Para la resolución de los ejercicios, se dispone de una implementación de árbol binario a través de la clase BinTree con la siguiente especificación.

```
public class BinTree<T> {
  public BTNode<T> root;  // la raiz del árbol
  BTNode<T> current;       // referencia un nodo, se utiliza para la construcción del árbol
  Stack<T> father;       // se almacenan los antecesores del nodo current

  public BTNode<T> getRoot();// devuelve la raiz del árbol;
  // ubica el elemento pRoot como raiz del árbol
  public void setRoot(BTNode<T> pRoot);
  // devuelve la información de la raíz
  public T getRootInfo();
  public void emptyTree();  // elimina todos los nodos del árbol
  public void goLeft();  // posiciona el current en el nodo hijo izquierdo al current
  public void goRight();     // posiciona el current en el nodo hijo derecho al current
  public void goBrother();  // posiciona el current en el nodo hermano al current
  public void goRoot();       // posiciona el current en el nodo raíz
  public void insertRoot(T o);  // inserta el objeto o como raíz
   // inserta el objeto o como hijo izquierdo del current
  public void insertLeftChild(T o);
  // inserta el objeto o como hijo derecho del current
  public void insertRightChild(T o);
  public T getCurrent();  //devuelve el contenido del nodo current
}
```

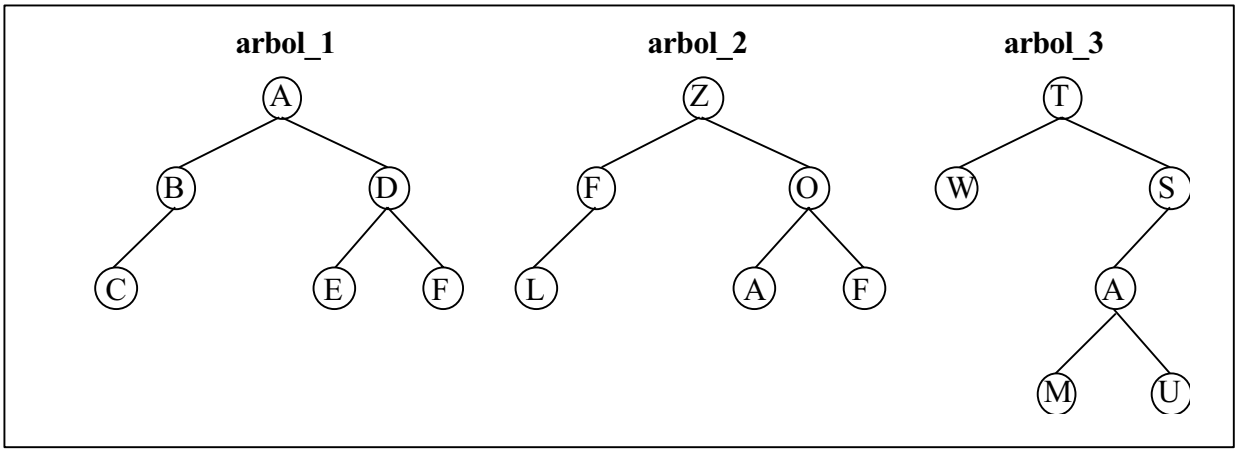
A su vez los métodos disponibles en la clase BTNode son los siguientes:

```
public class BTNode<T> {
     public T content;
     public BTNode<T> leftChild;
     public BTNode<T> rightChild;

     //Crea un nodo cuyo contenido está en theInfo
     public BTNode(T theInfo)

     // devuelve la información del Nodo
     public T getContent();
}
```

Los métodos que se pidan en todos los ejercicios se implementarán dentro de la clase BinTree

(J94) Diseñar y escribir en Java la función *mismaEstructura,* tal que dados dos árboles binarios diga si estos tienen o no la misma estructura *(misma estructura significa que los árboles son iguales, excepto los valores de los nodos).*

