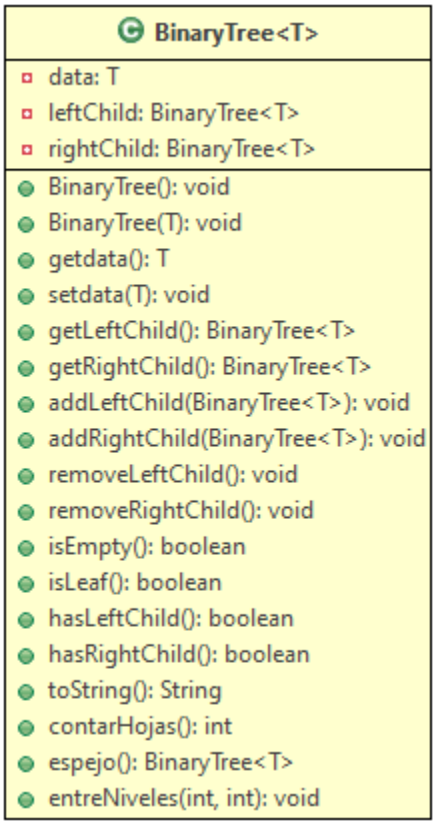
Juan Menduiña

Trabajo Práctico N° 2:
Árboles Binarios.

Ejercicio 1.

Considerar la siguiente especificación de la clase Java BinaryTree (con la representación hijo izquierdo e hijo derecho):



- *El constructor BinaryTree(T data) inicializa un árbol con el dato pasado como parámetro y ambos hijos nulos.*
- *Los métodos getLeftChild(): BinaryTree<T> y getRightChild(): BinaryTree<T> retornan los hijos izquierdo y derecho, respectivamente, del árbol. Si no tiene el hijo, tira error.*
- *El método addLeftChild(BinaryTree<T> child) y addRightChild(BinaryTree<T> child) agrega un hijo como hijo izquierdo o derecho del árbol.*
- *El método removeLeftChild() y removeRightChild() eliminan el hijo correspondiente.*
- *El método isEmpty() indica si el árbol está vacío y el método isLeaf() indica si no tiene hijos.*
- *El método hasLeftChild() y hasRightChild() devuelve un booleano indicando si tiene dicho hijo el árbol receptor del mensaje.*

Analizar la implementación en JAVA de la clase BinaryTree brindada por la cátedra.

Ver paquete “tp2.ejercicio1” en Java.

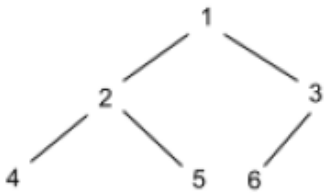
Ejercicio 2.

Agregar, a la clase BinaryTree, los siguientes métodos:

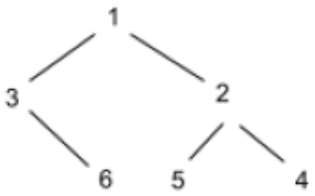
(a) *contarHojas(): int Devuelve la cantidad de árbol/subárbol hojas del árbol receptor.*

(b) *espejo(): BinaryTree<T> Devuelve el árbol binario espejo del árbol receptor.*

Por ejemplo, dado el árbol binario:



Retorna



(c) *entreNiveles(int n, m) Imprime el recorrido por niveles de los elementos del árbol receptor entre los niveles n y m (ambos inclusive). (0 ≤ n < m ≤ altura del árbol).*

Ver paquete “tp2.ejercicio2” en Java.

Ejercicio 3.

Definir una clase Java denominada ContadorArbol cuya función principal es proveer métodos de validación sobre árboles binarios de enteros. Para ello, la clase tiene como variable de instancia un BinaryTree<Integer>. Implementar, en dicha clase, un método denominado numerosPares() que devuelve, en una estructura adecuada (sin ningún criterio de orden), todos los elementos pares del árbol (divisibles por 2).

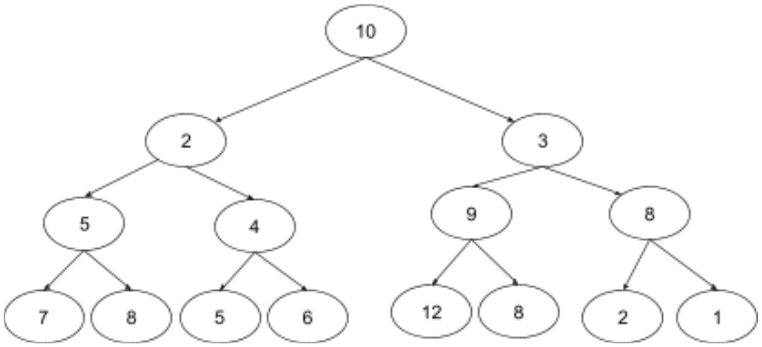
(a) *Implementar el método realizando un recorrido InOrden.*

(b) *Implementar el método realizando un recorrido PostOrden.*

Ver paquete “tp2.ejercicio3” en Java.

