# Tema 4 – Listado de Ejercicios (v1)

## Ejercicio 1

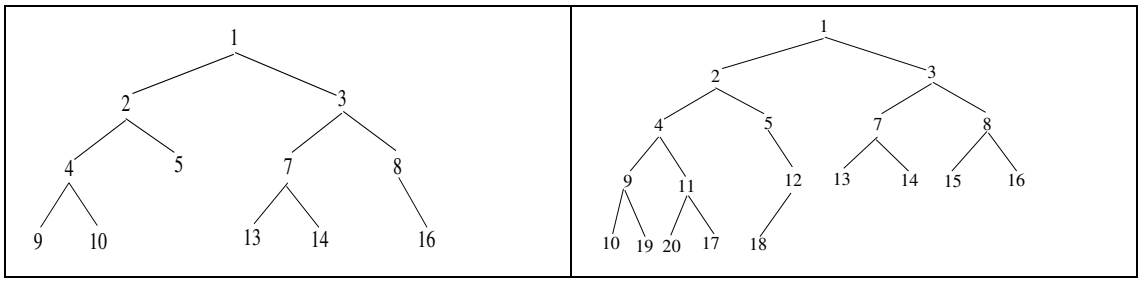
Demostra per inducció la següent propietat dels arbres binaris no buits: el nombre de fulles ($N_0$) d'un arbre binari no buit és exactament igual al nombre de nodes de grau 2 ($N_2$) més 1.

## Ejercicio 2

Arbres binaris equilibrats:

Quina és la propietat principal o característica dels arbres binaris equilibrats?

Són equilibrats els següents arbres binaris? Raona la teva resposta.




## Ejercicio 3

Dissenya un mètode recursiu que determini si un element donat es troba a un arbre arbre binari.

Feu dues implementacions:

- Com un mètode de la classe LinkedBinaryTree. Com aquesta operació pertany a la seva superinterfície Collection, i està definida rebent un element de tipus Object, la definició que fem haurà de respectar aquest tipus.
- Una en una classe apart, en un mètode estàtic, que rebi un BinaryTree i un Object (per mantenir la coherència amb l'anterior).
    - Hi ha dues maneres de declarar aquesta funció estàtica (usant paràmetre genèric, usant un comodí). La implementació serà la mateixa en tots dos casos.

**NOTA:** Altres anys es definia BinaryTree com a Collection. Ara ja no ho fem, però podeu resoldre l'exercici igualment eliminant l'@Override.

## Ejercicio 4

A la part de teoria, hem vist els principals recorreguts en arbres binaris i la seva implementació de manera recursiva. En aquesta pregunta, et demandem que implementis la versió iterativa del recorregut en *pre-ordre*.

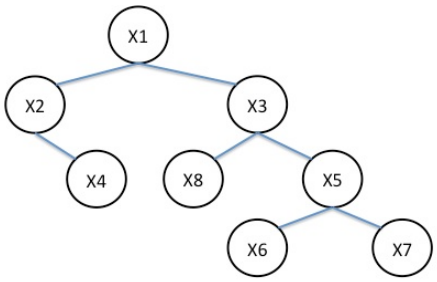
***public static <E> List<E> preOrderIteratiu (BinaryTree<E> arbre)***

Aquest mètode treballa amb els arbres binaris que hem definit a l'assignatura i retorna una llista. El mètodes de *BinaryTree* que pots necessitar són:

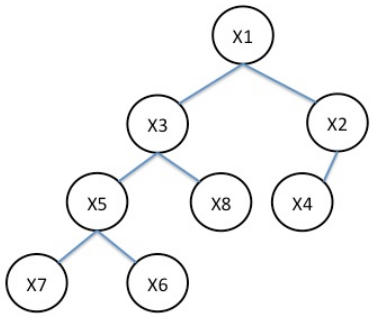
- *public BinaryTree<E> left()*: retorna l'arbre binari del fill esquerre de l'arrel de l'arbre.
- *public BinaryTree<E> right()*: retorna l'arbre binari del fill dret de l'arrel de l'arbre.
- *public E root()*: retorna l'atribut *element* del node arrel.
- *public boolean isEmpty()*: retorna cert si el nombre de nodes és 0. Retorna fals en cas contrari.

## Ejercicio 5

Dissenya un mètode recursiu que ens permeti intercanviar els fills esquerre i dret de cada node en un arbre binari. És a dir, si tenim l'arbre binari següent:




L'arbre resultant després de cridar al nostre mètode és:



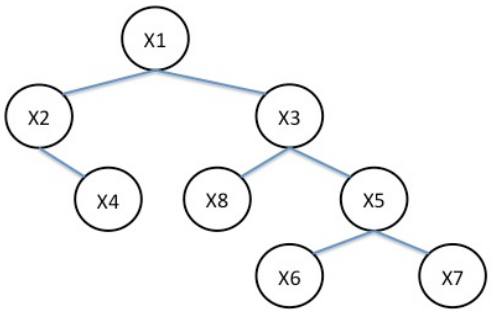

Feu dues implementacions:

- Com un mètode de la classe LinkedBinaryTree, sense paràmetres, i que modifiqui l'arbre receptor.

- Una en una classe apart, en un mètode estàtic, que rebi un BinaryTree i retorni un nou LinkedBinaryTree amb el resultat.

## Ejercicio 6

Dissenya un mètode recursiu que ens permeti calcular el nombre de descendents (no *null*) de qualsevol node d'un arbre binari. Per descendents ens referim a tots els nodes de grau 2, grau 1 i grau 0 que estan en nivells immediatament inferiors. És a dir, si tenim l'arbre binari següent ...




... i calculem el nombre de descendents del node X3, el resultat és 4. Els seus descendents són: {X8, X5, X6, X7}. El nombre de descendents de X2 és 1, {X4}. El nombre de descendents de X4 és 0.

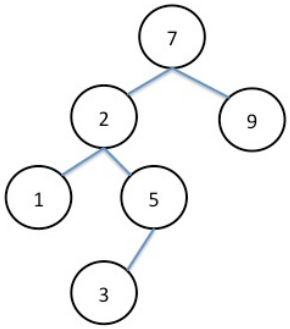
Implementeu-lo com un mètode de la classe LinkedBinaryTree (en aquest cas no es pot implementar en una en una classe apart, ja que els nodes de l'arbre són d'una classe interna privada). Podeu suposar com a precondició que la referència que ens passen és no nul·la i es correspon a un node de l'arbre.

## Ejercicio 7

Demostra per inducció la següent propietat dels arbres binaris no buits: el nombre de fulles ($N_0$) en un arbre binari complet de N nodes és exactament: $N_0 = (N+1) / 2$

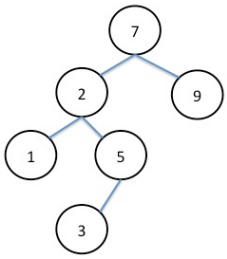
## Ejercicio 8

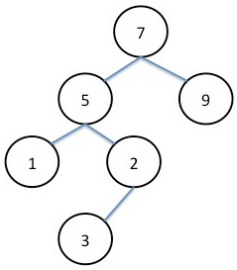
La següent figura mostra un Arbre Binari de Cerca. Quin dels tres recorreguts que hem vist a l'assignatura (*inordre, postordre, preordre*) ens retorna els elements ordenats de més petit a més gran? Raona la teva resposta.




## Ejercicio 9

Dissenya un mètode recursiu que ens permeti comprovar si hem creat correctament un Arbre Binari de Cerca (ABC). És a dir, si hem creat els següents arbres ...