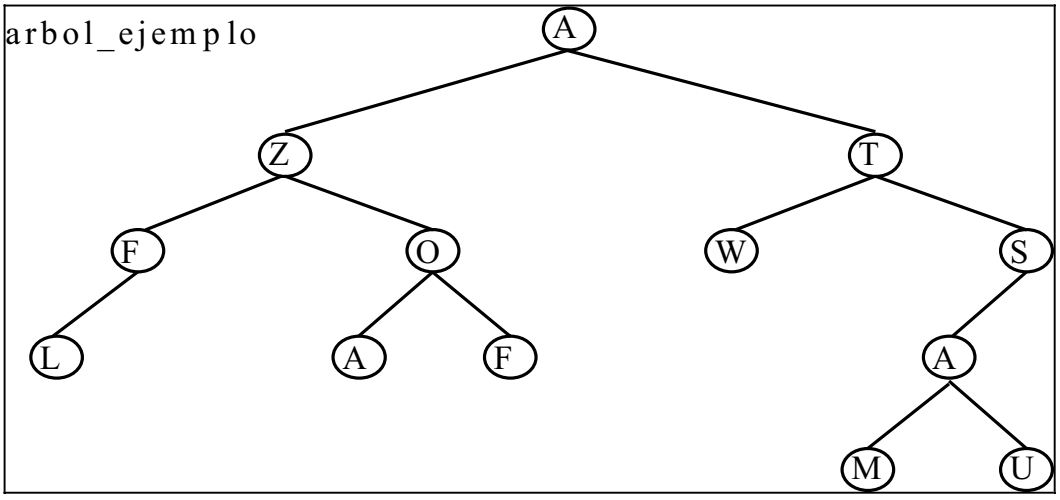
En el ejemplo de la figura *mismaEstructura(arbol_1, arbol_2)* devolvería true mientras que *mismaEstructura(arbol_1, arbol_3)* devolvería false.



(S94) Diseñar y escribir en Java la función *numNodos* tal que dado un árbol binario y un nivel (número positivo) devuelva el número de nodos que se encuentran en el árbol en dicho nivel.

En la figura, por ejemplo:

- *numNodos(arbol_ejemplo, 1)* devolvería 1
- *numNodos(arbol_ejemplo, 3)* devolvería 4
- *numNodos(arbol_ejemplo, 5)* devolvería 2

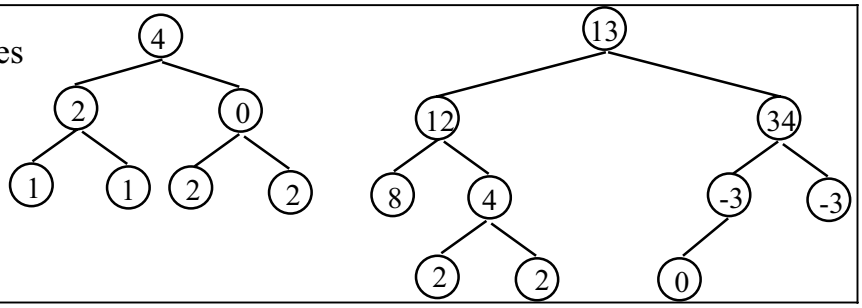


(J95) Diseñar y escribir en Java un subprograma tal que dado un árbol binario de enteros(Integer), determine si dicho árbol está *equiponderado*.

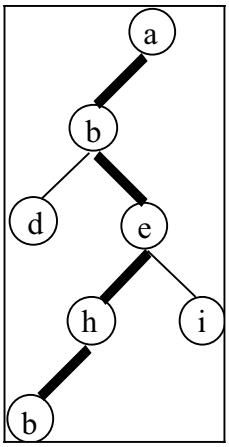
- Un árbol binario está *equiponderado* si se cumple que para todo nodo el *peso* del subárbol izquierdo es igual al *peso* del subárbol derecho.
- El árbol binario vacío está equiponderado por definición.

Llamamos *peso* de un árbol a la suma de todos sus elementos.

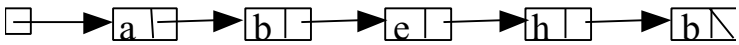
Ejemplo de árboles equiponderados:



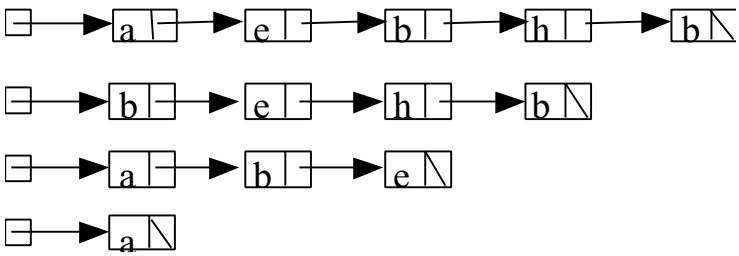
(S95) Diseñar y escribir en Java un subprograma tal que dado un árbol binario y una lista ligada doble(con su iterador DblLinkedListItr), determine si la lista **coincide exactamen** con alguna ***rama completa*** del árbol.
Una *rama del árbol* es *completa* si partiendo de la raíz acaba en alguna de las hojas.



