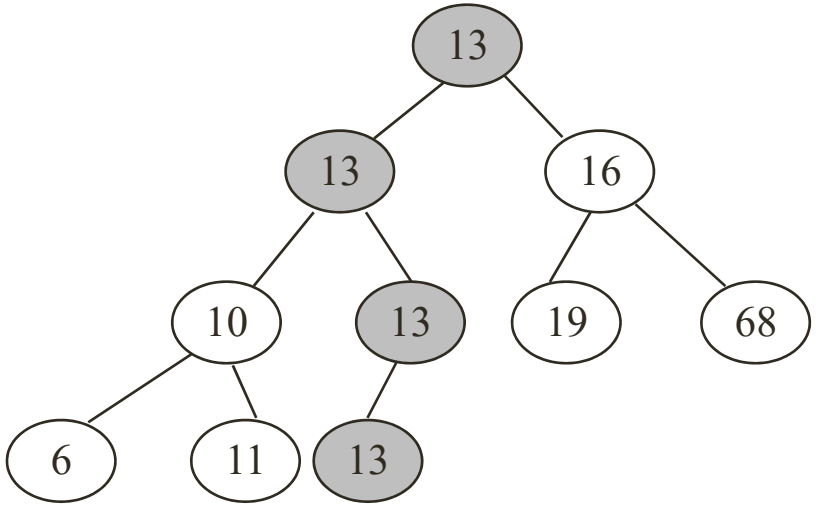
Diseña una función que compruebe si los elementos que contiene una Cola de Prioridad dada de valores Integer forman la secuencia de Fibonacci. Esta función debe dejar la Cola de Prioridad con los mismos elementos que contenía antes de la llamada.

Ejercicio 6

Diseña una función que, dado un array genérico de objetos comparables en el que todas sus posiciones están ocupadas, compruebe si dicho array cumple la propiedad de orden de un Montículo Minimal con raíz en la posición cero.

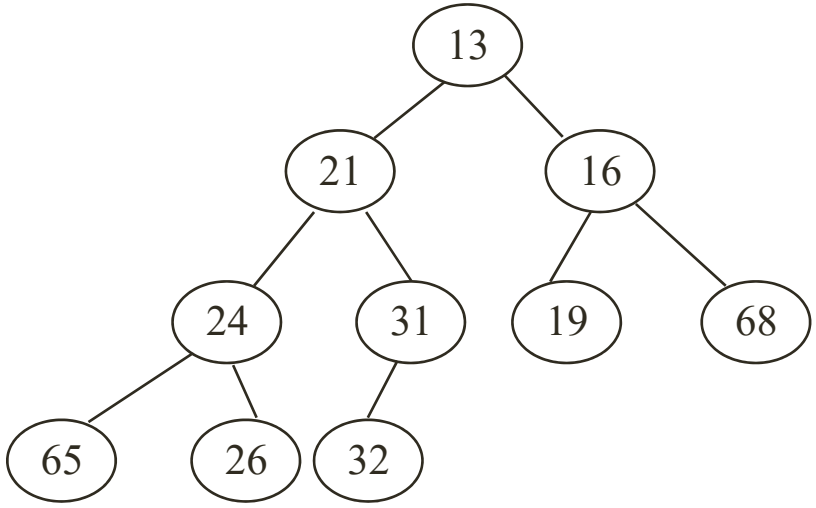
Ejercicio 7

Diseña una función en la clase MonticuloBinario que compruebe si hay un camino de la raíz a una de las hojas en el que todos nodos tengan el mismo valor.
En el siguiente Montículo, por ejemplo, sí que existiría un camino con todos sus elementos repetidos:



Ejercicio 8

Sea MonticuloBinarioInteger un montículo binario minimal con elementos de tipo Integer. Diseña en esta clase un método que elimine del montículo todas las hojas que sean pares.
Por ejemplo, en el siguiente montículo se eliminarán los nodos 68, 26 y 32:



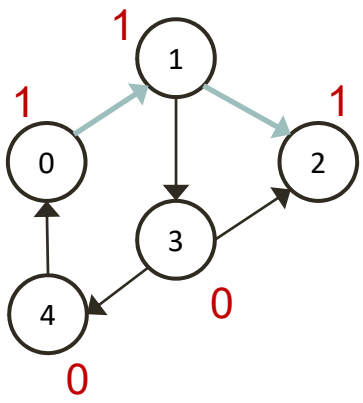
Ejercicio 9

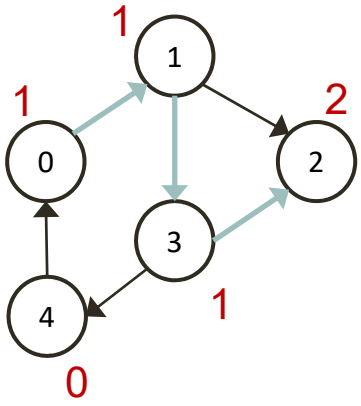
Diseñad en la clase Grafo un método *enCiclo* que compruebe si un vértice ***v*** dado forma parte de un ciclo en el grafo.

Ejercicio 10

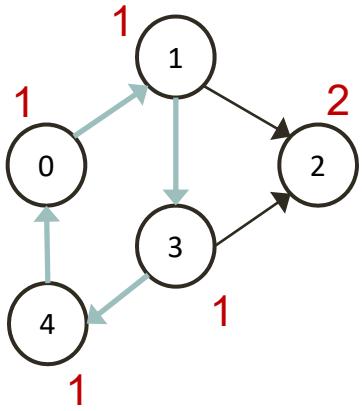
Diseña un método en la clase *Grafo* que compruebe si el grafo tiene ciclos.

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

Ejercicio 11

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 12

Diseña un método en la clase GrafoNoDirigido que devuelva el número de componentes conexas que tiene el grafo.

Ejercicio 13

Un vértice ***v*** es una raíz de un grafo acíclico si cada uno de los vértices del grafo es alcanzable desde ***v***.

Diseña una función en la clase *Grafo* que devuelva el primer vértice raíz de un grafo (o -1 si no existe tal vértice).

Ejercicio 14

En la clase ***Grafo***, implementa una función que, con el menor coste posible, devuelva el número máximo de aristas en las Componentes Conexas del grafo no dirigido.

Así, por ejemplo, para el siguiente grafo de tres componentes conexas, el método devolverá 5: en la componente de cuatro vértices hay 5 aristas, en la de dos 1 y en la de uno 0.