Estructuras de Datos Avanzadas

Prof: Dr. Fernando Esponda Darlington

César Darío Sotelo Aportela

Experimento ABB

Tabla de contenido

Introducción

El objetivo de este experimento es diseñar un algoritmo para mantener un árbol binario de búsqueda con altura O(lg(n)) sin utilizar rotaciones. Se propone un par de métodos que funcionan en conjunto para mantener un arbol con estas especificaciones. La idea general del sistema es deshacer el árbol para volverlo a construir con altura log(n).

Descripción del algoritmo

La solución implementada consiste en dos métodos que se usan en conjunto para ordenar el árbol. Para usar la solución se necesitan solo dos líneas de código:

```
list=tree.getValues(tree.root);

tree.root=tree.createTreeFromList(list);
```

La primera instrucción pide que los valores de un árbol no balanceado, se guarden en una lista. La segunda instrucción pide que la raíz del árbol se reemplace por el nuevo árbol que crea el método “createTreeFromList”.

Los métodos se describen adelante:

createTreeFromList:

```
public <T extends Comparable<T>> BTNode<T> createTreeFromList(List<T> list) {
    if (list==null || list.isEmpty())
       return null;
    int middleNumber = list.size()/2;
    BTNode<T> root = new BTNode<>(list.get(middleNumber));
    root.left = createTreeFromList(list.subList(0, middleNumber));
    root.right = createTreeFromList(list.subList(middleNumber+1, list.size()));
```

```
        return root;
}
```

Este método es una forma recursiva para ordenar los nodos en una lista y construir un árbol. La lista debe estar ordenada. Primero empieza con el caso base, en el que compara si la lista es nula o si está vacía, regresa un valor nulo. Después se guarda la mitad de la lista en un entero y se guarda una raíz con este indice. Este método se manda a llamar a sí mismo recursivamente y se asigna a sí mismo las raíces en sus nodos de izquierda y derecha con los rangos partiendo de 0 a middle number y de middleNumber al final respectivamente.

getValues:

```
public <T extends Comparable<T>> List<T> getValues(BTNode<T> root) {
        List<T> list = new ArrayList<>();
        if (root==null)
            return list;
        Stack<BTNode<T>> stack = new Stack<>();
        stack.push(root);
        while (!stack.isEmpty()){
            BTNode<T> node = stack.pop();
            list.add(node.getElem());
            if (node.right != null)
                stack.push(node.right);
            if (node.left != null)
                stack.push(node.left);
        }
        Collections.sort(list);
        return list;
    }
```

Este método obtiene los valores de un árbol y al final usa las propiedades de Collections.sort(list) para ordenar la lista. Parecido al pasado, usa metodología recursiva para pasar por cada uno de los nodos y estos nodos son guardados en una pila.

Métricas de eficiencia

Este algoritmo no es muy eficiente a comparación de los vistos en clase con rotaciones. Este método guarda todos los elementos en una lista, en este paso hace n operaciones. Después hace un árbol con la lista, usando todos los elementos, otras n operaciones. Por lo tanto el algoritmo es de orden O(n). Haría falta buscar una nueva solución o si se busca eficiencia, se debería tratar de implementar los métodos que se usan en los árboles AVL.

Resultados

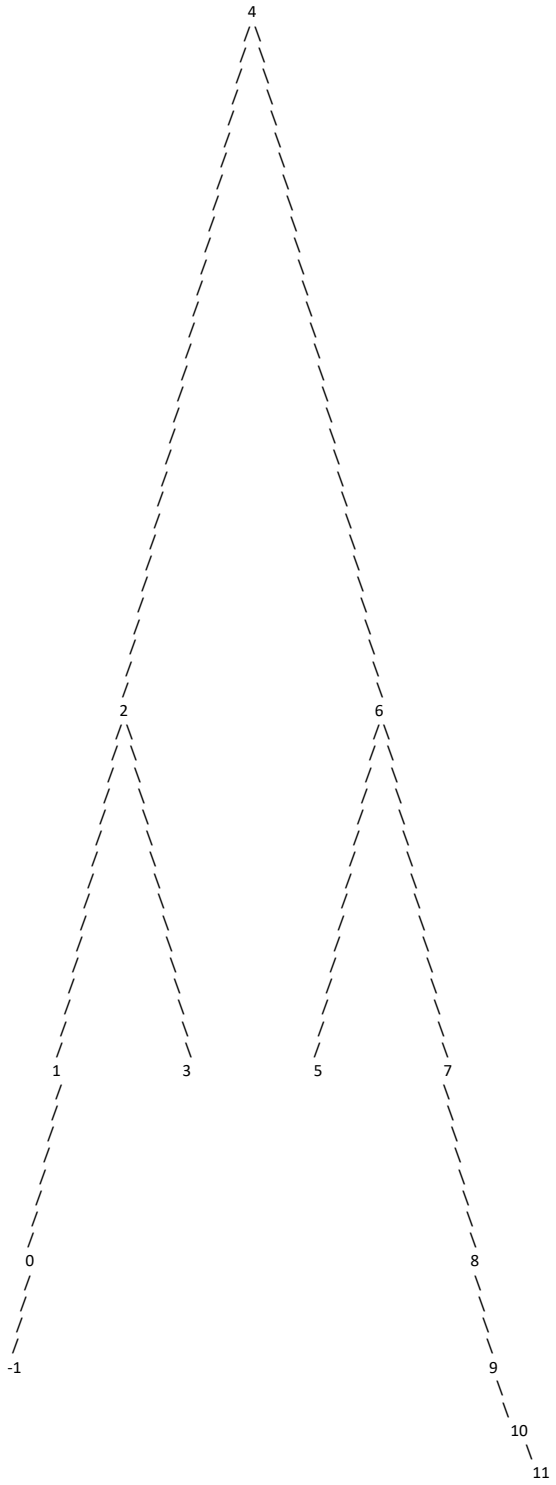
Para los resultados se declaró un árbol muy desbalanceado, empezamos declarando un árbol balanceado de tres niveles.

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

En esta sección se agregan nodos para construir un árbol desbalanceado

```
root.left.left.setLeft(new BTNode<>(0));
root.left.left.left.setLeft(new BTNode<>(-1));
root.right.right.setRight(new BTNode<>(8));
root.right.right.right.setRight(new BTNode<>(9));
root.right.right.right.right.setRight(new BTNode<>(10));
root.right.right.right.right.right.setRight(new BTNode<>(11));
```

Esto nos va a devolver un árbol que se ve de la siguiente forma:

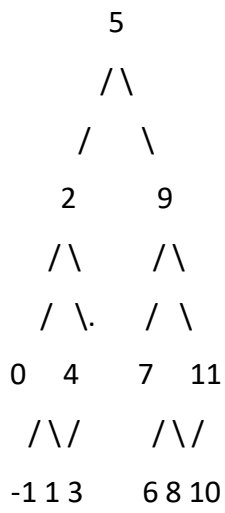


Usando los comandos que declaré arriba,

```
list=tree.getValues(tree.root);

tree.root=tree.createTreeFromList(list);
```

nos va a quedar el siguiente árbol:



Se nota considerablemente la diferencia. Podemos ver que los dos métodos cumplen con el objetivo del problema, si se va a usar para algún proyecto, se sugiere usarlo al acabar de agregar nodos.