Estructuras de Datos

Taller y Tarea de Árboles Binarios

Introducción

En esta sesión, usted implementará varios métodos de la clase `BinaryTree`, que abstrae el TDA árbol binario.

El código adjunto incluye las clases parametrizadas `BinaryNode` y `BinaryTree` que abstraen, respectivamente, los nodos de un árbol binario y la estructura en sí misma.

Un nodo tiene un contenido *genérico* y referencias a dos **subárboles** (izquierdo y derecho):

```
private T content;
private BinaryTree<T> left;
private BinaryTree<T> right;
```

Por otro lado, la clase `BinaryTree` tiene, como único atributo, una referencia al nodo raíz del árbol:

```
private BinaryNode<T> root;
```

Para los objetivos que esta práctica persigue, la clase `BinaryTree` incluye los métodos `countLeavesRecursive` y `recursiveSearch` para, respectivamente, contar el número de hojas del árbol y para buscar el nodo que almacena un contenido en particular. Ambos métodos aplican una estrategia recursiva. `countLeavesRecursive`, por ejemplo, es llamado por los hijos izquierdo y derecho del árbol (siempre que estos no sean `null`):

```
public int countLeavesRecursive () {
    if (this.isEmpty()) {
        return 0; // un árbol vacío tiene cero hojas
    } else if (this.isLeaf()) {
        return 1; // una hoja debería retornar 1
    } else {
        int leavesLeft = 0;
        int leavesRight = 0;
        if (this.root.getLeft() != null) {
            leavesLeft = this.root.getLeft().countLeavesRecursive();
        }
        if (this.root.getRight() != null) {
            leavesRight = this.root.getRight().countLeavesRecursive();
        }
       // retornamos la suma de las hojas que hay en los subárboles
        return leavesLeft + leavesRight;
    }
}
```

El método para contar las hojas de un árbol también puede implementarse de manera iterativa. Dicha implementación requiere de pilas para almacenar los hijos de cada sub-árbol del árbol original.
Inicialmente, la pila almacena únicamente al árbol original.
Mientras la pila no se vacíe, el algoritmo remueve el tope y pregunta si dicho sub-árbol es una hoja o no. En los casos afirmativos, incrementa una variable que lleva la cuenta de cuántas hojas han sido encontradas hasta ahora. Posteriormente, añade a la pila los sub-árboles hijos del tope (el izquierdo y derecho) y repite los pasos anteriores (remover tope, preguntar si es hoja, y apilar a sus hijos).

El código mostrado a continuación es la versión iterativa de método para contar el número total de hojas de un `BinaryTree`:

```
public int countLeavesIterative() {
       Stack<BinaryTree<T>> stack = new Stack();
        int totalLeaves = 0; // al inicio, temenos 0 hojas
        if (this.isEmpty()) {
              return 0; // un árbol vacío tiene 0 hojas
        } else {
             stack.push(this); // el árbol original es almacenado en la pila
             while (!stack.empty()) {
                 BinaryTree<T> subtree = stack.pop();
                 if (subtree.isLeaf()) {
                     totalLeaves++;
                 }
                 // añadimos los descendientes (solo si no son null)
                 if (subtree.getRoot().getLeft() != null) {
                     stack.push(subtree.getRoot().getLeft());
                 }
                 if (subtree.getRoot().getRight() != null) {
                     stack.push(subtree.getRoot().getRight());
                 }
             }
        }
        // al finalizar el while, retornamos el contador de hojas
        return totalLeaves;
    }
```

Ejercicio Introductorio

Dibuje, en una pieza de papel, el árbol binario referenciado por la variable `tree` que resulta al ejecutar las siguientes líneas de código:

```
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));
```

En el proyecto de Netbeans adjunto, reproduzca las líneas de código mostradas arriba en el programa principal de la clase llamada `Main`. Durante el resto de la práctica, utilice el árbol `tree` para probar la implementación de los algoritmos que se solicitan a continuación.

Note que esta sesión práctica está compuesta por dos componentes: un **TALLER** que debe ser entregado hacia el final de la clase y una **TAREA** que debe ser entregada posteriormente.

Ejercicios

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).

Taller

1. Implemente una versión iterativa (la recursiva ya está implementada en el código recibido) del método `recursiveSearch`, que busca el nodo de un árbol binario que almacena un contenido específico. Llame a este método **`iterativeSearch`**.

2. Implemente el método **`getMin`** que, al ser invocado por un árbol de genéricos, encuentra el nodo que almacena al "menor" de los valores contenidos en el árbol.

3. 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.

Tarea

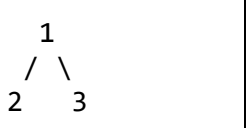
1. Implemente el método **`findParent`**, que dado un nodo de árbol binario, retorna el padre correspondiente. La implementación de su método debe considerar que el nodo raíz no tiene un padre.

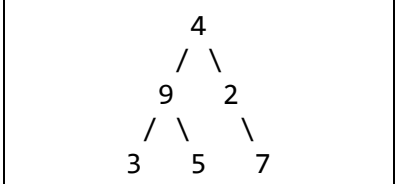
2. Implemente el método **`countLevels`** que calcule el número de niveles de árbol. Considere que un árbol vacío tiene 0 niveles, mientras que un árbol hoja tiene 1 solo nivel.

3. 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.

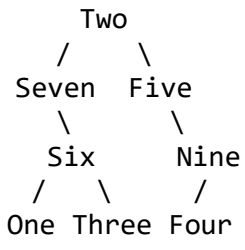
4. Implemente el método **`isIdentical`** que, dado un segundo árbol binario, retorne `true` o `false` indicando si dicho árbol es igual al que invoca el método (`this`).

5. 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

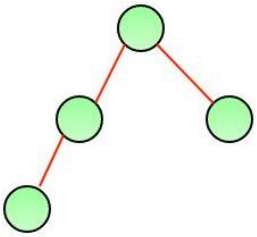
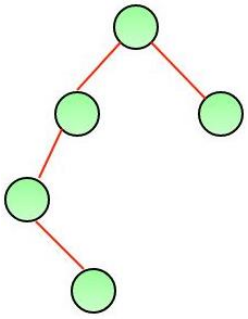
Árbol que invoca el método:

Salida en pantalla: 4 9 7

6. 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).

7. 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:

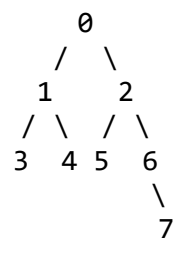
Árbol balanceado	Árbol **NO** balanceado
	

Otras formas de probar sus métodos

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"));`