Ejemplo de lista ligada que SI coincide exactamente con alguna de las ramas completas del árbol de la figura:



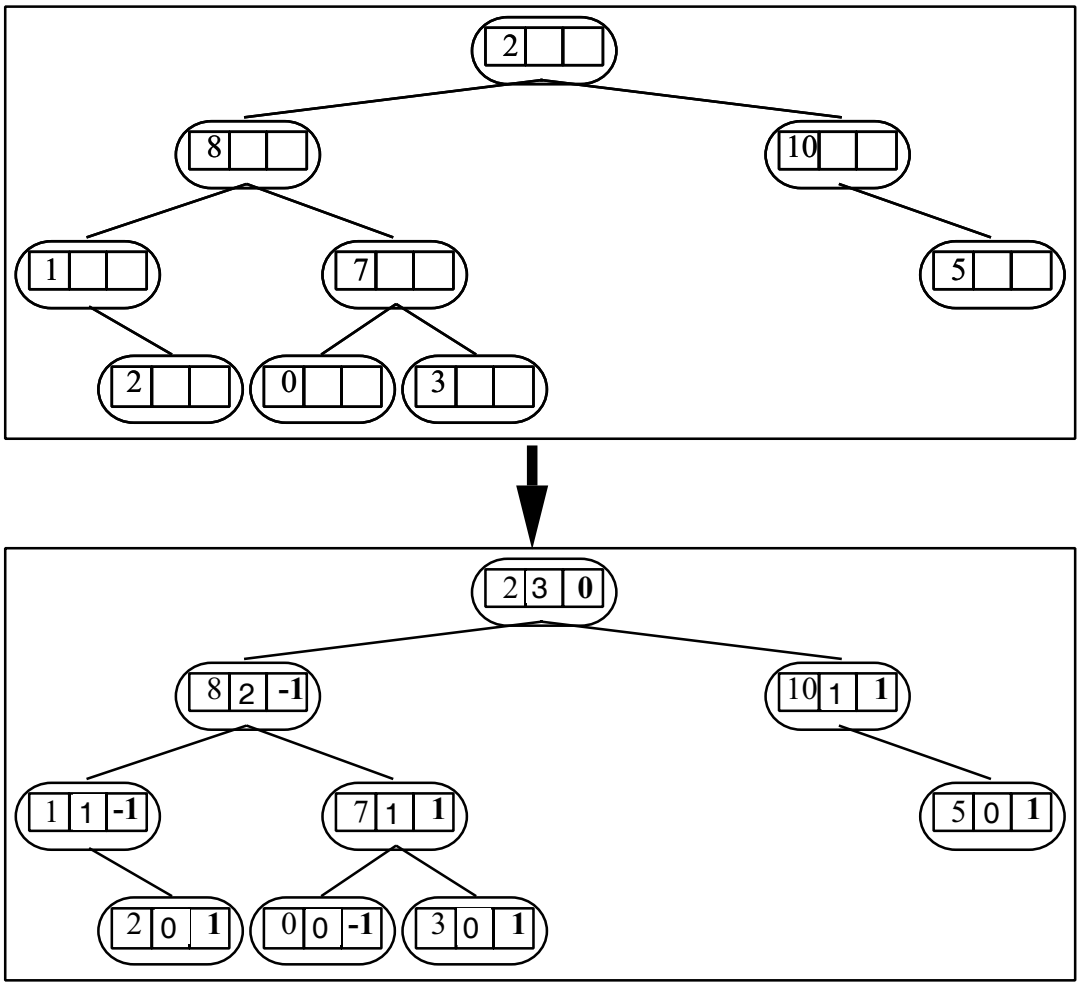
Ejemplos de listas que NO coinciden exactamente con ninguna de las ramas completas del árbol anterior:



(J96)

```
public class Nodo extends Object {

int dato;

int profundidad;

int claseDeHijo;

}
```

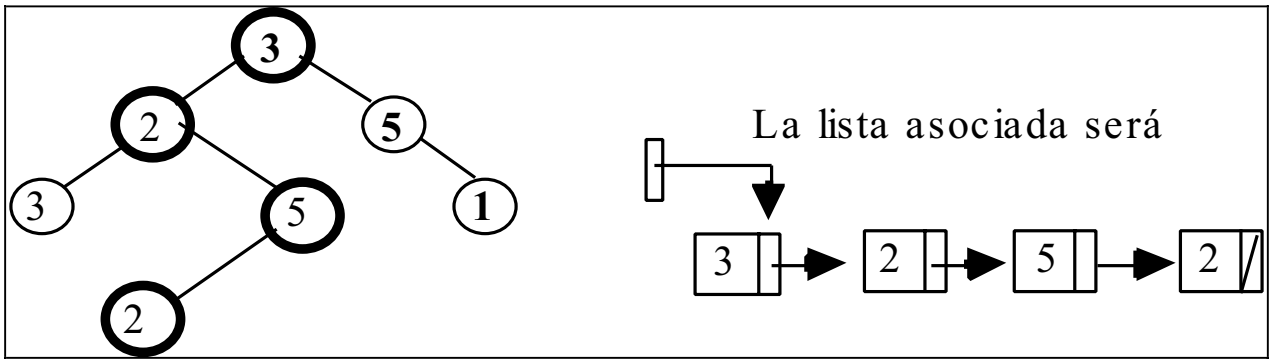
Diseñar y escribir en Java un subprograma tal que dado un árbol binario de elementos de tipo Nodo, devuelva el mismo árbol pero etiquetado. Esto es, que cada nodo del árbol contenga en su subcampo `profundidad` la información correspondiente a la profundidad del subárbol que comienza en ese nodo, y en el subcampo `claseDeHijo` el valor 0 si se trata del nodo raíz del árbol, -1 si es raíz de un subárbol izquierdo y 1 si es raíz de un subárbol derecho.

