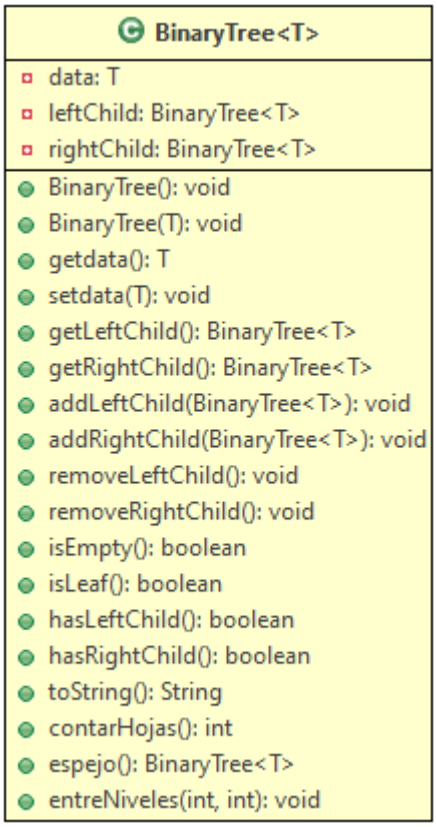
UNIVERSIDAD NACIONAL DE LA PLATA

Práctica 2
Árboles Binarios

Implemente cada ejercicio en un paquete que contenga los números del TP y del ejercicio. Ejemplo tp2.ejercicio3 (dentro del proyecto llamado "AYED").



Ejercicio 1

Considere la siguiente especificación de la clase Java **BinatyTree**(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.

a) Analice la implementación en JAVA de la clase **BinaryTree** brindada por la cátedra.

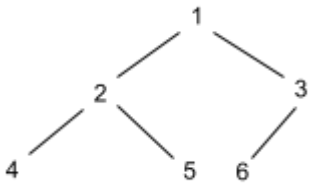
Ejercicio 2

Agregue a la clase **BinaryTree** los siguientes métodos:

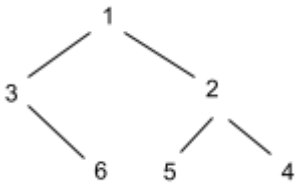
a) **contarHojas():int** Devuelve la cantidad de árbol/subárbol hojas del árbol receptor.

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)

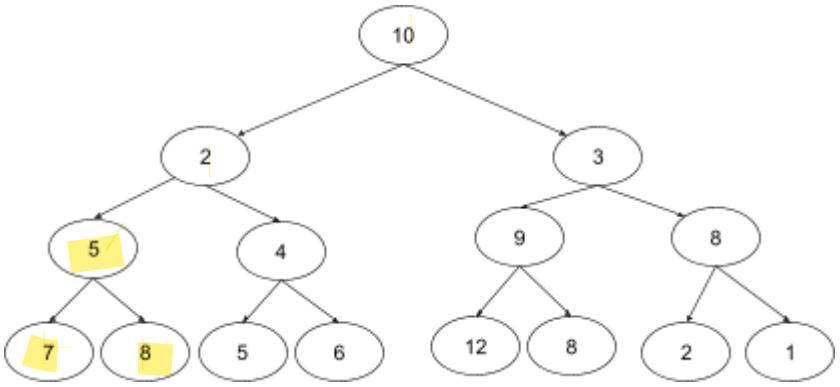
Ejercicio 3

Defina 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>. Implemente 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) Implemente el método realizando un recorrido InOrden.
b) Implemente el método realizando un recorrido PostOrden.

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 tiene 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.

Su 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, debería retornar 10+3+9+12=34 (Si hay más de un máximo retorne el último valor hallado).

Nota: asuma que cada nodo tiene el dato de retardo de reenvío expresado en cantidad de segundos.

a) Indique qué estrategia (recorrido en profundidad o por niveles) utilizará para resolver el problema. Profundidad