... l'arbre de l'esquerra és un ABC, però l'arbre de la dreta no ho és, ja que incompleix la propietat principal dels ABC: si ens fixem en el node 5, el seu fill dret és 2, que no és més gran que 5.

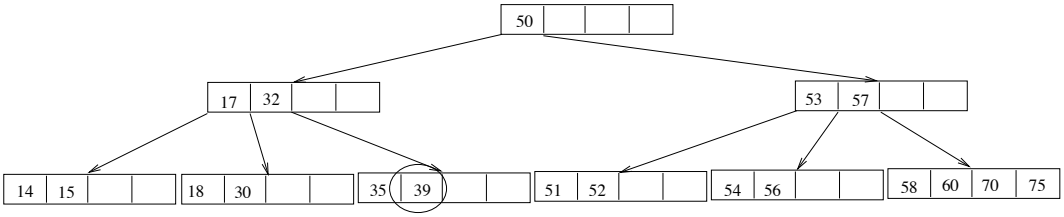
```
public class TreeOps {

    public static <E extends Comparable<? super E>>
    boolean isBST(BinaryTree<E> tree) { ¿? }
}
```

Fixeu-vos que per fer les comparacions, el mètode aprofitarà el fet que E és del tipus *Comparable*.

## Ejercicio 10

Considera l'arbre B de la figura:

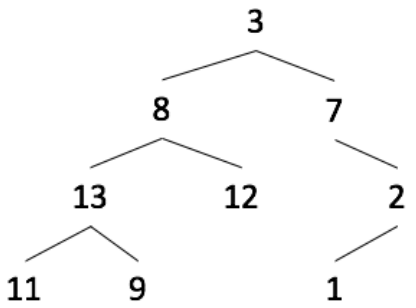



Quin és l'ordre d'aquest arbre B?

Elimina la clau 39 de l'arbre B aplicant l'algoritme d'eliminació següent (el que hem vist a l'assignatura) i mostra (a) les passes de l'algoritme d'eliminació pas a pas i (b) el resultat final.

## Ejercicio 11

Implementa el método recursivo *sumLeaves()*, que reciba un *BinaryTree<Integer>*, y que calcule la suma de las hojas de un árbol binario de enteros. Si el árbol está vacío, devolverá 0. En otro caso, por ejemplo, en el de la de la figura, devolverá 33.

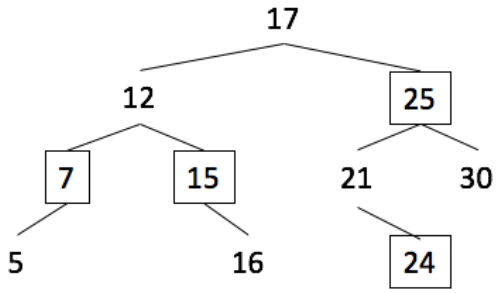



Árbol binario del ejercicio 11

## Ejercicio 12

Explica cómo se elimina un nodo en un árbol binario de búsqueda. Para hacer tu explicación básate en el árbol de la figura, mostrando cómo se eliminan los elementos 7, 15, 24 y 25, **partiendo siempre del árbol original**.

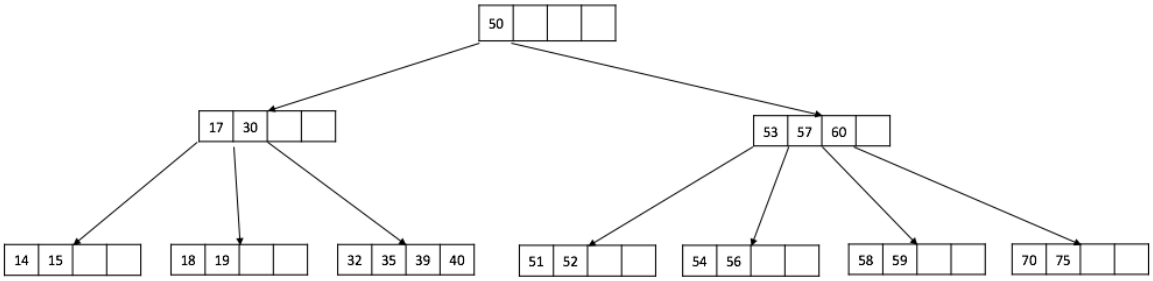
*Nota 1: no se pide que lo implementes si no que lo expliques en detalle.*
*Nota 2: suponemos que el árbol binario de búsqueda no tiene elementos repetidos.*




Ejemplo del ejercicio 12

## Ejercicio 13

Considera el siguiente árbol B:




Árbol B del ejercicio 13

Realiza las siguientes operaciones:

- Elimina del árbol B original la clave 70.
- Inserta en el árbol B original la clave 41.

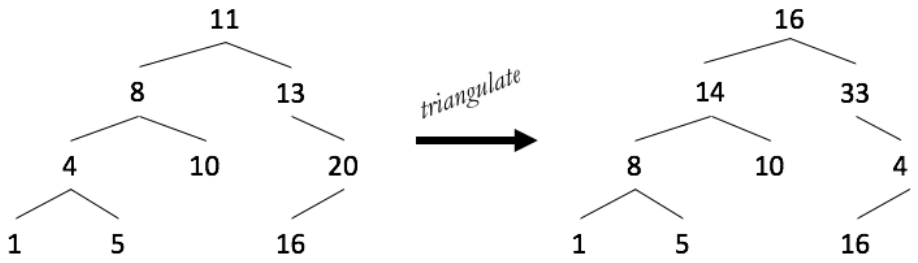
Explica en cada parte de esta pregunta (eliminación e inserción) las situaciones que se dan en cada momento. Debes explicar el proceso seguido, resultado de cada paso necesario para llegar al resultado final.

## Ejercicio 14

Implementa el método recursivo *triangulate*, que reciba un *BinaryTree<Integer>*, y que lo modifique de manera que cada nodo obtendrá como nuevo valor el resultado de la siguiente operación:

*valor_nuevo = valor_anterior – valor_raíz_hijo_izquierdo + valor_raíz_hijo_derecho*
*Si el hijo izquierdo es vacío el valor a restar es 0.*
*Si el hijo derecho es vacío el valor a sumar es 0.*

El método debe devolver un *LinkedBinaryTree<Integer>*. Si el árbol está vacío, devolverá vacío. En otro caso, por ejemplo, el resultado será el de la figura.




