Kata 6

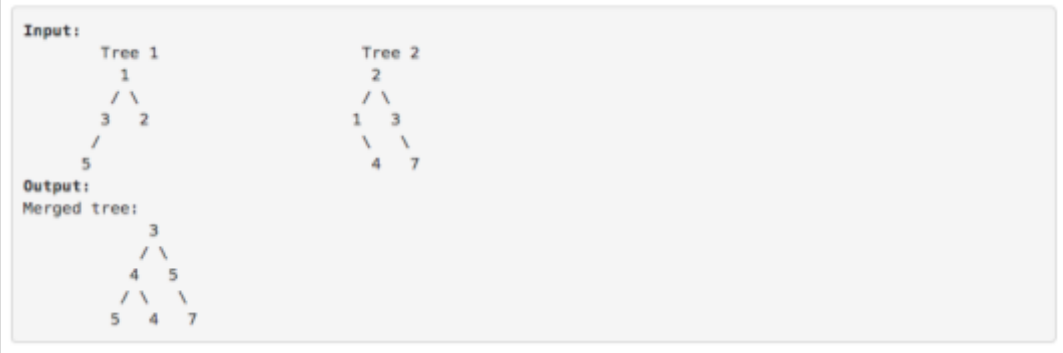
El siguiente reporte describe la solución a la Kata 6 de Programación Avanzada.

Problema

Merge Two Binary Trees

Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree.



Solución
Lo primero que pensé para realizar el problema, era directamente pasar a un array ambos árboles y sumar sus valores. Sin embargo, esta solución tiene fallas, ya que no resuelve el problema de cuando los árboles tienen huecos.

La segunda propuesta, la cual propongo como solución final, es recorrer ambos árboles en simultáneo (en *pre-orden*: raíz, izquierda y derecha). Cada vez que llegue a una raíz y vaya a pasar a la izquierda o derecha, revisa que en ambos árboles se pueda realizar esta operación:

a) si en **ambos árboles** existe nodo, suma su valor y lo deposita en el **árbol A**;
b) en caso de que sólo exista en el **árbol A**, se deja el nodo igual;
c) en caso de que sólo exista en el **árbol B**, se inserta un nuevo nodo en el **árbol A** con el valor del nodo en el árbol B.

El recorrido sigue una vez que se termina esto, procurando que ambos árboles naveguen por las mismas ramas de manera adecuada. Con esta propuesta, se evita el problema que teníamos en la primera solución. La **solución sería O(n)**, que es lo que toma recorrer todos los *n* nodos del árbol.

A continuación, se muestra el pseudocódigo de la solución:

Algorithm 1

```
mergeBT(rootA, rootB)
  rootA.value += rootB.value

  if rootA.left != null:
    if rootB.left != null:
      mergeBT(rootA.left, rootB.left)
  else:
    if rootB.left != null:
      rootA.left = rootB.left

  if rootA.right != null:
    if rootB.right != null:
      mergeBT(rootA.right, rootB.right)
  else:
    if rootB.right != null:
      rootA.right = rootB.right

  return rootA;
```

Una tercera solución sería casi igual a la segunda, con la diferencia de que tendríamos un **árbol C**. Conforme recorremos los árboles y ejecutamos las sumas, se guarda el resultado en un **array C**. Este array tendría la estructura de nodos del **árbol C** en *pre-orden*, a lo que sólo seguiría crear el árbol. Esta propuesta es más enfocada a su posible implementación en C, para evitar problemas con el manejo de memoria y punteros.