Nombre y Apellido: ..... N° Legajo: .....

Primer Parcial de Estructuras de Datos y Algoritmos

Primer Cuatrimestre de 2010 – 19/04/2010

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*

Condición Mínima de Aprobación: Tener por lo menos dos ejercicios con B-

Consideraciones a tener en cuenta. MUY IMPORTANTE

- **El ejercicio que no respete estrictamente el enunciado será anulado.**
- **Se puede entregar el examen escrito en lápiz**
- **Se tendrán en cuenta la eficiencia y el estilo de programación.**
- **Los teléfonos celulares deben estar apagados.**

Ejercicio 1

```
public class BST<T> implements BinarySearchTree<T> {

   private Node root;
   private Comparator<? super T> cmp;

   public BST(Comparator<? super T> cmp) {
      this.root = null;
      this.cmp = cmp;
   }

   public void add(T value) {
      root = add(root, value);
   }

   public boolean contains(T value) {
      return contains(root, value);
   }

   public void remove(T value) {
      root = remove(root, value);
   }

   private Node add(Node node, T value) {
          ...
   }

   private boolean contains(Node node, T value) {
          ...
   }

   private Node remove(Node node, T value) {
          ...
   }


   private class Node {
      T value;
      Node left;
      Node right;

      Node(T value) {
         this.value = value;
      }
   }
}
```

Agregar a la clase de BST un método que reciba un número real P y determine si el árbol es P-balanceado por peso. Se dice que un árbol es P-balanceado por peso si para todo nodo se cumple que:

- weight(left) <= P * weight(node)
- weight(right) <= P * weight(node)

Donde *weight(node)* es la cantidad de nodos que tiene el subárbol que tiene a *node* como raíz.

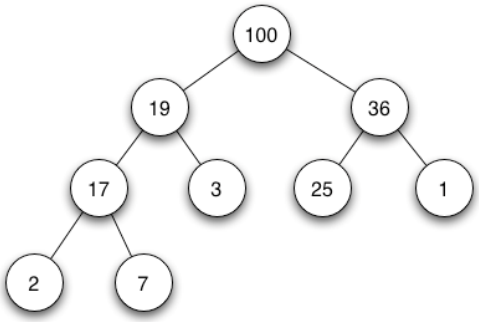
- No realizar ningún cambio en la estructura, sólo agregar los métodos necesarios.
- Un árbol vacío es P-balanceado.
- El método debe visitar cada nodo no más de una vez.
- Si el valor de P es tal que sea imposible que el árbol sea P-balanceado, debe retornar que no es P-balanceado sin recorrer el mismo.

Ejercicio 2

Se tiene un árbol binario donde se cumple, para todo nodo, que si B es nodo hijo de A, entonces el valor almacenado en B es menor o igual al valor almacenado en A, por lo que la raíz contiene siempre el máximo valor del árbol.

Algunas operaciones que se pueden realizar son:

- Delete: elimina la raíz del árbol
- Insert: inserta un nodo en algún lugar que le corresponde



Se pide

a) Adaptar, de ser necesario, la estructura del BST del ejercicio 1 para soportar este tipo de árboles.
b) Agregar a la clase resultante el método *delete*, que retorna la raíz del árbol y elimina del mismo el nodo correspondiente.
c) ¿Cuál es el orden de complejidad temporal del método implementado en el inciso anterior?

Ejercicio 3

Escribir una clase con un método estático que reciba un char[] que representa una expresión y determine si la misma contiene los paréntesis, corchetes y llaves balanceados.
Por ejemplo:

“(8) [ ] { }” está balanceada
“(([ ]) { ( ) })” está balanceada
“( ]” no está balanceada
“) (“ no está balanceada

Definir todas las clases auxiliares que considere necesarias. No utilizar la API de Java.