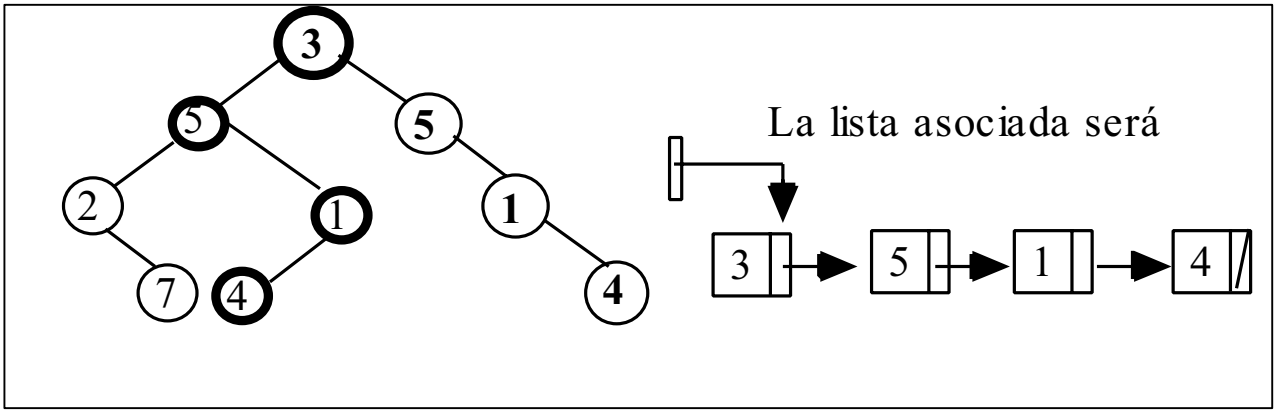


(S96)

Diseñar y escribir en Java un subprograma tal que dado un árbol binario, devuelva una lista que contenga los elementos de la rama más larga del árbol de entrada.

Si hubiera varias ramas con la misma profundidad, la lista contendría los elementos de una cualquiera de ellas (como ocurre en el ejemplo de la segunda figura).





(J97)

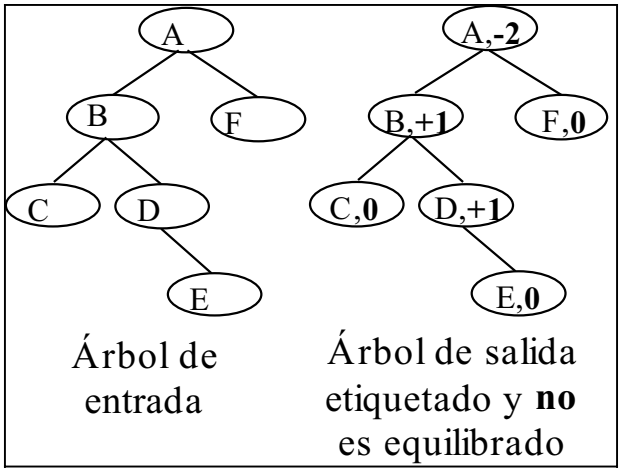
```
public class BTNode {

public int info;
public int equilibrio;
private BTNode leftChild;
private BTNode rightChild;
}
```

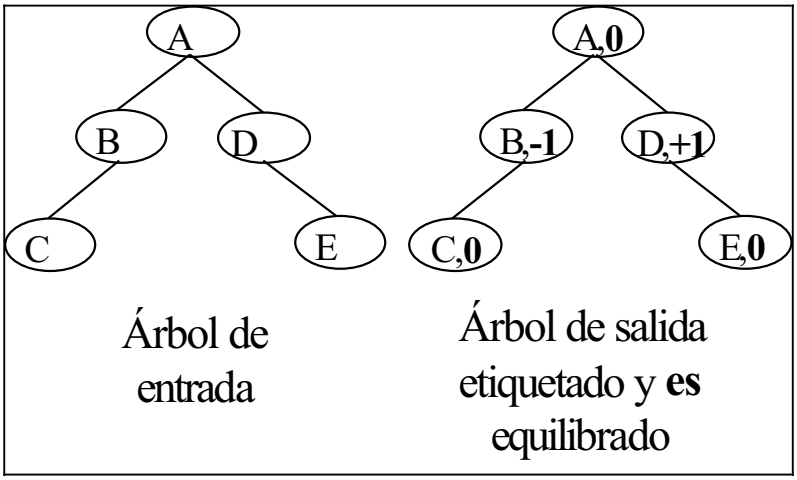
Diseñar y escribir en Java un subprograma tal que dado un árbol binario de nodos de tipo **BTNode** haga dos cosas:

