Segunda Práctica Individual de ADDA

Memoria técnica del proyecto

Roberto Camino Bueno, grupo asignado número **3**

Firmado por
Roberto Camino

Índice:

1. Código con la soluciones de los problemas.

P.I. 1. Ejercicio 138 en JAVA

```
// SOLUCIÓN RECURSIVA:

public static <T extends Comparable <? super T>, E> boolean
esEquilibrado(BinaryTree<E> t) {
		boolean res=false;
		switch (t.getType()) {
		case Empty:
			res=true;
			break;
		case Leaf:
			res=true;
			break;
		case Binary:
			if(esEquilibrado(t.getLeft())&&
			  esEquilibrado(t.getRight())&&
			  (Math.abs( t.getLeft().getHeight() -  t.getRight().getHeight() )
<= 1)
			  ) {
				res = true;
			}
			break;
		}
		return res;
	}
```

He implementado un algoritmo que recibe árboles binarios con cualquier elemento y su función es evaluar si es equilibrado. Para ello es importante saber los siguientes conceptos:
- Un árbol binario es equilibrado si es vacío, es hoja o es binario: sus hijos están equilibrados y sus alturas no difieren en más de una unidad.
- Altura de un nodo: longitud del camino más largo de ese nodo a una hoja.
- Altura del árbol: altura del nodo raíz.

Dicho esto, solo queda implementar el algoritmo en base a esta definición que nos diga si es equilibrado: en caso de que el tipo esté vacío o sea hoja será cierto pero en caso binario debemos hacer la llamada recursiva mirando la parte izquierda, la parte derecha