Ejemplo relativo al ejercicio 14

## Ejercicio 15

Dada una tabla implementada con un árbol binario de búsqueda, dibuja paso a paso al estilo de la figura del ejercicio 14, cómo queda la tabla (en forma de árbol binario de búsqueda) con las operaciones indicadas en la figura. Además, es necesario explicar brevemente qué se realiza en cada situación nueva, es decir, la primera vez que sucede una situación (no ocurrida previamente) se explica, después no es necesario.

| | | |
|---|---|---|
| 1ª Operación -> Insertar 97 | 6ª Operación -> Eliminar 34 | 11ª Operación -> Eliminar 99 |
| 2ª Operación -> Insertar 34 | 7ª Operación -> Insertar 38 | 12ª Operación -> Insertar 138 |
| 3ª Operación -> Insertar 100 | 8ª Operación -> Insertar 32 | |
| 4ª Operación -> Insertar 99 | 9ª Operación -> Insertar 37 | |
| 5ª Operación -> Insertar 35 | 10ª Operación -> Eliminar 35 | |

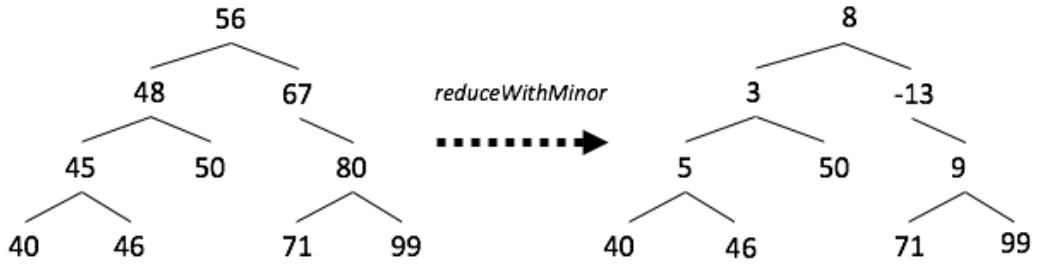
Operaciones del ejercicio 15

## Ejercicio 16

Implementa un método, llamado *reduceWithMinor*, que reciba un *BinaryTree<Integer>* y devuelva un *LinkedBinaryTree<Integer>* tal que, en cada posición del árbol, contenga el resultado de restar al nodo equivalente del árbol inicial, el valor del menor de sus hijos. En concreto, la operación para obtener el valor de cada nodo será (ver figura):

*valor_nuevo = valor_anterior – valor_menor_de_sus_hijos*
*Si el hijo izquierdo y derecho son vacíos el valor a restar es 0*
*Si el árbol recibido es vacío, la operación devuelve un árbol vacío.*

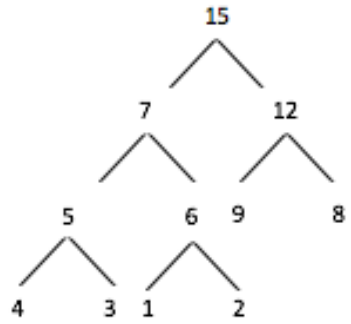
**Importante**: Tened en cuenta el orden de las operaciones, tal y como se puede ver en el ejemplo de la figura.




Ejemplo relativo al ejercicio 16

## Ejercicio 17

Dado el siguiente max-heap:




Max-heap del ejercicio 17

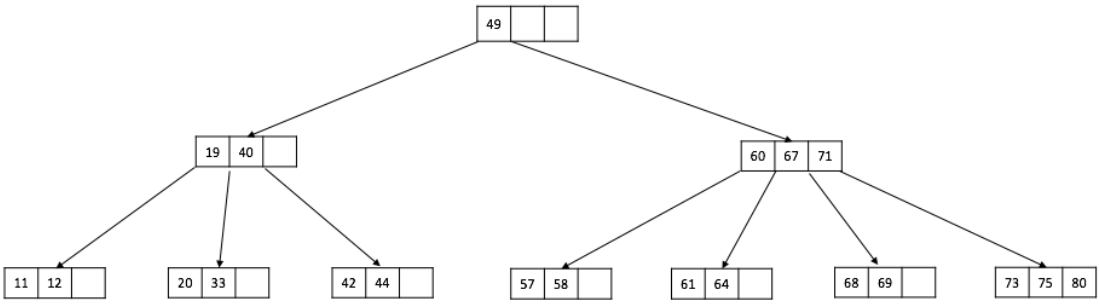
Realiza las siguientes operaciones:

- *Extract* sobre el max-heap original, que devuelve el elemento mayor.
- *Insert del elemento 14* sobre el max-heap original.

Explica en cada parte de esta pregunta (extract e insert) las situaciones que se dan en cada momento. Debes explicar el proceso seguido, resultado de cada paso necesario para llegar al resultado final.

## Ejercicio 18

Considera el siguiente árbol B:




Árbol B del ejercicio 18

Realiza la siguiente operación:

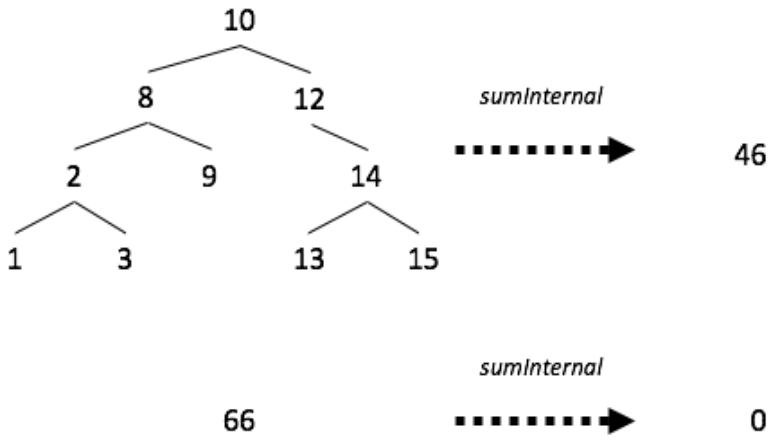
- Inserta en el árbol B original la clave 83.

Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, resultado de cada paso necesario para llegar al resultado final.

## Ejercicio 19

Implementa un método, llamado *sumInternal*, que reciba un *BinaryTree<Integer>* y devuelva la suma de todos los nodos interiores (no hojas). Un ejemplo de uso del método es el mostrado por la figura.

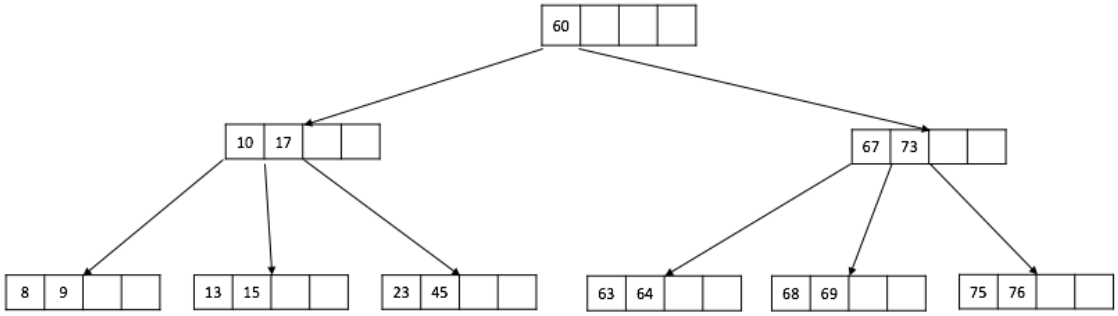
| |
|---|
| *Si el árbol es vacío devuelve 0*<br>*Si el árbol sólo está compuesto por su raíz devuelve 0* |




Ejemplo relativo a la operación *sumInternal* (ejercicio 19)

## Ejercicio 20

Considera el siguiente árbol B:




Árbol B del ejercicio 20

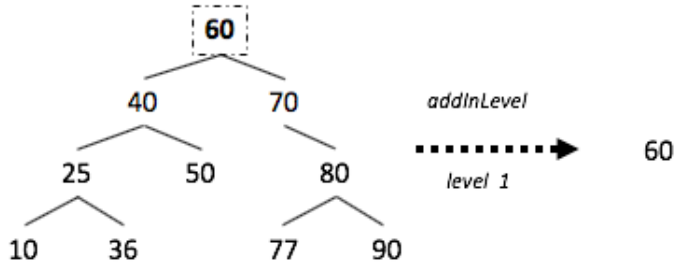
Realiza la siguiente operación:

- Elimina en el árbol B original la clave 45.