1. Por un lado, etiquete el campo **`equilibrio`** de *cada nodo* de la siguiente manera:
   - Con un 0 si las profundidades de sus subárboles son iguales.
   - Con la diferencia de profundidad entre los dos subárboles. Este número será negativo si el subárbol izquierdo tiene mayor profundidad que el derecho y positivo en caso contrario.
2. Por otro lado, el subprograma **devolverá un valor booleano** True si el árbol es *equilibrado* y False en caso contrario.

Nota: se dice que un árbol es *equilibrado* si **para cada nodo** la profundidad de sus subárboles es la misma o difiere a lo sumo en 1.

Ejemplos:

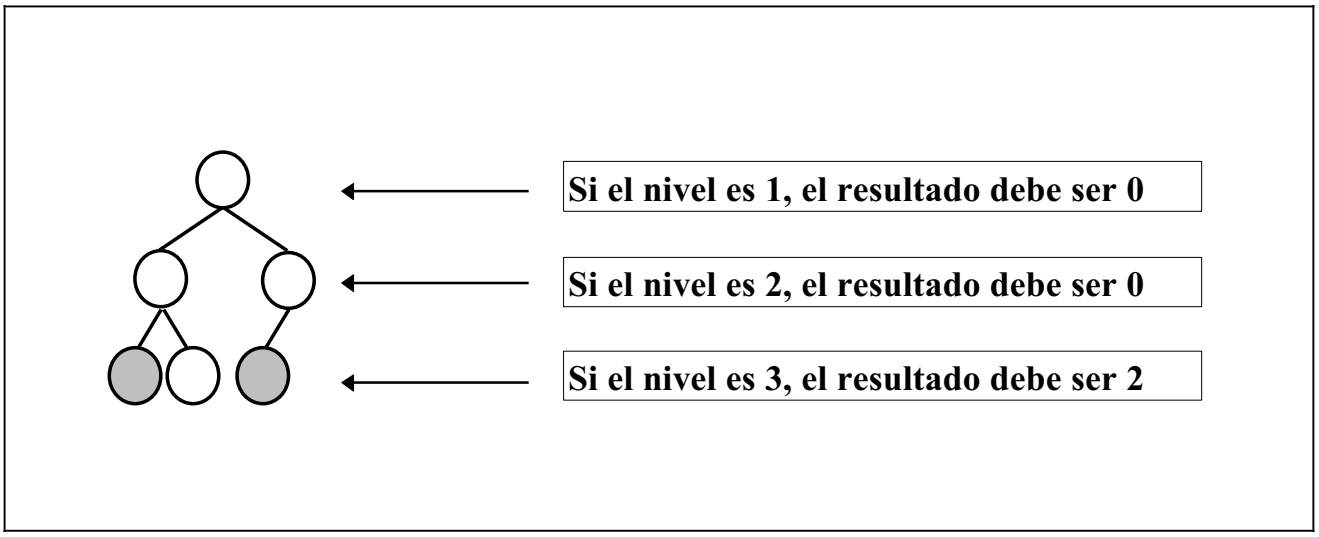


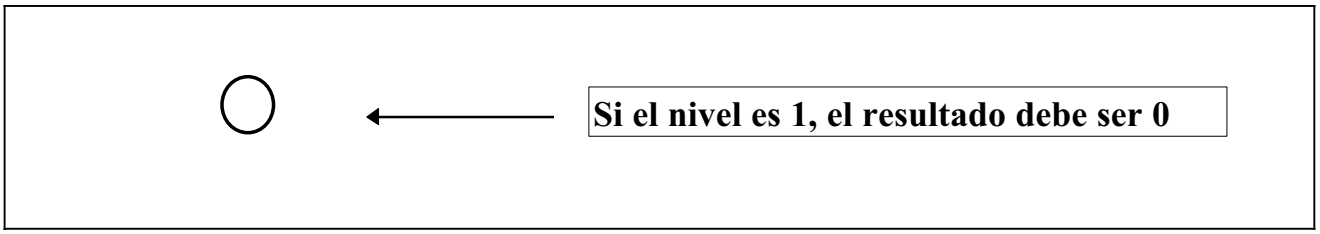


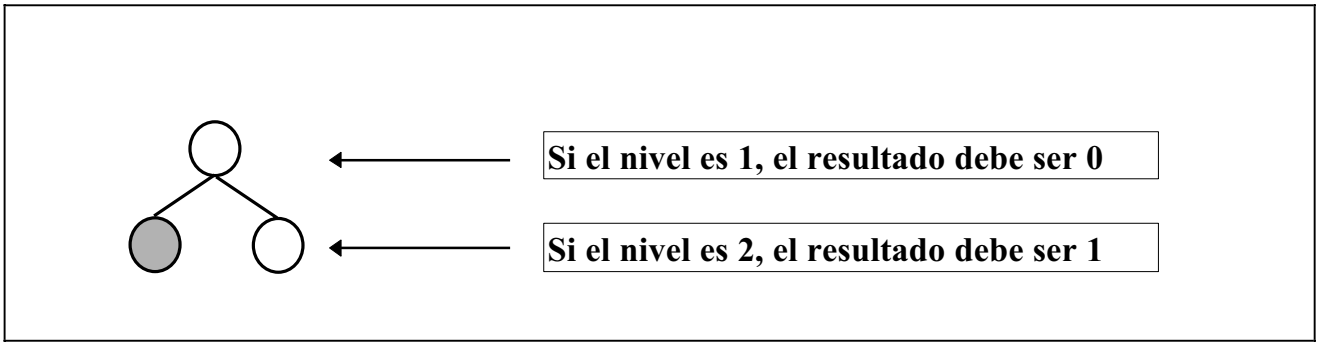
(S97) Diseñar y escribir en Java un subprograma que dado un árbol binario de elementos de tipo `char` y un nivel, devuelva el **número de nodos** que tiene dicho árbol, en el nivel dado, cumpliendo ***al mismo tiempo*** que:

- son hojas **y**
- son hijos izquierdos.

Ejemplos:







(J98)

Se dice que **un nodo** de un árbol binario tiene **grado 2** cuando los subárboles izquierdo y derecho son distintos de vacío.