y realizando la diferencia entre sus alturas. Recuerda que debe ser nula o 1 para considerarse equilibrado, sino el árbol binario no es equilibrado.

```
// TEST
	public static void main(String[] args) {
		BinaryTree<Character> arbolBinario1 = BinaryTree.binary(
				'A',
				BinaryTree.binary('B',
						BinaryTree.leaf('D'),
						BinaryTree.leaf('E')),
				BinaryTree.binary('C',
						BinaryTree.leaf('F'),
						BinaryTree.leaf('G'))
								);

		BinaryTree<Integer> arbolBinario2 = BinaryTree.binary(
				0,
				BinaryTree.binary(1,
						BinaryTree.empty(),
						BinaryTree.empty()),
				BinaryTree.binary(2,
						BinaryTree.leaf(5),
						BinaryTree.leaf(6))
								);

		BinaryTree<Integer> arbolBinario3 = BinaryTree.binary(
				1,
				BinaryTree.empty(),
				BinaryTree.binary(1,
						BinaryTree.leaf(1),
						BinaryTree.binary(1,
								BinaryTree.leaf(1),
								BinaryTree.leaf(1))
								));

		BinaryTree<Integer> arbolBinario4 = BinaryTree.binary(
				0,
				BinaryTree.binary(1,
						BinaryTree.leaf(3),
						BinaryTree.leaf(4)),
				BinaryTree.binary(2,
						BinaryTree.leaf(5),
```

```
                                        BinaryTree.binary(6,
                                                BinaryTree.leaf(7),
                                                BinaryTree.leaf(8))
                                                ));

                BinaryTree<String> arbolBinario5 = BinaryTree.binary(
                                "raiz",
                                BinaryTree.binary("IZQ",
                                                BinaryTree.leaf("izq"),
                                                BinaryTree.empty()),
                                BinaryTree.binary("DER",
                                                BinaryTree.leaf("izq"),
                                                BinaryTree.leaf("der"))
                                                        );

                BinaryTree<Character> arbolBinario6 = BinaryTree.binary(
                                'A',
                                BinaryTree.binary('B',
                                                BinaryTree.leaf('D'),
                                                BinaryTree.leaf('E')),
                                BinaryTree.binary('C',
                                                BinaryTree.leaf('F'),
                                                BinaryTree.binary('G',
                                                        BinaryTree.leaf('H'),
                                                        BinaryTree.binary('I',

BinaryTree.leaf('J'),

BinaryTree.leaf('K'))
                                                ));

                BinaryTree<Character> arbolBinario7 = BinaryTree.empty();

                BinaryTree<Character> arbolBinario8 = BinaryTree.leaf('A');

                System.out.println("\n EJERCICIO 1. Decidir si un árbol binario es
equilibrado. ");

                System.out.println("\n Árbol binario 1: "+arbolBinario1+". Este caso es
tipo Character, Altura: IZQ=1, DER=1, diferencia=0 (equilibrado)");
                System.out.println("\n ¿Este arbol binario 1 es equilibrado? :
"+esEquilibrado(arbolBinario1));
```

```
                System.out.println();

                System.out.println("\n Árbol binario 2: "+arbolBinario2+". Este caso es
tipo Integer, Altura: IZQ=1 (Vacíos), DER=1, diferencia=0 (equilibrado)");
                System.out.println("\n ¿Este arbol binario 2 es equilibrado? :
"+esEquilibrado(arbolBinario2));
                System.out.println();

                System.out.println("\n Árbol binario 3: "+arbolBinario3+". Este caso es
tipo Integer, Altura: IZQ=0, DER=2, diferencia=2 (NO equilibrado)");
                System.out.println("\n ¿Este arbol binario 3 es equilibrado? :
"+esEquilibrado(arbolBinario3));
                System.out.println();

                System.out.println("\n Árbol binario 4: "+arbolBinario4+". Este caso es
tipo Integer, Altura:  IZQ=1, DER=2, diferencia=1 (equilibrado)");
                System.out.println("\n ¿Este arbol binario 4 es equilibrado? :
"+esEquilibrado(arbolBinario4));
                System.out.println();

                System.out.println("\n Árbol binario 5: "+arbolBinario5+". Este caso es
tipo String, Altura: IZQ=1, DER=1, diferencia=0 (equilibrado)");
                System.out.println("\n ¿Este arbol binario 5 es equilibrado? :
"+esEquilibrado(arbolBinario5));
                System.out.println();

                System.out.println("\n Árbol binario 6: "+arbolBinario6+". Este caso es
tipo Character, Altura: IZQ=1, DER=3, diferencia=2 (NO equilibrado)");
                System.out.println("\n ¿Este arbol binario 6 es equilibrado? :
"+esEquilibrado(arbolBinario6));

                System.out.println("\n Árbol binario 7: "+arbolBinario7+". En este caso
está vacío (equilibrado)");
                System.out.println("\n ¿Este arbol binario 7 es equilibrado? :
"+esEquilibrado(arbolBinario7));

                System.out.println("\n Árbol binario 8: "+arbolBinario8+". En este caso
es una hoja raiz (equilibrado)");
                System.out.println("\n ¿Este arbol binario 8 es equilibrado? :
"+esEquilibrado(arbolBinario8));
        }
```

P.I. 4. Ejercicio 78 en JAVA

```
// SOLUCIÓN RECURSIVA:

	public static Integer busquedatrinaria(List<String> vector, String p) {
		return busquedatrinariaG(vector, 0, vector.size(), p);
	}

	private static Integer busquedatrinariaG(List<String> vector, Integer start,
Integer end, String p) {
		Integer res = null;
		if (end.equals(start)) {
			res = -1;
		} else {
			Integer puntomed1 = start + (end - start) / 3;
			Integer puntomed2 = start + 2 * (end - start) / 3;

			if (p.compareTo(vector.get(puntomed1)) == 0) {
				res = puntomed1;
			} else if (p.compareTo(vector.get(puntomed2)) == 0) {
				res = puntomed2;
			} else if (p.compareTo(vector.get(puntomed1)) < 0) {
				res = busquedatrinariaG(vector, start, puntomed1, p);
			} else if (p.compareTo(vector.get(puntomed2)) < 0) {
				res = busquedatrinariaG(vector, puntomed1 + 1,
puntomed2, p);
			} else if (p.compareTo(vector.get(puntomed2)) > 0) {
				res = busquedatrinariaG(vector, puntomed2 + 1, end, p);
			}
		}
		return res;
	}
```