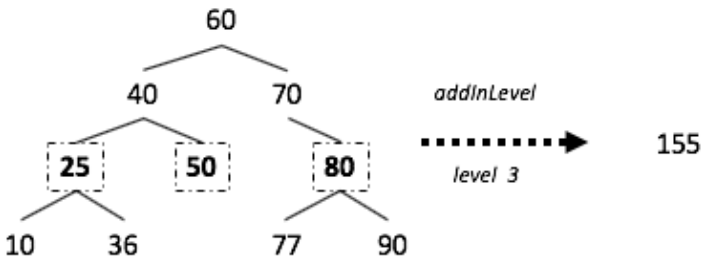
Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, resultado de cada paso necesario para llegar al resultado final.

## Ejercicio 21

Implementa un método, llamado *addInLevel*, que reciba un *BinaryTree<Integer>* y un entero que represente un nivel del árbol recibido tal que, devuelva la suma de los valores de los nodos situados en dicho nivel. Por ejemplo, si se recibe el árbol de la primera figura y el nivel indicado es el 1, el resultado devuelto por *addInLevel* será 60. Del mismo modo, si se recibe el árbol de la segunda figura y el nivel indicado es el 3, el resultado devuelto por *addInLevel* será 155.




Ejemplo 1 relativo a la operación *addInLevel* del ejercicio 21 donde quedan resaltados los valores implicados en la operación

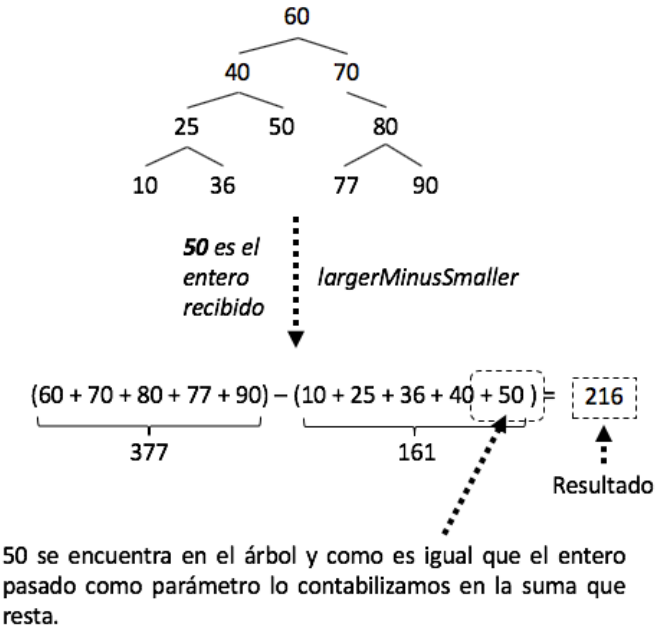



Ejemplo 2 relativo a la operación *addInLevel* del ejercicio 21 donde quedan resaltados los valores implicados en la operación

Considerando *t* el árbol y *n* el nivel como parámetros del método *addInLevel*, si no se cumple que 1 <= n <= altura(t) se devuelve 0. En el caso de recibir un árbol vacío, *addInlevel* devolverá 0 para cualquier valor de *n*.

**NOTA:** Altres anys (i aquest és un problema d'un examen anterior) es definia l'alçada de l'arbre buit com a zero i, per tant, el nivell de l'arrrel era l'1. Podeu adaptar l'enunciat i la solució a la nova situació actual o resoldre'l tal i com està plantejat.

## Ejercicio 22




Ejemplo relativo a la operación *largerMinusSmaller* del ejercicio 22

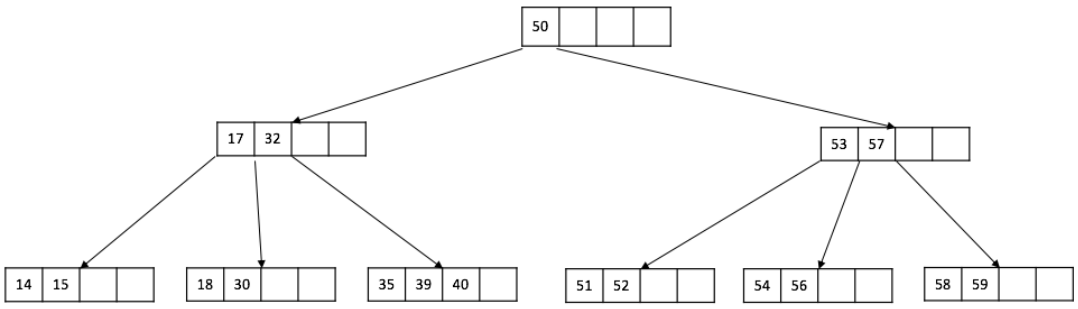
Implementa un método, llamado *largerMinusSmaller*, que reciba un *BinaryTree<Integer>* y un entero, y devuelva el resultado de restar a la suma de los números (contenidos en el árbol) que sean mayores que el entero recibido, la suma de los números (contenidos en el árbol) que sean menores o iguales que dicho entero. Por ejemplo, si se recibe el árbol de la figura y el entero indicado es el 50, el resultado devuelto por *largerMinusSmaller* será 216.

En el caso de recibir un árbol vacío, *largerMinusSmaller* devolverá 0 para cualquier entero recibido sobre el que realizar l

## Ejercicio 23

Considerando el árbol B de la figura, primero elimina la clave 32 y después, sobre el árbol B resultante elimina la clave 40.

Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, resultado de cada paso necesario para llegar al resultado final.



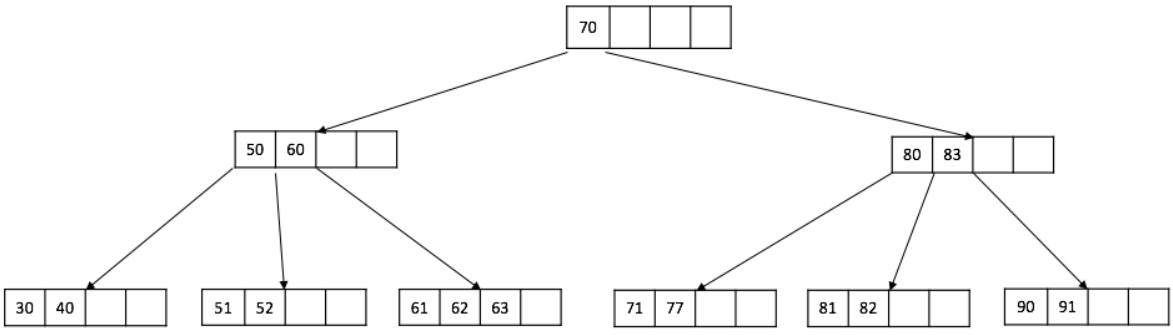

Árbol B del ejercicio 23

## Ejercicio 24

Considerando el árbol B de la figura, realiza las siguientes operaciones:

- Elimina la clave 91 sobre el árbol B original.
- Eliminar la clave 52 sobre el árbol B original.

Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final.




