Estructuras de Datos

Tarea TDA Árbol Binario

Los siguientes métodos deben ser incluidos dentro de la clase `BinaryTree`. Cada método solicitado debe ser implementado tanto recursiva como iterativamente (a menos que se especifique lo contrario). Los sufijos *Recursive* e *Iterative* deben ser incluidos en el nombre de cada método para indicar su naturaleza (de manera similar a lo hecho con los métodos `countLeaves`**`Recursive`** y `countLeaves`**`Iterative`** de la introducción):

1. Implemente el método **`countDescendants`**, que cuenta el número de descendientes que tiene un árbol. Su método no debe contar a la raíz del árbol que lo invoca. Por ejemplo, una hoja tiene cero descendientes.

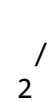
2. Implemente el método **`findParent`**, que dado un nodo de árbol binario, retorna el padre correspondiente. La implementación de su método debe considerar que el nodo raíz no tiene un padre.

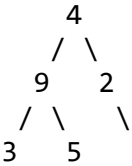
3. Implemente el método **`countLevels`** que calcule el número de niveles de árbol. Considere que un árbol vacío tiene 0 niveles, mientras que un árbol hoja tiene 1 solo nivel.

4. Se dice que un árbol binario es zurdo si el árbol: 1) está vacío, 2) es una hoja, o 3) si sus hijos son ambos zurdos y tiene a más de la mitad de sus descendientes en el hijo izquierdo. Implementar el método **`isLefty`** que indique si un árbol binario es zurdo o no.

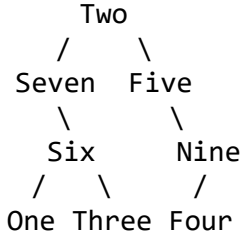
5. Implemente el método **`isIdentical`** que, dado un segundo árbol binario, retorne `true` o `false` indicando si dicho árbol es igual al que invoca el método.

6. Encontrar el valor más grande de cada nivel del árbol. El método **`largestValueOfEachLevel`** debe imprimir el mayor valor presente en cada nivel de un árbol binario cuyos nodos contienen números enteros. Ejemplos:

Árbol que invoca el método:

Salida en pantalla: 1 3

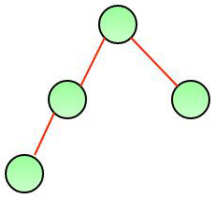
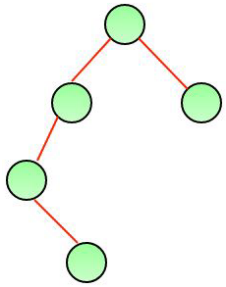
Árbol que invoca el método:

Salida en pantalla: 4 9 7

7. El método **`countNodesWithOnlyChild`** debe retornar el número de nodos de un árbol que tienen un solo hijo. Ejemplo:

Árbol que invoca el método:

Valor retornado: 3 (los nodos `Seven`, `Five` y `Nine` tienen un solo hijo).

8. El método **`isHeightBalanced`** debe retornar si un árbol binario está balanceado en altura o no. Un árbol vacío está siempre balanceado en altura. Un árbol binario no vacío está balanceado si y solo si:
   1) Su subárbol izquierdo está balanceado,
   2) Su subárbol derecho está balanceado, y
   3) La diferencia entre las alturas de sus subárboles izquierdo y derecho es menor que 1.

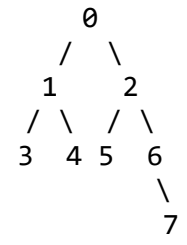
   El siguiente diagrama muestra dos árboles, uno de ellos está balanceado en altura y el otro no. El segundo árbol no está balanceado en altura porque la altura de su subárbol izquierdo es mayor en 2 unidades que la altura de su subárbol derecho:

Árbol balanceado	Árbol **NO** balanceado
	

Cada uno de los métodos implementados deben ser probados en un programa principal. El código que se muestra a continuación puede ser utilizado en su programa principal para instanciar dos árboles binarios con los que usted puede probar sus soluciones.

Árbol binario de enteros:

Código	**Árbol**

`BinaryTree<Integer> tree = new BinaryTree(0);`
`tree.setLeft(new BinaryTree(1));`
`tree.setRight(new BinaryTree(2));`
`tree.getLeft().setLeft(new BinaryTree(3));`
`tree.getLeft().setRight(new BinaryTree(4));`
`tree.getRight().setLeft(new BinaryTree(5));`
`tree.getRight().setRight(new BinaryTree(6));`
`tree.getRight().getRight().setRight(new BinaryTree(7));`
	

Árbol binario de cadenas de caracteres:

Código	**Árbol**

`BinaryTree<Integer> tree = new BinaryTree("Zero");`
`tree.setLeft(new BinaryTree("One"));`
`tree.setRight(new BinaryTree("Two"));`
`tree.getLeft().setLeft(new BinaryTree("Three"));`
`tree.getLeft().setRight(new BinaryTree("Four"));`
`tree.getRight().setRight(new BinaryTree("Five"));`