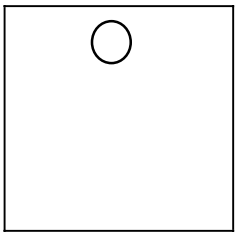
Se dice que **un árbol** binario **es 2-equilibrado** cuando bien es el árbol vacío, o bien se cumplen al mismo tiempo las siguientes condiciones:

- El subárbol izquierdo es **2-equilibrado**.
- El subárbol derecho es **2-equilibrado**.
- El número de nodos de **grado 2** en el subárbol izquierdo coincide con el número de nodos de **grado2** en el subárbol derecho.

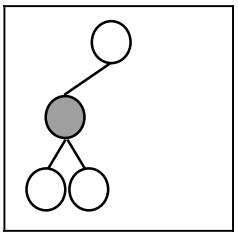
Diseñar y escribir en Java un subprograma tal que dado un árbol binario devuelva si el árbol es **2-equilibrado**.

Ejemplos:

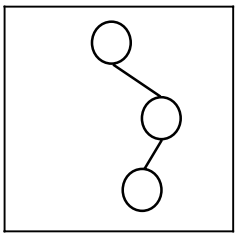
(representamos los nodos de grado 0 o 1 con ○ y los de grado 2 con ●)



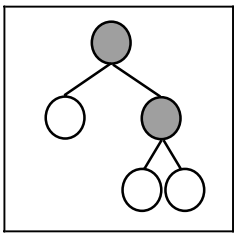
Arbol 2-equilibrado
(sin nodos de grado 2)



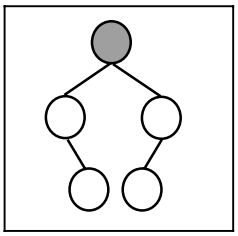
Arbol **no** 2-equilibrado
(un nodo de grado 2)



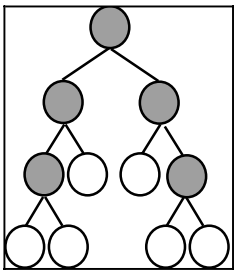
Arbol 2-equilibrado
(sin nodos de grado 2)



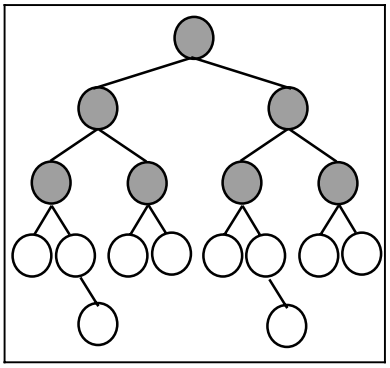
Arbol **no** 2-equilibrado
(2 nodos de grado 2)