Árbol B del ejercicio 24

## Ejercicio 25

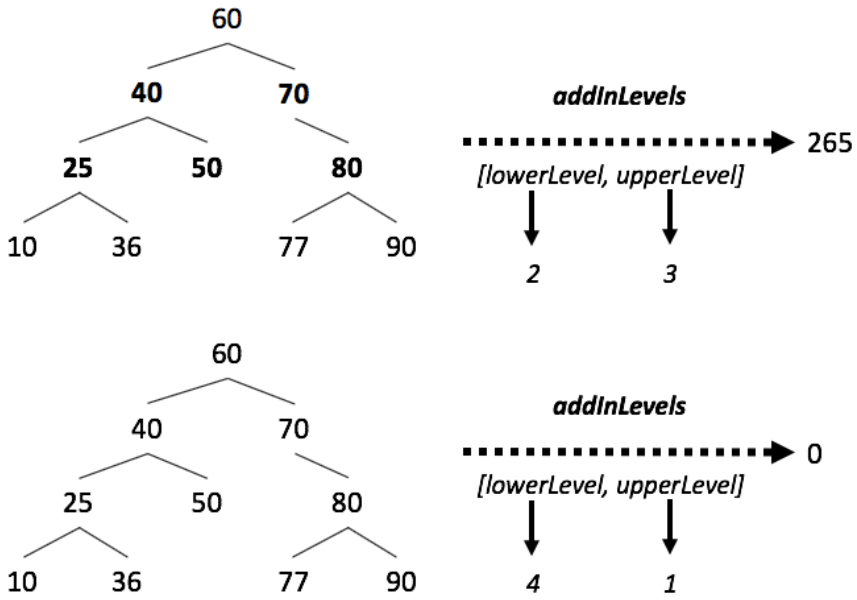
Implementa un método de forma recursiva, llamado *addInLevels*, que reciba un *BinaryTree<Integer>* y devuelva la suma de todos los nodos contenidos entre dos niveles pasados como parámetros (también se incluyen en la suma los nodos de ambos niveles). Será importante tener en cuenta que el primer nivel, en el que se encuentra la raíz, será el nivel 1.

Como se puede entender, los niveles recibidos como parámetros definen el rango donde encontrar los nodos a sumar. En este sentido, uno de los niveles recibidos definirá el límite inferior del rango y se llamará *lowerLevel*. El otro definirá el límite mayor del rango y se llamará *upperLevel*. Así, avanzando en los niveles del árbol, si el nivel actual está en el rango [*lowerLevel, upperLevel*] se suman los nodos de ese nivel y si no, no se suman.

Dado que existen algunas situaciones a considerar en la implementación en función de los parámetros recibidos, a continuación, se indican una serie de comportamientos en los que el método devolverá 0:

- Si el árbol recibido es vacío.
- Si *upperLevel* es menor que *lowerLevel*. Esto supone que se ha pasado un rango vacío.

En la siguiente figura hay dos ejemplos del método:




Ejemplos relativos a la operación *addInLevels* del ejercicio 25

Finalmente, como ayuda para realizar la solución, será recomendable hacer uso de una función auxiliar que, entre otros parámetros, tenga el nivel actual

sobre el que se está trabajando. Es decir, si estamos trabajando sobre el nivel 2, en un momento determinado, será necesario saberlo.

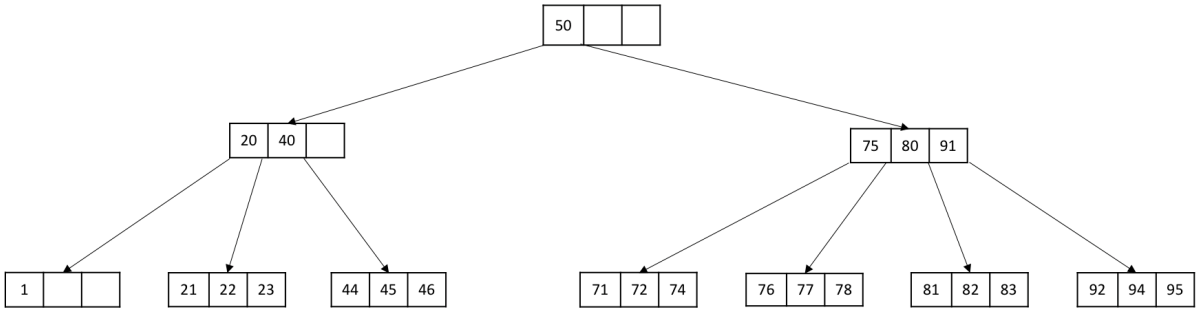
**NOTA:** Altres anys (i aquest és un problema d'un examen anterior) es definia l'alçada de l'arbre buit com a zero i, per tant, el nivell de l'arrrel era l'1. Podeu adaptar l'enunciat i la solució a la nova situació actual o resoldre'l tal i com està plantejat.

## Ejercicio 26

Considerando el árbol B de la figura, realiza las siguientes operaciones:

- Añadir la clave 99 sobre el árbol B original.
- Eliminar la clave 1 sobre el árbol B original.

Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final. La explicación es necesaria para que la solución sea válida.



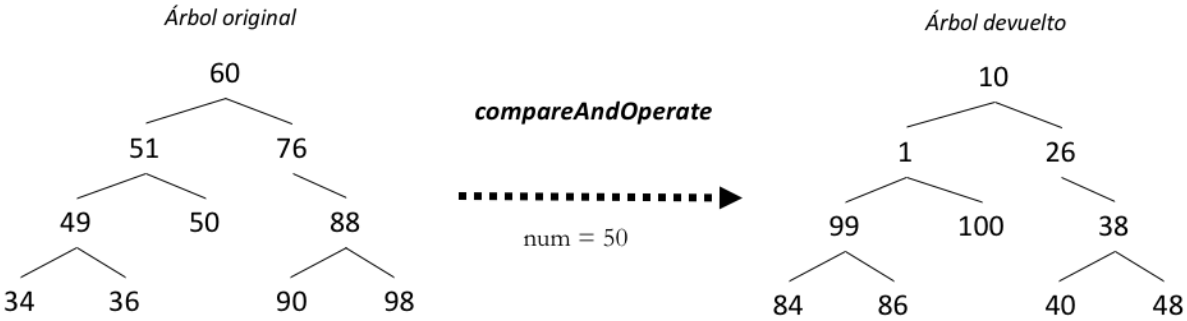

Árbol B del ejercicio 26

## Ejercicio 27

Implementa un método, llamado *compareAndOperate*, que reciba un *BinaryTree<Integer>* (llamado *original)* junto a un entero (llamado *num*) y devuelva un *LinkedBinaryTree<Integer>* tal que, los valores de sus nodos quedarán definidos en base al valor del nodo equivalente en el árbol *original* y las siguientes condiciones (la figura muestra un ejemplo del uso de *compareAndOperate*):

- Si el valor del nodo en el árbol *original* es menor o igual que *num* entonces el nuevo valor será la suma del valor del nodo en el árbol original y el entero recibido como parámetro (*num).*
- Si el valor del nodo en el árbol *original* es mayor que *num* entonces el nuevo valor será la resta del valor del nodo en el árbol original y el entero recibido como parámetro (*num).*
- Si el árbol es vacío, la operación devuelve un árbol vacío.

La implementación del método debe ser realizada **fuera** de la clase *LinkedBinaryTree<E>*.