El método público recibe una lista de Strings y la palabra a buscar, entonces inicializa el privado. En el privado tenemos el caso base cuando el principio de la lista y el final son la misma posición que devuelve -1, sino entramos en lo interesante del problema. Definimos 2 pivotes, punto medio 1 y 2, con posiciones céntricas en su mitad del vector, así nos dividen el vector en 3 partes y realizaremos búsquedas recursivas en cada intervalo a excepción de que la palabra coincida en la posición del pivote, también debemos contemplar esa posibilidad y entonces devolverla.

Las llamadas recursivas van actualizando el principio y final de la lista gracias a los pivotes hasta encontrar el elemento que busco.

Comentario extra: Este método también puede implementar un Comparador que nos ahorra poner compareTo en cada else if y además podría comparar elementos genéricos. Para ello ponemos en el método público:
Comparator<E> cmp = Comparator.naturalOrder();
A continuación de los pivotes:
Integer c1 = cmp.compare(p, vector.get(puntomed1));
Integer c2 = cmp.compare(p, vector.get(puntomed2));
Y ahora podemos usarlos por ejemplo cuando coincide con pivote 1:
if (c1 == 0) {
res = puntomed1;
}

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

		System.out.println("\n EJERCICIO 4. Encontrar la posición de la palabra dada P y si no está devolverá -1. ");

		List<String> nombres = Arrays.asList("Alejandro", "Beatriz", "Carlos", "Diego", "Eugenia", "Federico",
				"Gonzalo", "Hector", "Ines");
		System.out.println("\n -PRIMERO CASO. Dada la siguiente lista ordenada con palabras de la A a la I consecutivas: \n" +
				nombres + ", buscamos todas las posiciones:");
		System.out.println("\n Posición de la palabra Alejandro en la lista: " + busquedatrinaria(nombres, "Alejandro"));
		System.out.println("\n Posición de la palabra Beatriz en la lista: " + busquedatrinaria(nombres, "Beatriz"));
		System.out.println("\n Posición de la palabra Carlos en la lista: " + busquedatrinaria(nombres, "Carlos"));
		System.out.println("\n Posición de la palabra Diego en la lista: " + busquedatrinaria(nombres, "Diego"));
		System.out.println("\n Posición de la palabra Eugenia en la lista: " + busquedatrinaria(nombres, "Eugenia"));
		System.out.println("\n Posición de la palabra Federico en la lista: " + busquedatrinaria(nombres, "Federico"));
		System.out.println("\n Posición de la palabra Gonzalo en la lista: " + busquedatrinaria(nombres, "Gonzalo"));
```

```
		System.out.println("\n Posición de la palabra Hector en la lista: " +
busquedatrinaria(nombres, "Hector"));
		System.out.println("\n Posición de la palabra Ines en la lista: " +
busquedatrinaria(nombres, "Ines"));
		System.out.println("\n Posición de la palabra Zidane (no esta, devolvera
-1) en la lista: " +
		busquedatrinaria(nombres, "Zidane"));

		List<String> palabras = Arrays.asList("anillo",  "cocido", "repetido",
"repetido");
		System.out.println("\n\n -SEGUNDO CASO. Dada la siguiente lista
ordenada con palabras no consecutivas y repetidas: " +
		palabras);
		System.out.println("\n Posición de la palabra repetido en la lista: " +
busquedatrinaria(palabras, "repetido")+
				". Como ve devuelve la primera posicion encontrada");

		List<String> nombres1 = Arrays.asList("Alejandro");
		System.out.println("\n\n -TERCER CASO. Solo 1 palabra " +
nombres1);
		System.out.println("\n Posición de la palabra Alejandro en la lista: " +
busquedatrinaria(nombres1, "Alejandro"));

