2° PARCIAL DE ALGORITMICA Y ESTRUCTURA DE DATOS II - 2020

1) Agregar a la clase **LinkedPositionalList** el siguiente método:

```
/**
 * Encuentra todas las posiciones que tiene un elemento dado
 *
 * @param e elemento a encontrar
 * @return lista posicional con todas las posiciones que contiene el elemento e
 */
  PositionalList<Position<E>> findAllPosition(E e);
```

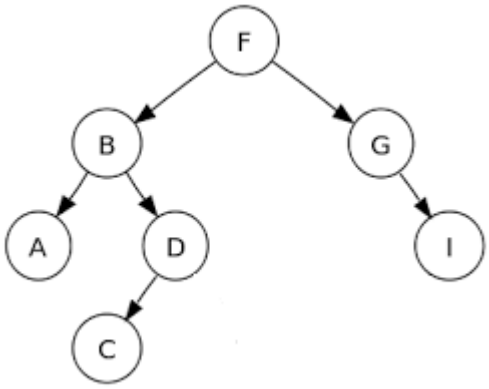
Utilizar el siguiente test de **JUnit** para probar el funcionamiento del método implementado.

```
public class TestFindAllPosition {

	private PositionalList<String> pl = new LinkedPositionalList<String>();
	private Position<String> pos1;
	private Position<String> pos2;
	private Position<String> pos3;

	@Before
	public void setUp() throws Exception {
		pos1 = pl.addFirst("Ana");
		pos2 = pl.addAfter(pos1,"Juan");
		pos3 = pl.addAfter(pos2,"Ana");
	}

	@Test
	public void testPosicionValida1() {
		PositionalList<Position<String>> f = pl.findAllPosition("Ana");
		assertEquals(f.size(),2);
		for (Position<String> p: f)
			assertEquals(p.getElement(),"Ana");
	}

	@Test
	public void testPosicionValida2() {
		PositionalList<Position<String>> f = pl.findAllPosition("Juan");
		assertEquals(f.size(),1);
		assertEquals(f.first().getElement() ,pos2);
	}

	@Test
	public void testPosicionNoValida1() {
		PositionalList<Position<String>> f = pl.findAllPosition("Pedro");
		assertEquals(f.size(),0);
	}
}
```

2) Agregar a la clase **AbstractTree** los siguientes métodos:

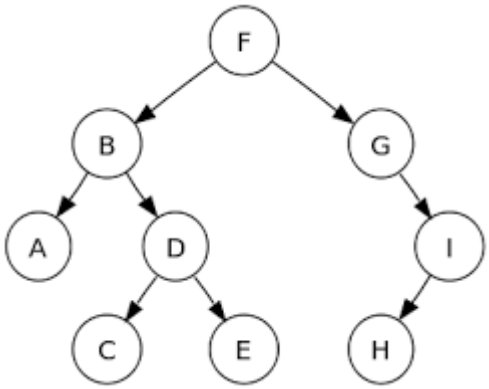
```
/**
 * Retorna un iterable con todas las posiciones de los nodos externos (hijos)
 *
 * @return iterable de las posiciones de los nodos externos
 */
 public Iterable<Position<E>> listChildren() {

/**
 * Retorna un iterable con todas las posiciones de la rama más larga de un
 * árbol comenzando desde un nodo externo hasta llegar a la raíz
 *
 * Si hay más de una rama con la misma profundidad, retorna una de las ramas que
 * cumpla con la condición dada
 *
 * @return iterable de las posiciones de rama más larga de un árbol
 */
public Iterable<Position<E>> listGreaterAncestor()
```

Ejemplo 1:



listChildren retorna una lista con las posiciones de los elementos: {A, C, I}
llistGreaterAncestor retorna una lista con las posiciones de los elementos: {C, D, B, F}

Ejemplo 2:



listChildren retorna una lista con las posiciones de los elementos: {A, C, E, H}
listGreaterAncestor retorna **una** de las siguientes listas: {C, D, B, F} {E, D, B, F} {H, I, G, F}

3) Realizar un método que retorne un **Map** con las frecuencias de cada palabra que aparece en un arreglo de String.

```
TreeMap<Integer, List<String>> wordFrec(String[] words)
```

Ejemplo: la siguiente aplicación utiliza el método `wordFrec`:

```
String wr[] = { "while", "for", "if", "for", "static", "while", "public" };

TreeMap<Integer, List<String>> frec = wordFrec(wr);

for (Entry<Integer, List<String>> f : frec.entrySet())
      System.out.println("Frecuencia: " + f.getKey() + " Palabras: " + f.getValue());
```

El resultado mostrado por consola es el siguiente:

```
Frecuencia: 1 Palabras: (static, public, if)
Frecuencia: 2 Palabras: (while, for)
```

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. Defina qué es un Tipo Abstracto de Datos y qué es su implementación. Dé un ejemplo de definición del TAD árbol binario y una implementación básica en Java.

3. Defina el TAD árbol binario de búsqueda y describa en qué consiste el proceso de balanceo desde el punto de vista de la complejidad del algoritmo de búsqueda.

IMPORTANTE:

1. Los enunciados no deberían dejar lugar a dudas de los ejercicios a resolver y preguntas a responder. De todas maneras si surge alguna consulta del enunciado enviar la misma al **Foro** que está en la **sección Evaluación**. Las preguntas serán respondidas dentro de los 30 minutos de realizadas.

2. Enviar un WhatsApp al grupo **solo** si tienen algún inconveniente (no pueden entrar al foro, no reciben la respuesta en el tiempo indicado, no pueden subir las soluciones, etc.)

3. Para la parte práctica del parcial, subir solamente los archivos con extensión .java que contiene los métodos solicitados. **No entregar todo el proyecto.**

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