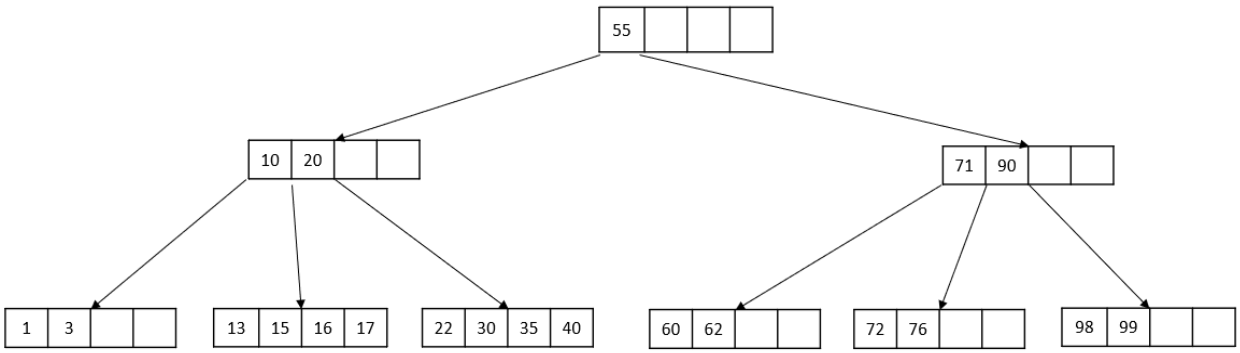

Ejemplo relativo a la operación *compareAndOperate* del ejercicio 27

## Ejercicio 28

Considerando el árbol B de la figura, realiza las siguientes operaciones:

- Añadir la clave 12 sobre el árbol B original.
- Eliminar la clave 62 sobre el árbol B original.

Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final.




Árbol B del ejercicio 28

## Ejercicio 29

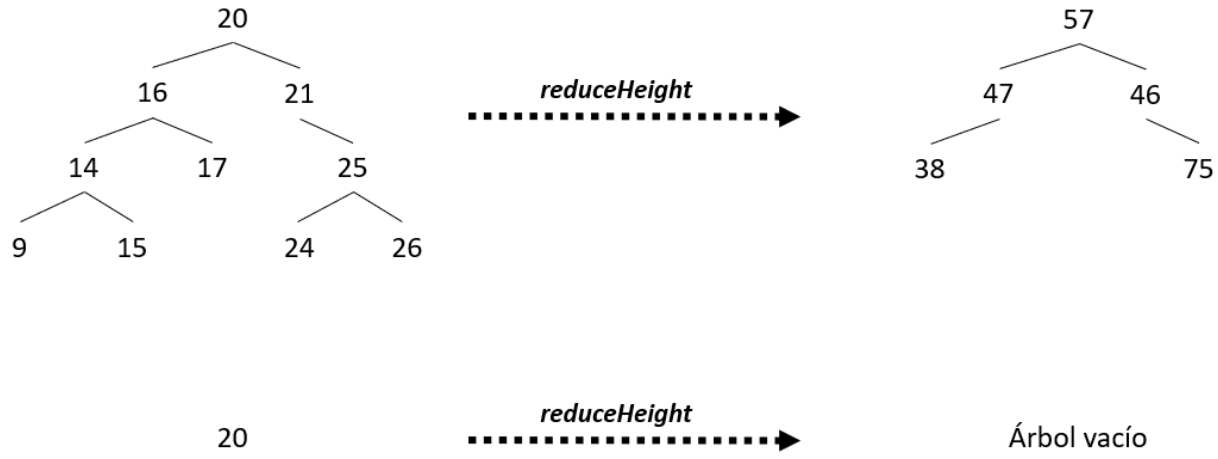
Implementa un método, llamado *reduceHeight*, que reciba un *BinaryTree<Integer>* (llamado *original*) y devuelva un *LinkedBinaryTree<Integer>* en el que cada posición del nuevo árbol contenga el resultado de sumar al nodo equivalente del árbol inicial, el valor de sus hijos también en el árbol inicial. En el caso de una hoja en el árbol original, su equivalente en el nuevo

árbol será vacío. Por lo tanto, el nuevo árbol reducirá en uno la altura del árbol original.

Si el árbol recibido como parámetro es vacío, la operación devolverá un árbol vacío. Para la realización del ejercicio el árbol recibido nunca será *null*.

Con el fin de tener más clara la operación a realizar, la figura contiene un ejemplo de uso del método *reduceHeight*.

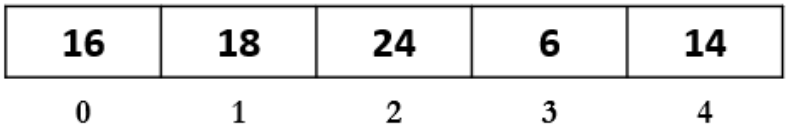
La implementación del método debe ser realizada **fuera** de la clase *LinkedBinaryTree<E>*.




Ejemplo relativo a la operación reduceHeight del ejercicio 29

## Ejercicio 30

Dado el array de cinco posiciones de la figura, se pide ordenarlo (de menor a mayor) por medio del algoritmo HeapSort. No se pide realizar la implementación, se pide explicar textual y gráficamente (diagramas) el proceso siendo ambas explicaciones necesarias.



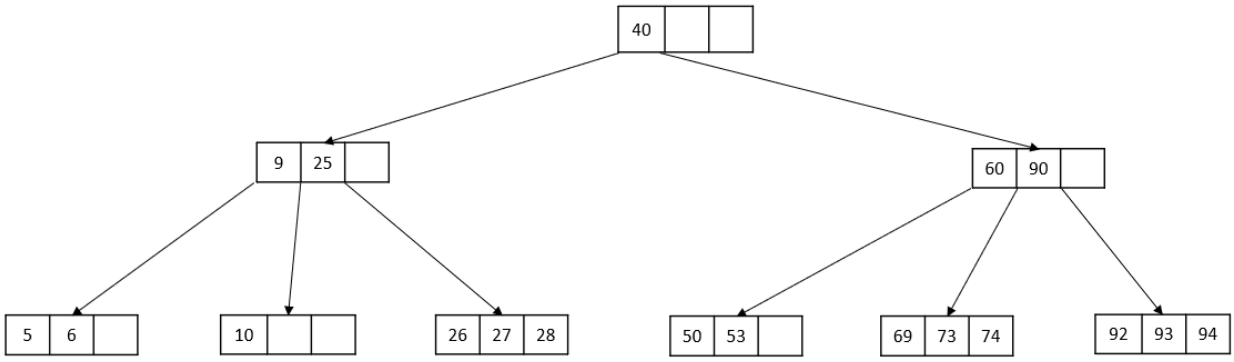

Array correspondiente al ejercicio 30

## Ejercicio 31

Considerando el árbol B de la figura, realiza las siguientes operaciones:

- Añadir la clave 71 sobre el árbol B original.
- Eliminar la clave 10 sobre el árbol B original.