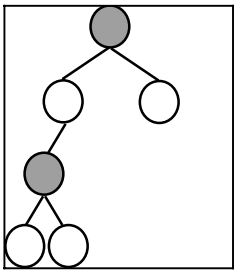
Arbol 2-equilibrado
(un nodo de grado 2)



Arbol **no** 2-equilibrado
(5 nodos de grado 2)

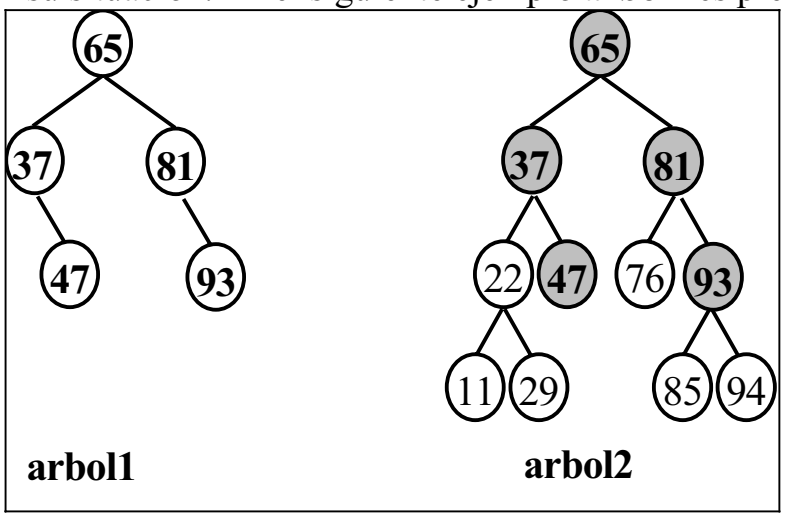


Arbol 2-equilibrado
(7 nodos de grado 2)



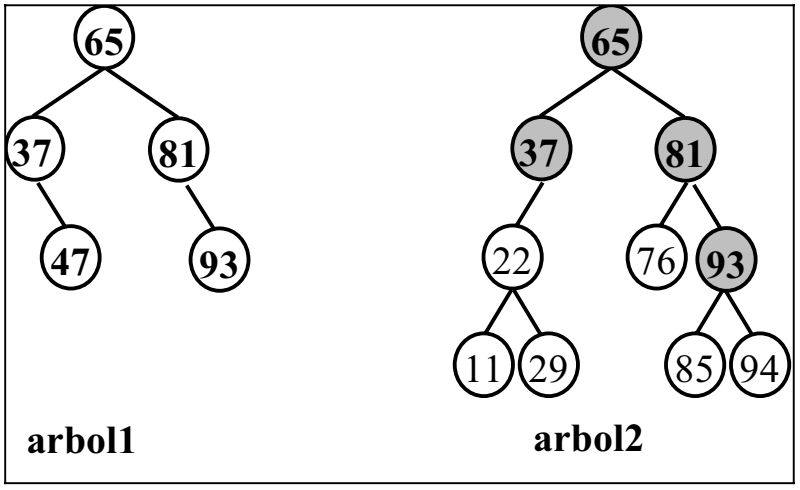
Arbol **no** 2-equilibrado
(2 nodos de grado 2)

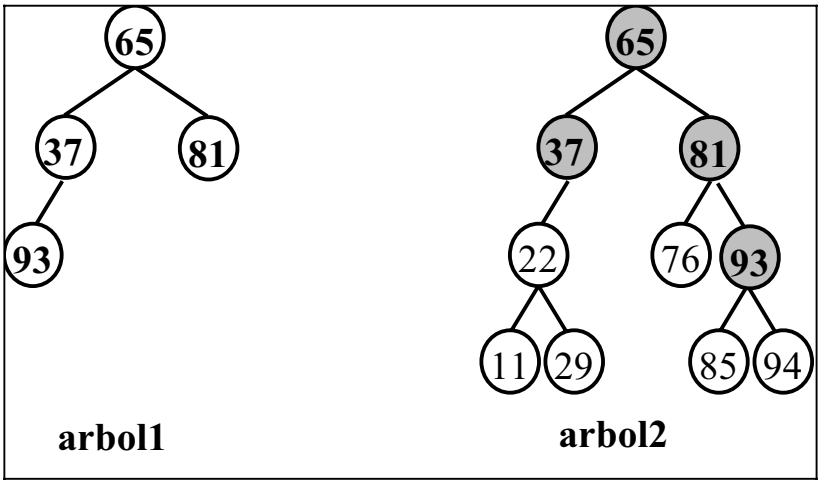
(S98)
Se dice que un árbol binario **arbol1** es prefijo de otro árbol binario **arbol2**, cuando **arbol1** coincide con la parte inicial del árbol **arbol2** tanto en el contenido de los elementos como en su situación. En el siguiente ejemplo **arbol1** es prefijo de **arbol2**:



Diseñar y escribir en Java un subprograma tal que dados dos árboles binarios, **arbol1** y **arbol2**, decida si **arbol1** es prefijo de **arbol2**.