Ejercicio 4.

Una red binaria es una red que posee una topología de árbol binario lleno. Por ejemplo:



Los nodos que conforman una red binaria llena tienen la particularidad de que todos ellos conocen cuál es su retardo de reenvío. El retardo de reenvío se define como el período comprendido entre que un nodo recibe un mensaje y lo reenvía a sus dos hijos.

La tarea es calcular el mayor retardo posible, en el camino que realiza un mensaje desde la raíz hasta llegar a las hojas en una red binaria llena. En el ejemplo, se debería retornar 10 + 3 + 9 + 12= 34 (si hay más de un máximo, retornar el último valor hallado).

NOTA: Asumir que cada nodo tiene el dato de retardo de reenvío expresado en cantidad de segundos.

(a) *Indicar qué estrategia (recorrido en profundidad o por niveles) se utilizará para resolver el problema.*

Para resolver el problema, se utilizará un recorrido en profundidad.

(b) *Crear una clase Java llamada RedBinariaLlena donde se implementará lo solicitado en el método retardoReenvio(): int.*

Ver paquete “tp2.ejercicio4” en Java.

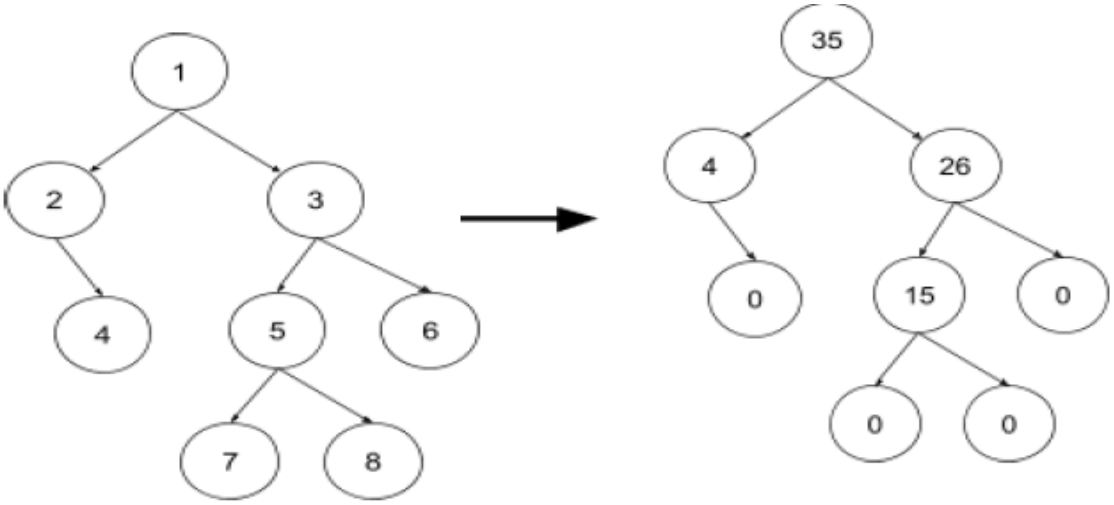
Ejercicio 5.

Implementar una clase Java llamada ProfundidadDeArbolBinario que tiene, como variable de instancia, un árbol binario de números enteros y un método de instancia sumaElementosProfundidad(int p): int, el cuál devuelve la suma de todos los nodos del árbol que se encuentren a la profundidad pasada como argumento.

Ver paquete “tp2.ejercicio5” en Java.

Ejercicio 6.

Crear una clase Java llamada Transformacion que tenga como variable de instancia un árbol binario de números enteros y un método de instancia suma(): BinaryTree<Integer>, el cual devuelve el árbol en el que se reemplazó el valor de cada nodo por la suma de todos los elementos presentes en su subárbol izquierdo y derecho. Asumir que los valores de los subárboles vacíos son ceros. Por ejemplo:



¿La solución recorre una única vez cada subárbol? En el caso que no, ¿se puede mejorar para que sí lo haga?

Ver paquete “tp2.ejercicio6” en Java.