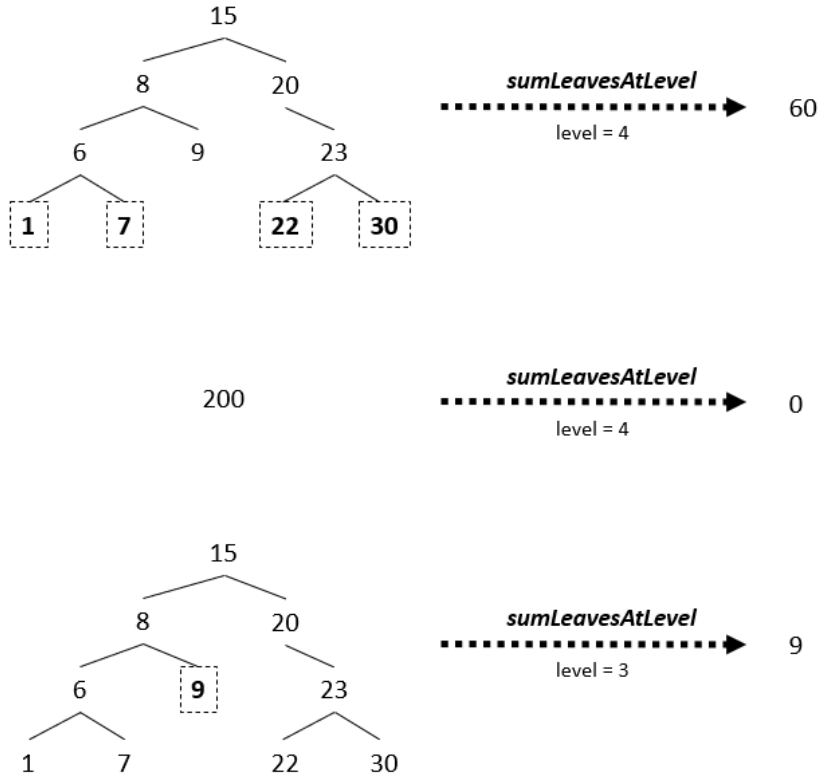
Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final.




Árbol B del ejercicio 31

## Ejercicio 32

Implementa un método, llamado *sumLeavesAtLevel*, que reciba un *BinaryTree<Integer>* (llamado *original*) y un entero llamado *level*. Este método devolverá un entero que será el resultado de sumar las hojas de *original* y que se encuentren en el nivel indicado por *level*. Si no hay ninguna hoja a ese nivel el resultado es 0.




Ejemplo relativo a la operación *sumLeavesAtLevel* del ejercicio 32

Con el fin de tener más clara la operación a realizar, la figura contiene un ejemplo de uso del método *sumLeavesAtLevel*. Además, os pedimos que mostréis con diagramas los diferentes casos existentes lo cual, además de ser la metodología explicada, os ayudará a encontrar la solución.

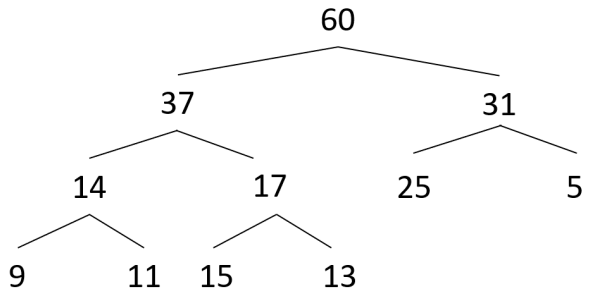
La implementación del método debe ser realizada **fuera** de la clase *LinkedBinaryTree<E>*.

Si es necesario hacer uso de algún método auxiliar, deberá ser implementado.

**NOTA:** Altres anys (i aquest és un problema d'un examen anterior) es definia l'alçada de l'arbre buit com a zero i, per tant, el nivell de l'arrrel era l'1. Podeu adaptar l'enunciat i la solució a la nova situació actual o resoldre'l tal i com està plantejat.

## Ejercicio 33

Dado el siguiente max-heap:




Max-heap del ejercicio 33

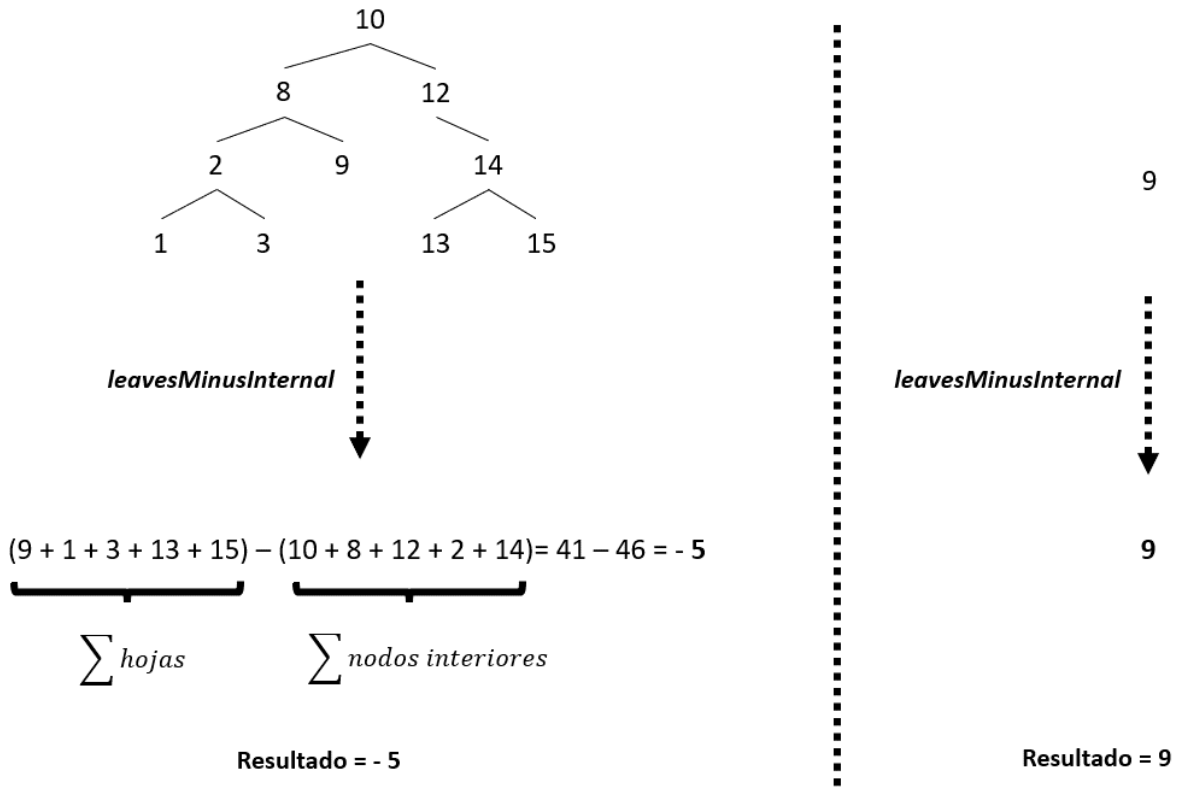
Realiza las siguientes operaciones:

- *Remove* sobre el max-heap original.
- *Insert del elemento 65* sobre el max-heap original.

Explica en cada parte de esta pregunta (*remove* e *insert*) las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final.

## Ejercicio 34

Implementa un método, llamado *leavesMinusInternal,* que reciba un *BinaryTree<Integer>* y devuelva el resultado de restar la suma de los nodos interiores a la suma de las hojas. Si el árbol es vacío, devolverá 0. A continuación, en la figura, se muestran dos ejemplos.