En los siguientes ejemplos **arbol1 no** es prefijo de **arbol2**:

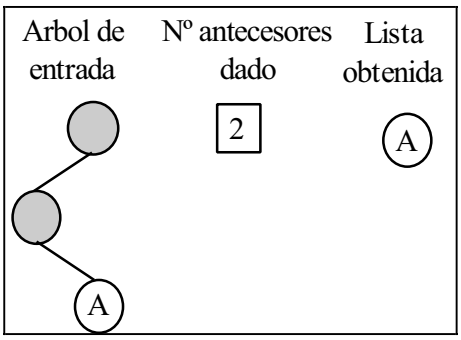


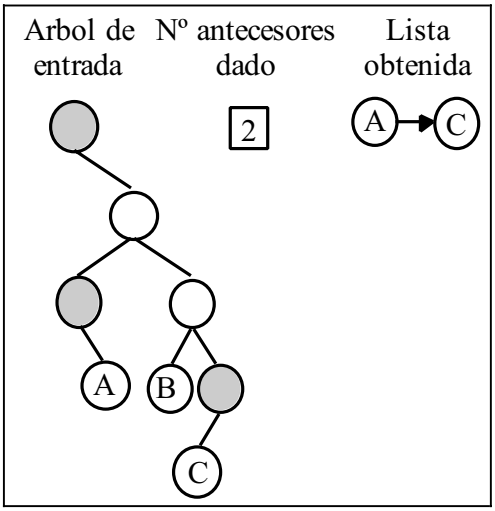


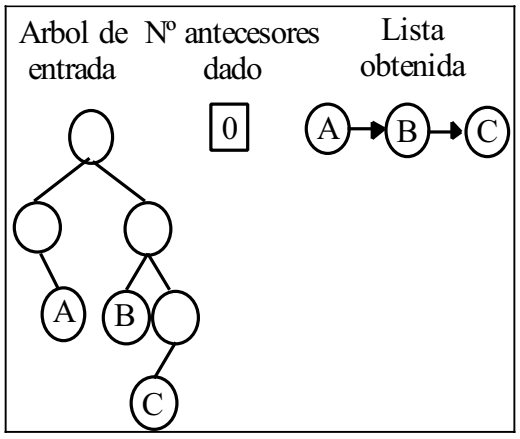
(1) Diseñar y escribir en Java un método tal que dado un árbol, **devuelva el número** de nodos **hoja** del árbol que cumplan lo siguiente:

- En la rama que une la raíz del árbol con la hoja hay un número de nodos (que nos viene dado) que tiene exactamente uno de sus subárboles vacío.

(2) Diseñar y escribir en Java un método recursivo, que devuelva la lista de los nodos hoja del árbol que cumplen los requisitos del problema anterior.







(3) Dado un árbol binario de elementos de tipo carácter, un número de nivel y un caracter, **diseña e implementa** un método Java que cuente el número de nodos de ese nivel que contiene el carácter dado.

(4) Dado un árbol binario de elementos de tipo carácter, y dado un número de nivel, **diseña e implementa** un método Java que obtenga la lista de caracteres almacenados en ese nivel del árbol.

(5) Dado un árbol binario, **diseña e implementa** un método Java que lo transforme en su imagen especular.

(6) Dado un árbol binario, **diseña e implementa** un método Java que cree un nuevo árbol que sea su imagen especular.

(7) Dados dos árboles binarios, **diseña e implementa** un método Java que decida si uno es la imagen especular del otro.

(8) Diseñar y escribir en Java un método tal que dado un árbol binario de elementos de tipo entero y un entero, devuelva **el número máximo de repeticiones** del entero que nos dan como entrada en una misma rama.

(9) Diseñar y escribir en Java un método tal que dado un árbol binario de elementos de tipo entero y un entero, devuelva **la longitud** de la rama que contiene más veces al entero que nos dan como entrada.

(10) Dado un árbol binario de números enteros, **diseña e implementa** un método Java que devuelva **la longitud** de la rama que contiene mayor peso. Se entiende como peso de una rama la suma de los enteros contenidos en sus nodos.

(11) Dado un árbol binario de números enteros, **diseña e implementa** un método Java que devuelva **el peso** que contiene la rama de mayor peso.

(12) Dado un árbol binario de números enteros, **diseña e implementa** un método Java que devuelva una lista que contenga los valores de la rama de mayor peso.

(15) Dado un árbol binario con elementos de tipo entero, **diseña e implementa** un método Java que decida si dicho árbol es ABB (árbol binario de búsqueda).

(21) Dado un árbol binario, **diseña e implementa** un método que calcule la profundidad del árbol.