Primer Parcial
Grado en Ingeniería Informática
Curso 2017-2018
Estructura de Datos y Algoritmos
Grupo 81M, 7 Marzo 2018

Nombre y apellidos:

Problema 1

```
public class DNode {

	public Integer elem;
	public DNode prev;
	public DNode next;
	public DNode(Integer elem) {
		this.elem = elem;
	}
}

public class DList {
	DNode header;
	DNode trailer;
	int size=0;

	public DList() {
		header = new DNode(null);
		trailer = new DNode(null);
		header.next = trailer;
		trailer.prev= header;
	}


	}
```

A. (0,40 pto.) DList es una clase que implementa una lista doblemente enlazada de enteros. Añadir un método (no estático) isBinary() que devuelva un boolean que determine si los enteros (contenidos en el objeto lista) son solo 0 o 1. Por ejemplo si la lista contiene los enteros 4,3,1,0 devolvería false, mientras que si la lista es 1,0,0,1 devolvería true.

```
public boolean isBinary () {
	boolean esBin=true;
	for (DNode nodeIt = header.next; nodeIt != trailer && esBin==
true; nodeIt = nodeIt.next) {
		if (!nodeIt.elem.equals(0) && !nodeIt.elem.equals(1)) {
			esBin=false;
		}
	}
	return esBin;
	}
```

B. (0,80 pto.) Un número binario es ondulado cuando alterna el 0 y el 1, por ejemplo 0101 o 101 son números ondulados, pero 011 no es ondulado porque tiene dos 1 consecutivos. Dada Dlist, lista doblemente enlazada, de enteros. Añade un método (no estático) undulate() que transforme el objeto lista para que el número binario que contiene sea ondulado. No se permite modificar el valor de ningún nodo (es decir, no puedes modificar el valor de elem), sin embargo, cuando lo necesites si deberás insertar un nuevo nodo (no se pueden utilizar llamadas a métodos como insertAt(), getIndexOf(), …), que incluya el valor 0 o 1, según sea necesario. De este modo, la lista 0,1,1 se transformará en 0,1,0,1 (mediante la inserción de un nodo con valor 0 en la posición 2). En el caso de la lista: 1,1,1,0, se transformará en la siguiente lista 101010, en la que hemos tenido que insertar dos nodos con valor 0 en la posición 1 y en la posición 3.

Como un primer paso del algoritmo, se recomienda comprobar que el número contenido en la lista es binario. En caso de no serlo, el método sólo muestra un mensaje y no hace nada más.

```
public void undulate () {
            if (isEmpty()) {
                  System.out.println("la lista está vacía, no se puede
generar un numero binario ondulado");
            } else if (isBinary ()) {
            for (DNode nodeIt = header.next; nodeIt != trailer.prev;
nodeIt = nodeIt.next) {
                  if (nodeIt.elem.equals(nodeIt.next.elem)) {
                        Integer numInsertar=null;
                        if (nodeIt.elem.equals(1)) {
                                    numInsertar=0;
                        } else numInsertar=1;
                  //ahora insertamos un Nuevo nodo entre el nodo actual
                  //y el siguiente
                        DNode newNode = new DNode (numInsertar);
                        newNode.next =nodeIt.next;
                        newNode.prev = nodeIt;
                        nodeIt.next.prev = newNode;
                        nodeIt.next = newNode;
                        size++;
                  }
            }
            } else {
                  System.out.println("No es un número binario, no se
puede generar un numero binario ondulado");}
            }
```

C. (0,30 pto.) Explicar brevemente la complejidad (Big-O) del método undulate (). No tienes que calcular T(n).

En el peor de los casos, el método tendrá que comprobar que la lista contiene un número binario (O(n)) y visitar nodo por nodo para comparar con su siguiente (comparas los elementos), siendo la complejidad del bucle O(n). Por tanto, la complejidad final es O(n), lineal.

Problema2

A. (0,20 pto.) Escribir un método recursivo que dado un número n>0, devuelva la suma de los dobles de los números desde 1 a n (ambos incluidos).

Ejemplo: SumaDoble (3) = 2*1+2*2+2*3 =12

```
public static int sumaDoble(int n) { //n>0
      if (n==1) return 2;
      else return 2*n + sumaDoble(n-1);
 }
```

B. (0,30 pto.) Escribir un método recursivo que verifique si una palabra de longitud impar es capicúa (por ejemplo, *radar*, *rajar*, *dañad*, *seres*, *nadan*). Recibe como parámetro un String, y devuelve un boolean.

```
public static boolean checkPalindrome(String s) {
            if (s==null || s.length==0) return true;
            else return checkPalindrome(s,0,s.length()-1);
 }

private static boolean checkPalindrome(String s, int left, int right)
{
            if (left==right) return true;

            else if (left<right) {
                  return s.charAt(left)==s.charAt(right) &&
checkPalindrome(s, left+1, right-1);
            } return false;
      }
}
```