Los siguientes ejercicios fueron tomados en parciales, en los últimos años. Tener en cuenta que:

1. No se pueden agregar más variables de instancia ni de clase a la clase ParcialArboles.
2. Se debe respetar la clase y la firma del método indicado.
3. Se pueden definir todos los métodos y variables locales que se consideren necesarios.
4. Todo método que no esté definido en la sinopsis de clases debe ser implementado.
5. Se debe recorrer la estructura sólo 1 vez para resolverlo.
6. Si corresponde, completar, en la firma del método, el tipo de datos indicado con signo de "?".

Ejercicio 7.

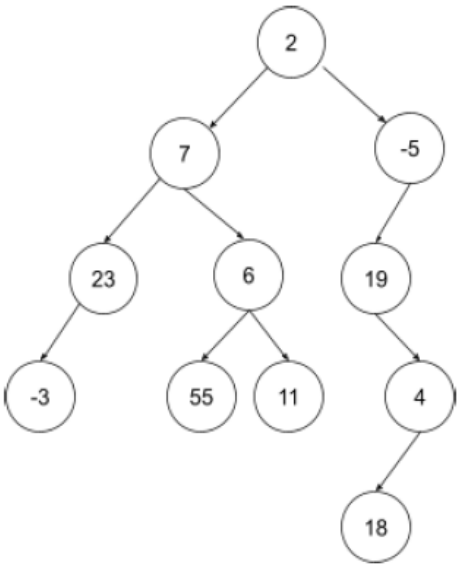
Escribir una clase ParcialArboles que contenga UNA ÚNICA variable de instancia de tipo BinaryTree de valores enteros NO repetidos y el método público con la siguiente firma:

public boolean isLeftTree(int num).

El método devuelve true si el subárbol cuya raíz es "num" tiene, en su subárbol izquierdo, una cantidad mayor estricta de árboles con un único hijo que en su subárbol derecho. Y false en caso contrario. Consideraciones:

- *Si "num" no se encuentra en el árbol, devuelve false.*
- *Si el árbol con raíz "num" no cuenta con una de sus ramas, considerar que, en esa rama, hay -1 árboles con único hijo.*

Por ejemplo, con un árbol como se muestra en la siguiente imagen:



Si num = 7 devuelve **true** ya que en su rama izquierda hay 1 árbol con un único hijo (el árbol con raíz 23) y en la rama derecha hay 0. (1 > 0) → true

Si num = 2 devuelve **false**, ya que en su rama izquierda hay 1 árbol con único hijo (árbol con raíz 23) y en la rama derecha hay 3 (árboles con raíces -5, 19 y 4). (1 > 3) → false)

Si num = -5 devuelve **true**, ya que en su rama izquierda hay 2 árboles con único hijo (árboles con raíces 19 y 4) y al no tener rama derecha, tiene -1 árboles con un único hijo. (2 > -1) → true

Si num = 19 debería devolver **false**, ya que al no tener rama izquierda tiene -1 árboles con un único hijo y en su rama derecha hay 1 árbol con único hijo. (-1 > 1) → false

Si num = -3 debería devolver **false**, ya que al no tener rama izquierda tiene -1 árboles con un único hijo y lo mismo sucede con su rama derecha. (-1 > -1) → false

Ver paquete "tp2.ejercicio7" en Java.

Ejercicio 8.

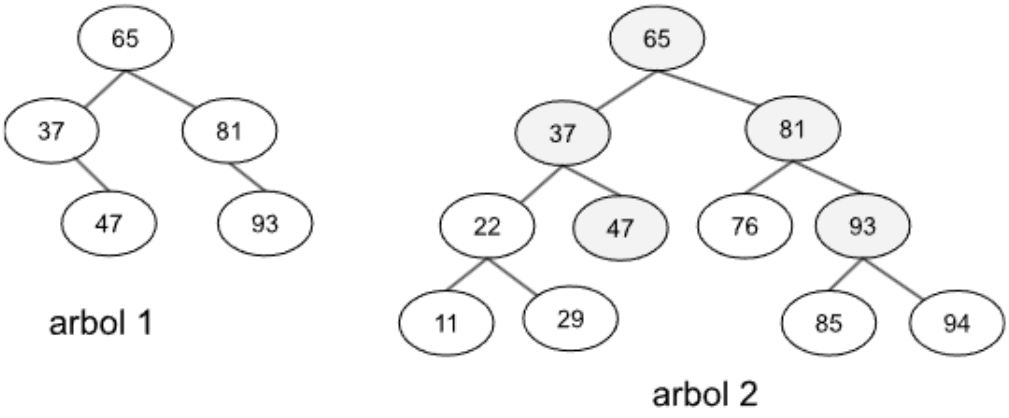
Escribir, en una clase ParcialArboles, el método público con la siguiente firma:

public boolean esPrefijo(BinaryTree<Integer> arbol1, BinaryTree<Integer> arbol2).