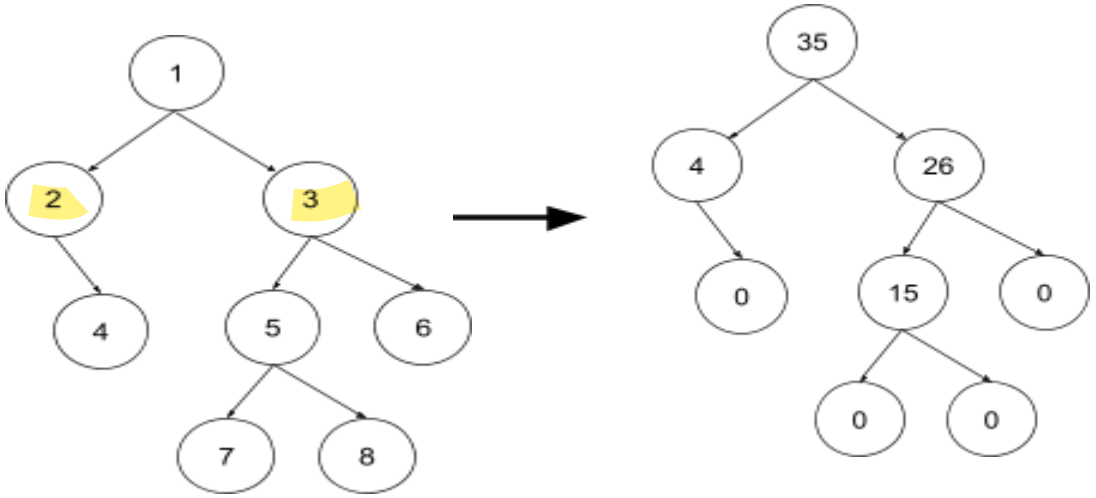
b) Cree una clase Java llamada **RedBinariaLlena** donde implementará lo solicitado en el método **retardoReenvio():int**

Ejercicio 5

Implemente 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.

Ejercicio 6

Cree 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 cuál 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. Asuma que los valores de los subárboles vacíos son ceros. Por ejemplo:



¿Su solución recorre una única vez cada subárbol? En el caso que no, ¿Puede mejorarla para que sí lo haga?

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

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

Ejercicio 7

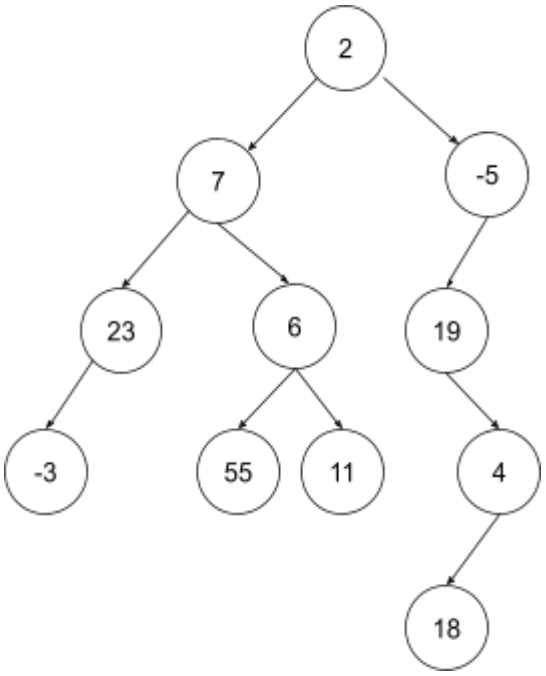
Escribir en una clase **ParcialArboles** que contenga **UNA ÚNICA variable de instancia de tipo BinaryTree** de valores enteros **NO** repetidos y el método público con la siguiente firma:

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 f**alse** 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, considere 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

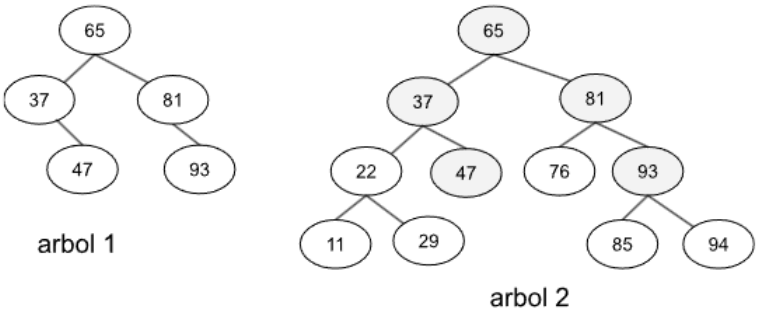
Ejercicio 8

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

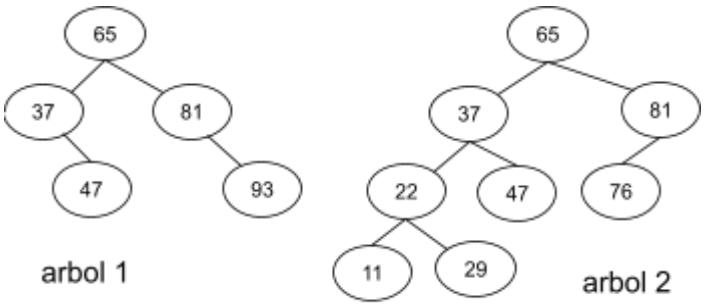
public boolean **esPrefijo**(BinaryTree<Integer> arbol1, BinaryTree<Integer> arbol2)

