2° PARCIAL DE ALGORITMICA Y PROGRAMACION II - 2022

1) Agregar a la clase **ArrayList.java** el siguiente método:

```
/**
 * Verifica si la lista l es una subLista. Es decir que la lista l está
 * contenida dentro de la lista
 *
 * @param l lista a verificar
 * @return true si l es una subLista o false si no lo es.
 */
/**
 * Ejemplo:
 *
 * Dada la lista: [1, 2, 4, 5, 3, 6, 7]
 *
 * Las siguientes listas son subListas: [3, 6, 7] [2, 4] [] [1]
 *
 * Las siguientes listas no son subListas: [9] [3, 7, 6] [2, 1]
 *
 */
public boolean isSubList(List<E> l)
```

Correr el siguiente Test de JUnit y probar su funcionamiento.

```
package test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.BeforeClass;
import org.junit.Test;

import net.datastructures.ArrayList;
import net.datastructures.List;

public class TestSubList {

	private static List<Integer> lista1;
	private static List<Integer> lista2;

	/**
	 * Ejemplo:
	 *
	 * Dada la lista: [1, 2, 4, 5, 3, 6, 7]
	 *
	 * Las siguientes listas son sublista: [3, 6, 7] [2, 4] [] [1]
	 *
	 * Las siguientes listas no son sublista: [9] [3, 7, 6] [2, 1]
	 *
	 */

	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
		lista1 = new ArrayList<Integer>();
		lista1.add(0, 1);
		lista1.add(1, 2);
		lista1.add(2, 4);
		lista1.add(3, 5);
		lista1.add(4, 3);
		lista1.add(5, 6);
		lista1.add(6, 7);
	}

	@Test
	public void test1() {
		lista2 = new ArrayList<Integer>();
		lista2.add(0, 3);
		lista2.add(1, 6);
		lista2.add(2, 7);
		assertTrue(lista1.isSubList(lista2));
	}

	@Test
	public void test2() {
		lista2 = new ArrayList<Integer>();
		lista2.add(0, 2);
		lista2.add(1, 4);
```

```
                assertTrue(lista1.isSubList(lista2));
        }

        @Test
        public void test3() {
                lista2 = new ArrayList<Integer>();
                assertTrue(lista1.isSubList(lista2));
        }

        @Test
        public void test4() {
                lista2 = new ArrayList<Integer>();
                lista2.add(0, 1);
                assertTrue(lista1.isSubList(lista2));
        }

        @Test
        public void test5() {
                lista2 = new ArrayList<Integer>();
                lista2.add(0, 9);
                assertFalse(lista1.isSubList(lista2));
        }

        @Test
        public void test6() {
                lista2 = new ArrayList<Integer>();
                lista2.add(0, 3);
                lista2.add(1, 7);
                lista2.add(2, 6);
                assertFalse(lista1.isSubList(lista2));
        }

        @Test
        public void test7() {
                lista2 = new ArrayList<Integer>();
                lista2.add(0, 2);
                lista2.add(1, 1);
                assertFalse(lista1.isSubList(lista2));
        }

}
```

2) Agregar a la clase **LinkedBinaryTree.java** el siguiente método:

```
/**
 * Verifica si el árbol t es un subArbol. Es decir que el árbol t está contenido
 * dentro del árbol
 *
 * @param t árbol a verificar
 * @return true si t es una subArbol o false si no lo es.
 */
/**
 * Ejemplo, dado el árbol:
 *
 * 1
 *
 * / \
 *
 * 2 3
 *
 * / \ / \
 *
 * 4 5 6 7
 *
 * El siguiente árbol es un subArbol:
 *
 * 3
 *
 * / \
 *
 * 6 7
*/
public boolean isSubtree(LinkedBinaryTree<E> t)
```

Una posible implementación es recorrer ambos árboles en inorder, colocando sus elementos en una lista y verificar si la lista con menos elementos es una subLista de la otra. Realizar el mismo procedimiento recorriendo ambos árboles en postorder y volver a verificar si la lista con menos elementos es una subLista de la otra. Si se cumplen ambas condiciones entonces se puede afirmar que el árbol que se está probando es un subArbol.

Por ejemplo:

Lista generada al recorrer inorder del árbol {4, 2, 5, 1, 6, 3, 7}
Lista generada al recorrer inorder del árbol t {6, 3, 7}

a) Se verifica que la lista generada del árbol t es una subLista.

Lista generada al recorrer postorder del árbol {4, 5, 2, 6, 7, 3, 1}
Lista generada al recorrer postorder del árbol t {6, 7, 3}

b) Se verifica que la lista generada del árbol t es una subLista.

Si las verificaciones del punto a) y b) son verdaderas entonces el árbol t es un subArbol.

3) a) Realizar un programa que cargue un arreglo de nombres a un Map. Los nombres en el arreglo pueden estar repetidos. La clave es el nombre y el valor el número de ocurrencias en la que aparece. Mostrar los resultados.

Por ejemplo, dado el siguiente arreglo:

{"Juan", "Ana", "Ana", "Pedro", "Juan"};

El programa debe mostrar un resultado similar al siguiente:

```
<Ana, 2>
<Juan, 2>
<Pedro, 1>
```

b) Dado el Map generado en a) crear un nuevo Map donde la clave es el número de ocurrencia y el valor una lista de nombres. Mostrar los resultados.

El programa debe mostrar un resultado similar al siguiente:

```
<2, (Ana, Juan)>
<1, (Pedro)>
```

TEORIA

1. Defina que es un iterador y explique cuál es la ventaja de que una implementación de tipo abstracto de datos (TAD) lo provea.
2. Explique que es el balanceo en un árbol binario de búsqueda y de un ejemplo. Indique qué estrategias conoce para su implementación.
3. Defina el TAD mapa y explique cuál es la diferencia con un arreglo estándar.
4. ¿Cuáles son los dos conceptos que provee una tabla Hash? Explique cómo se gestionan las colisiones.

Nota: *se tendrá en cuenta la redacción correcta de las respuestas, es decir, que se entiendan los conceptos y que sintácticamente estén bien construidas.*

IMPORTANTE:

1. Para la parte práctica del parcial, subir solamente los archivos con extensión .java.

2. Para la parte teórica del parcial, subir un solo archivo de Word. (con extensión doc o .docx)