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

Primer Parcial de Estructuras de Datos y Algoritmos

Primer Cuatrimestre de 2011 – 23/04/2012

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

Se cuenta con la siguiente interfaz que representa una bolsa (una colección de elementos que admite repetidos):

```
public interface Bag<T> {

   /** Agrega un elemento a la colección. */
   public void add(T elem);

   /** Elimina un elemento de la colección. */
   public void remove(T elem);

   /** Imprime los elementos sin repetir, indicando la cantidad de
    * veces que aparece cada uno, y ordenado descendentemente por dicha cantidad. */
   public void print();
}
```

A continuación se muestra un ejemplo de uso:

```
   public static void main(String[] args) {

          Bag<String> bag = new BagImpl<String>();

          bag.print();    // No imprime nada.

          bag.add("A"); bag.add("B"); bag.add("C"); bag.add("B");

          bag.print();    // Imprime  B (2)  C (1)  A (1)

          bag.add("A"); bag.add("C"); bag.add("C");

          bag.print();    // Imprime  C (3)  A (2)  B (2)

          bag.remove("C"); bag.add("D");

          bag.print();    // Imprime  C (2)  A (2)  B (2)  D (1)

          bag.remove("C");

          bag.print();    // Imprime  A (2)  B (2)  C (1)  D (1)

          bag.remove("C");

          bag.print();    // Imprime  A (2)  B (2)  D (1)

   }
```

Realizar una implementación de Bag eficiente en espacio, que almacene los elementos en una lista lineal.

- **Los métodos add, remove y print deben resolverse con complejidad temporal O(N), siendo N la cantidad de elementos diferentes que están almacenados en la colección.**
- **El espacio utilizado por la estructura para almacenar los datos debe ser O(N), siendo N la cantidad de elementos diferentes que están almacenados en la colección.**
- **No utilizar clases de la API de Java.**

Ejercicio 2

Se tiene la siguiente clase que representa un árbol binario:

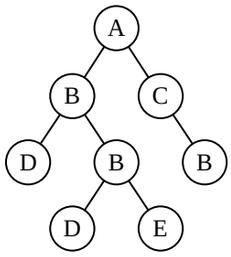
```
public class BinaryTree<T> {
    private T value;
    private BinaryTree<T> left, right;

    public BinaryTree(T value, BinaryTree<T> left, BinaryTree<T> right) {
           this.value = value;
           this.left = left;
           this.right = right;
    }
    public T getValue() {
           return value;
    }
    public BinaryTree<T> getLeft() {
           return left;
    }
    public BinaryTree<T> getRight() {
           return right;
    }
}
```

Implementar un método que dado un árbol binario y un valor a buscar, busque todas las ocurrencias del mismo, y retorne un árbol recubridor que contenga únicamente las ramas desde la raíz hasta cada nodo que contenga el valor buscado.

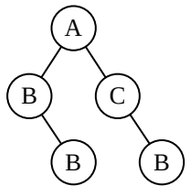
`public static <T> BinaryTree<T> spanning(BinaryTree<T> tree, T value)`

- **No se pueden crear métodos auxiliares ni modificar la clase BinaryTree.**
- **Los nodos no deben ser visitados más de una vez.**

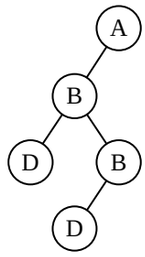
Ejemplo: dado el árbol de la figura (1), si se ejecuta el algoritmo buscando el valor "B", se obtiene el árbol de la figura (2). Si se busca el valor "D", se obtiene el árbol de la figura (3). Si se busca el valor "E" se obtiene el árbol vacío (null).



(1)



(2)



(3)

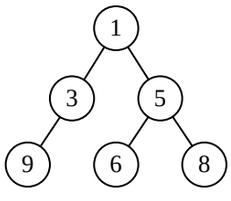
Ejercicio 3

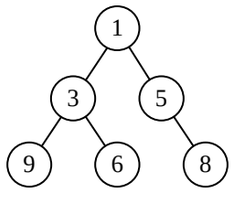
Un *heap* es un árbol binario que cumple las siguientes condiciones:

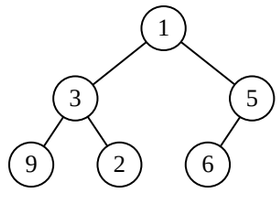
- El valor de cada nodo es menor al de su hijo izquierdo y al de su hijo derecho.
- Es perfectamente balanceado
- Todas sus hojas están lo más a la izquierda posible

Implementar un método que dado un árbol binario (clase BinaryTree del ejercicio anterior) y un comparador, retorne **true** si el árbol cumple con la definición de *heap*, y **false** en caso contrario.

Ejemplos: los siguientes árboles **no** cumplen con la definición de *heap*.







Los siguientes árboles **sí** cumplen con la definición de *heap*.