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

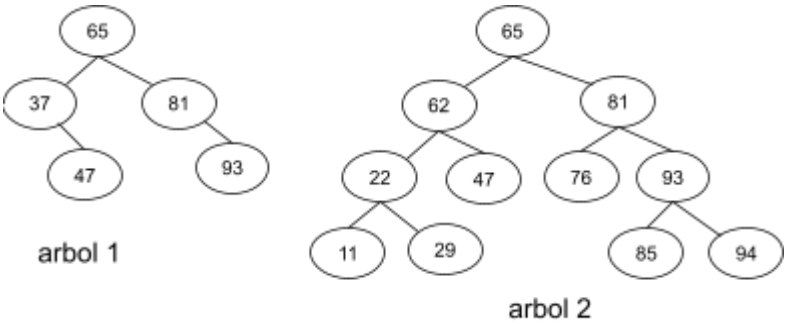
Se dice que un árbol binario arbol1 **es prefijo** de otro árbol binario arbol2, cuando arbol1 coincide con la parte inicial del árbol arbol2 **tanto en el contenido de los elementos como en su estructura**. Por ejemplo, en la siguiente imagen: arbol1 **ES** prefijo de arbol2.



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.



Ejercicio 9

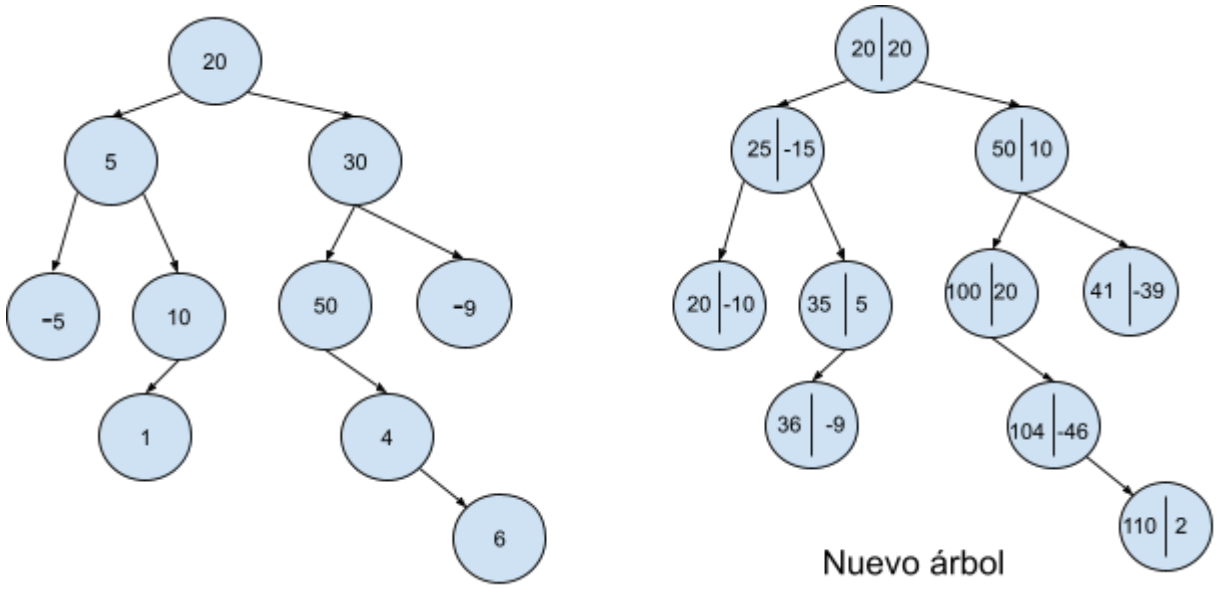
Escribir en una clase **ParcialArboles** el método público con la siguiente firma:

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

El método recibe un árbol binario de enteros y devuelve un **nuevo árbol** que contenga 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 considere que el valor del nodo padre es 0.