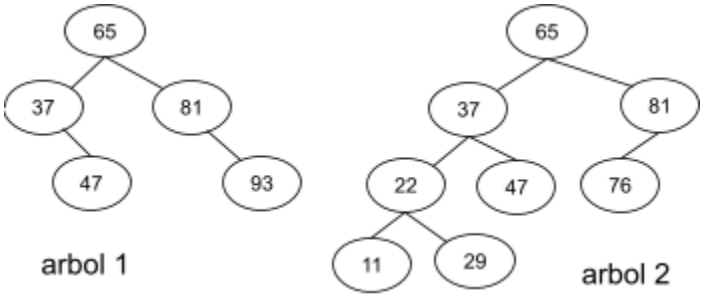
El método devuelve true si arbol1 es prefijo de arbol2, false en caso contrario.

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 estructura.

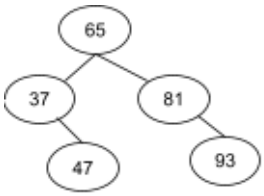
Por ejemplo, en la siguiente imagen, arbol1 ES prefijo de arbol2.



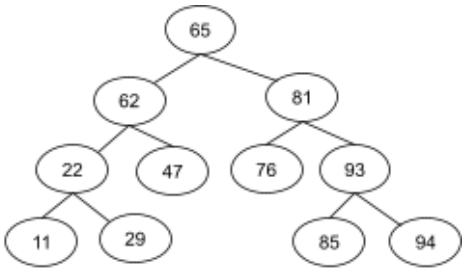
En esta otra, arbol1 NO es prefijo de arbol2 (el subárbol con raíz 93 no está en el árbol2).



En la siguiente, no coincide el contenido. El subárbol con raíz 37 figura con raíz 62, entonces, arbol1 NO es prefijo de arbol2.



arbol 1



arbol 2

Ver paquete “tp2.ejercicio8” en Java.

Ejercicio 9.

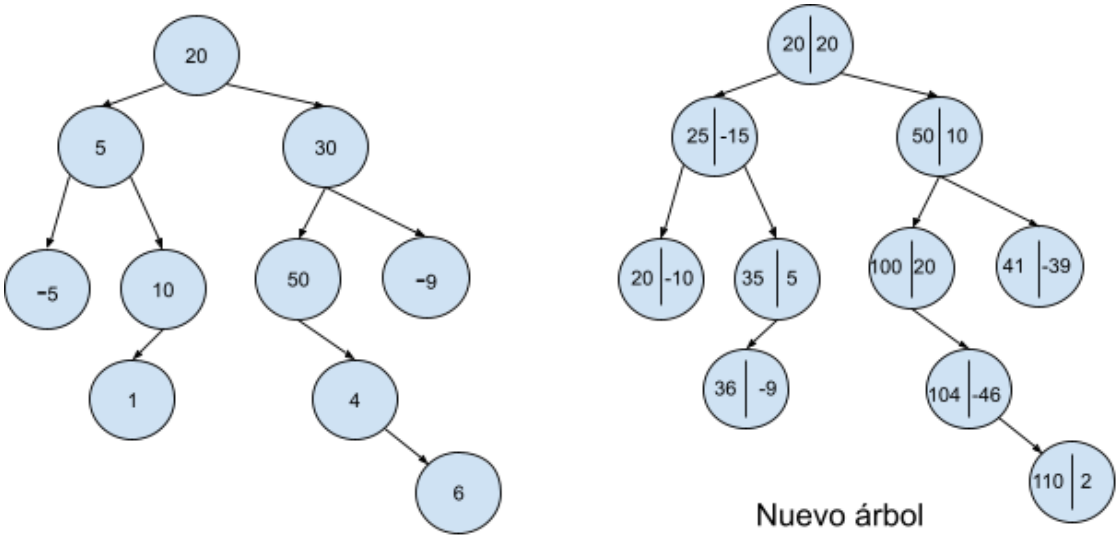
Escribir, en una clase ParcialArboles, el método público con la siguiente firma:

public BinaryTree<?> sumAndDif(BinaryTree<Integer> arbol).

El método recibe un árbol binario de enteros y devuelve un nuevo árbol que contiene, en cada nodo, dos tipos de información:

- *La suma de los números a lo largo del camino desde la raíz hasta el nodo actual.*
- *La diferencia entre el número almacenado en el nodo original y el número almacenado en el nodo padre.*

Ejemplo:



NOTA: En el nodo raíz, considerar que el valor del nodo padre es 0.

Ver paquete “tp2.ejercicio9” en Java.