		List<String> nombres2 = Arrays.asList("Alejandro", "Beatriz");
		System.out.println("\n\n -CUARTO CASO. Solo 2 palabras " +
nombres2);
		System.out.println("\n Posición de la palabra Alejandro en la lista: " +
busquedatrinaria(nombres2, "Alejandro"));
		System.out.println("\n Posición de la palabra Beatriz en la lista: " +
busquedatrinaria(nombres2, "Beatriz"));


		List<String> nombres3 = Arrays.asList("Alejandro", "Beatriz",
"Carlos");
		System.out.println("\n\n -QUINTO CASO. Solo 3 palabras " +
nombres3);
		System.out.println("\n Posición de la palabra Alejandro en la lista: " +
busquedatrinaria(nombres3, "Alejandro"));
		System.out.println("\n Posición de la palabra Beatriz en la lista: " +
busquedatrinaria(nombres3, "Beatriz"));
		System.out.println("\n Posición de la palabra Carlos en la lista: " +
busquedatrinaria(nombres3, "Carlos"));
```

```
                List<String> nombres4 = Arrays.asList();
                System.out.println("\n\n -SEXTO CASO. No hay palabras " +
nombres4);
                System.out.println("\n Posición de la palabra Alejandro en la lista: " +
busquedatrinaria(nombres4, "Alejandro"));


                List<String> palabras2 = Arrays.asList("Alejandro", "Beatriz",
"Carlos","Paco","Por favor", "Quiero", "Sobresaliente");
                System.out.println("\n\n -ULTIMO CASO; INTERACTIVO. Dada la
siguiente lista ordenada con palabras no consecutivas: "+palabras2+"."
                                + " \n Por favor, inserte a continuación la palabra que
quiere buscar: \n");
                @SuppressWarnings("resource")
                Scanner sc = new Scanner (System.in);
                System.out.println("\n La palabra buscada se encuenta en la posicion
"+busquedatrinaria(palabras2, sc.nextLine()));
        }
```

2. Definir los tamaños y calcular los T(n) para las distintas funciones implementadas considerando los casos mejor y peor.

P.I. 1. Ejercicio 138

La complejidad del problema 1 reside en la altura del árbol que vamos a recorrer ya que debemos visitar hasta el último nodo, es decir, la altura por la parte derecha y por la parte izquierda. Por tanto tiene una complejidad de orden **lineal**.

El **tamaño** n será el caso base: **n = t.getHight() - i_nodo**

Entonces el **orden de complejidad** al hacer llamadas recursivas por la izquierda y por la derecha del árbol es: **T(n)** = 2*T(n-1) + k **∈ O(n)**

Para el **caso mejor** será cuando analiza el caso Empty o el caso Leaf donde ambos tienen el mismo orden de complejidad: **T(n)** = k **∈ O(1)**

Para el **caso peor** será entrar en el caso Binary: **T(n)** = 2*T(n-1) + k **∈ O(n)**

Comentario extra: Me gustaría mencionar que si el árbol tuviera etiquetas por por cada nodo con su altura, el problema se reduciría a comparar la etiqueta del mayor nodo y sería más simple, de manera que la complejidad sería de orden constante.

P.I. 4. Ejercicio 78

La complejidad del problema 4 reside en cuántas veces puedo dividir el vector por 3 intervalos hasta que tenga la posición del elemento que busco. El algoritmo descarta ⅔ del vector busca en el tercio correspondiente: O(log3(n)). Por tanto tiene una complejidad de orden **logarítmica**.

El **tamaño** n será el caso base: **n = end - start**

Entonces el **orden de complejidad** es: **T(n)** = (log(n) / log(3)) + k **∈ O(log(n))**

Para el **caso mejor** sería encontrar la posición del elemento nada más comparar, es decir, que fuese coincidiese la posición con un pivote o caso base: **T(n)** = k **∈ O(1)**

Para el **caso peor** sería entrar en las búsquedas recursivas: **T(n)** = **O(log(n))**

3. Volcado de pantalla con los resultados obtenidos en las pruebas realizadas.

P.I. 1. Ejercicio 138

El test sobre este ejercicio consiste en evaluar si son equilibrados 8 árboles binarios distintos.