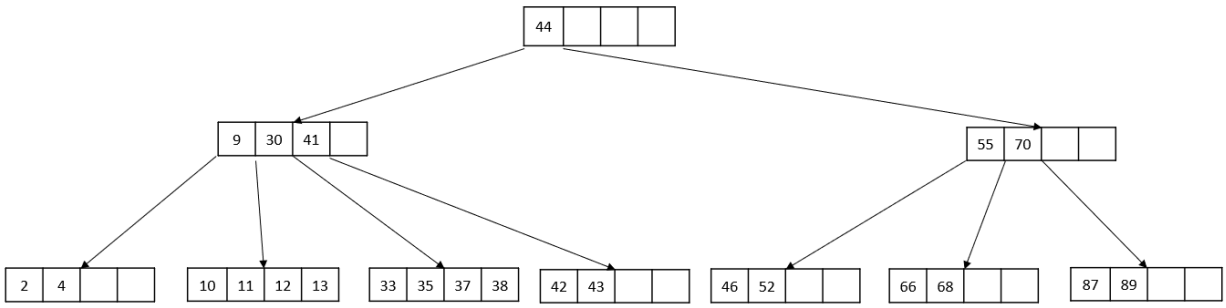

Ejemplos del ejercicio 34

## Ejercicio 35

Considerando el árbol B de la figura realiza las siguientes operaciones:

- Añadir la clave 15 sobre el árbol B original.
- Eliminar la clave 89 sobre el árbol B original.

Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final.




Árbol B original de ejercicio 35

## Ejercicio 36

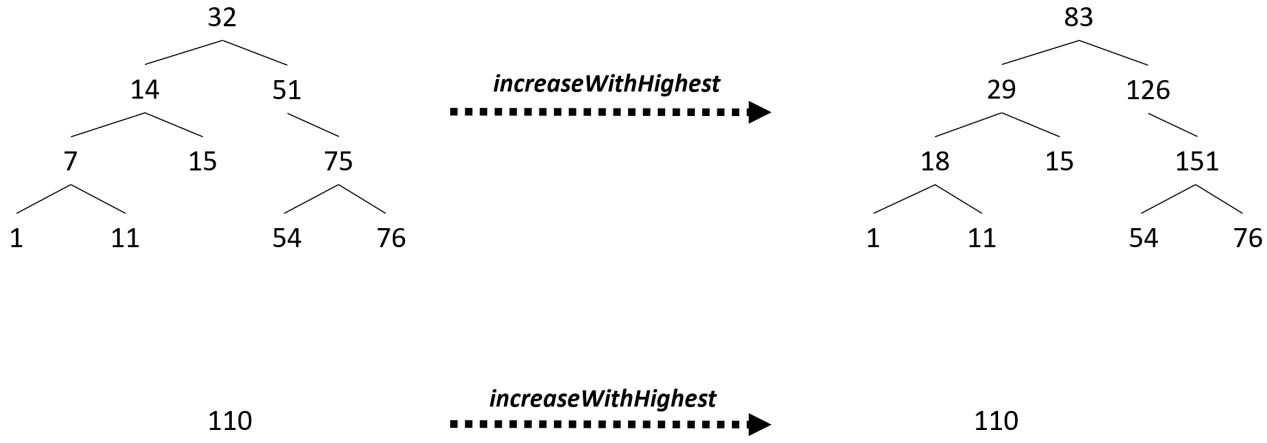
Implementa un método, llamado *increaseWithHighest*, que reciba un *BinaryTree<Integer>* (llamado *original*) y devuelva un *LinkedBinaryTree<Integer>* en el que cada posición del nuevo árbol contenga el resultado de sumar al nodo equivalente del árbol inicial, el valor del mayor de sus hijos también en el árbol inicial. En el caso de una hoja en el árbol original, su equivalente en el nuevo árbol será igual que en el original.

Si el árbol recibido como parámetro es vacío, la operación devolverá un árbol vacío. Para la realización del ejercicio, el árbol recibido nunca será *null*.

Con el fin de tener más clara la operación a realizar, la figura**¡Error! No se encuentra el origen de la referencia.** contiene un ejemplo de uso del método *increaseWithHighest*.

La implementación del método debe ser realizada **fuera** de la clase *LinkedBinaryTree<E>*.

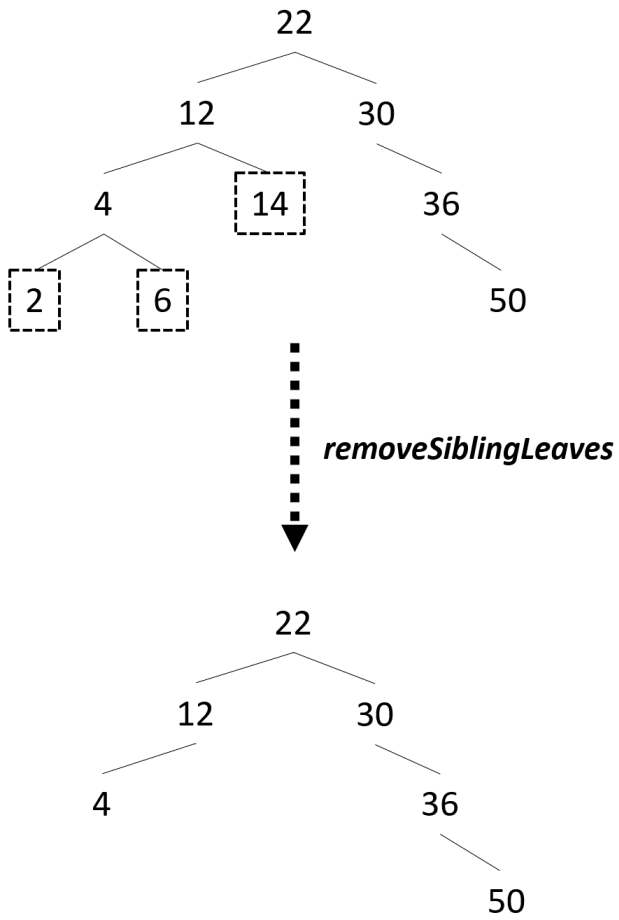
El árbol original no se modifica.




Ejemplo relativo a la operación *increaseWithHighest* del ejercicio 36

## Ejercicio 37

Implementa un método llamado *removeSiblingLeaves* que reciba un *BinaryTree<E>* (llamado *original*) y devuelva un *LinkedBinaryTree<E>* que será el resultado de **eliminar las hojas del árbol *original* cuyo padre sea un nodo interior de grado 2**. Si el árbol *original* es vacío, devolverá un *LinkedBinaryTree<E>* vacío. A continuación, en la Figura 3, se muestra un ejemplo para un *BinaryTree <Integer>*.



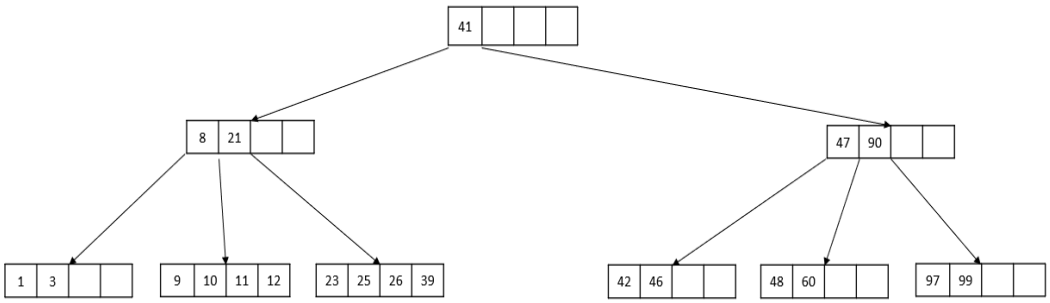

*Ejemplo de la removeSiblingLeaves para el caso de un BinaryTree<Integer>*

## Ejercicio 38

Considerando el árbol B dibujado más abajo realiza las siguientes operaciones:

- **Añadir** la clave 15 **sobre el árbol B original**.
- **Eliminar** la clave 47 **sobre el árbol B original**.

Explica las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final.




Árbol B del ejercicio 38