```
EJERCICIO 1. Decidir si un árbol binario es equilibrado.
Árbol binario 1: A(B(D,E),C(F,G)). Este caso es tipo Character, Altura: IZQ=1, DER=1, diferencia=0 (equilibrado)
¿Este arbol binario 1 es equilibrado? : true

Árbol binario 2: 0(1,2(5,6)). Este caso es tipo Integer, Altura: IZQ=1 (Vacíos), DER=1, diferencia=0 (equilibrado)
¿Este arbol binario 2 es equilibrado? : true

Árbol binario 3: 1(_,1(1,1(1,1))). Este caso es tipo Integer, Altura: IZQ=0, DER=2, diferencia=2 (NO equilibrado)
¿Este arbol binario 3 es equilibrado? : false
Árbol binario 4: 0(1(3,4),2(5,6(7,8))). Este caso es tipo Integer, Altura:  IZQ=1, DER=2, diferencia=1 (equilibrado)
¿Este arbol binario 4 es equilibrado? : true

Árbol binario 5: raiz(IZQ(izq,_),DER(izq,der)). Este caso es tipo String, Altura: IZQ=1, DER=1, diferencia=0 (equilibrado)
¿Este arbol binario 5 es equilibrado? : true

Árbol binario 6: A(B(D,E),C(F,G(H,I(J,K)))). Este caso es tipo Character, Altura: IZQ=1, DER=3, diferencia=2 (NO equilibrado)
¿Este arbol binario 6 es equilibrado? : false
Árbol binario 7: _. En este caso está vacío (equilibrado)
¿Este arbol binario 7 es equilibrado? : true
Árbol binario 8: A. En este caso es una hoja raiz (equilibrado)
¿Este arbol binario 8 es equilibrado? : true
```

P.I. 4. Ejercicio 78

El test sobre este ejercicio consta de 7 casos donde el último le pide que introduzca la palabra que desea buscar.

```
 EJERCICIO 4. Encontrar la posición de la palabra dada P y si no está devolverá -1.

 -PRIMERO CASO. Dada la siguiente lista ordenada con palabras de la A a la I consecutivas:
[Alejandro, Beatriz, Carlos, Diego, Eugenia, Federico, Gonzalo, Hector, Ines], buscamos todas las posiciones:

 Posición de la palabra Alejandro en la lista: 0

 Posición de la palabra Beatriz en la lista: 1

 Posición de la palabra Carlos en la lista: 2

 Posición de la palabra Diego en la lista: 3

 Posición de la palabra Eugenia en la lista: 4

 Posición de la palabra Federico en la lista: 5

 Posición de la palabra Gonzalo en la lista: 6

 Posición de la palabra Hector en la lista: 7

 Posición de la palabra Ines en la lista: 8

 Posición de la palabra Zidane (no esta, devolvera -1) en la lista: -1


 -TERCER CASO. Solo 1 palabra [Alejandro]

 Posición de la palabra Alejandro en la lista: 0


 -CUARTO CASO. Solo 2 palabras [Alejandro, Beatriz]

 Posición de la palabra Alejandro en la lista: 0

 Posición de la palabra Beatriz en la lista: 1


 -QUINTO CASO. Solo 3 palabras [Alejandro, Beatriz, Carlos]

 Posición de la palabra Alejandro en la lista: 0

 Posición de la palabra Beatriz en la lista: 1

 Posición de la palabra Carlos en la lista: 2


 -SEXTO CASO. No hay palabras []

 Posición de la palabra Alejandro en la lista: -1


 -ULTIMO CASO; INTERACTIVO. Dada la siguiente lista ordenada con palabras no consecutivas: [Alejandro, Beatriz, Carlos, Paco, Por favor, Quiero, Sobresaliente].
 Por favor, inserte a continuación la palabra que quiere buscar:

Paco

 La palabra buscada se encuenta en la posicion 3
```