ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA DEL SOFTWARE

INGENIERÍA DEL SOFTWARE
GRUPO 3

ANÁLISIS Y DISEÑO DE DATOS Y ALGORITMOS
ESTRUCTURA DE DATOS Y ALGORITMOS

PRÁCTICA 1: Comprobado de integridad

FERNANDO RAMÍREZ VILLALBA

Índice

1 Enunciado

Un algoritmo, *comprobarIntegemplridad*, permita determinar la integridad de un texto en una comunicación. Para realizar dicha comprobación, las cadenas de texto que se reciben han sido marcadas de forma especial antes de ser enviadas. Veamos un ejemplo del texto:

"Esto es un text securizado$2516 "

Como puede comprobarse la cadena tiene una marca $2516, la marca $ indica el final de la cadena y el 2516 es la suma del valor numérico ascii de cada carácter de la cadena de texto.

El algoritmo hace uso de una función *valorNumerico* que determine el valor numérico de la cadena; hasta el carácter $ (sin incluir este). Posteriormente comparará el valor obtenido por esta función con el valor tras el carácter $. Si ambos valores coinciden quiere decir que el texto está íntegro y no ha sido modificado en la comunicación y el algoritmo devolverá *true*, en cualquier otro caso el texto ha sido comprometido devolverá *false*.

1.1 Cuestiones a resolver

1. Defina la función recursiva *valorNumerico* que dado una cadena de texto marcada devuelva su valor numérico entero.

$$valorNumerico\ (String\ s) = valorNumerico\ (String\ s, Integer\ n)$$

$$valorNumerico\ (s, n) \begin{cases} 0 & , n == 0 \\ valor(s.charAt(n)) + valorNumerico(n-1) & , n > 0 \end{cases}$$

2. Implemente una función **recursiva no final** empleando el lenguaje de programación que le ha indicado su profesor que solucione el problema planteado, partiendo de la función definida previamente.

3. Implemente el algoritmo *comprobarIntegridad;* empleando el lenguaje de programación que le ha indicado su profesor, que dado una cadena de caracteres de entrada y haciendo uso de la implementación de la función recursiva no final del apartado anterior, determine si la cadena es íntegra o no.

4. Implemente una función **recursiva final** a partir de la definición del aparatado 2.

5. A partir de las funciones anteriores, diseñar e implementar un **algoritmo iterativo** que proporcione el valor deseado.

6. Indique el tamaño del problema, T(n) y la complejidad del algoritmo implementado en el apartado 2.

$$T(n) = \begin{cases} 0 & , n == s.indexOf('\$') \\ valor\ (s.charAt(n)) + T(n-1) & , n\ != s.indexOf('\$') \end{cases}$$

El orden de complejidad lo determinamos con la ecuación:

$$T(n)=T(n-1) + k$$

donde podemos determinar los valores de $a=1$, $b=1$, $c=1$ y $d=0$. Con estos datos y con la hoja de ayuda podemos calcular el orden de complejidad, que sería O(n).

2 Seguimiento

2.1 Control de seguimientos y tutorías

El alumno debe listar y detallar las visitas a tutoría.

Fecha	Profesor	Detalles y dudas resueltas
04/11/2015	Ángel	Corrección del profesor en el código: utilizar el código dado en el problema y no otro.

2.2 Portafolio

Agenda de seguimiento en la que el alumno deberá registrar toda la actividad que realice para resolver el ejercicio, así como los problemas encontrados.

Fecha	Detalles y dudas resueltas
26/10/2015	• Búsqueda en internet como pasar de String a código ASCII. • Código iterativo hecho a falta de comparar la suma del String en ASCII y el número detrás del $. • Métodos de apoyo: separaTexto y separaCódigo. • Repaso de Java, ya que estoy un poco oxidado.
02/11/2015	• Esquema recursivo del ejercicio e implementación del código en Java.
04/11/2015	• Comienzo del documento de entrega mientras pasa el profesor el seguimiento de la práctica a los demás alumnos.
09/11/2015	• Continuación del documento escrito y corrección del código java, quito la implementación del paso de un *char* a *ASCII* y utilizo el método dado en el ejercicio. • Implementación del código recursivo final.
15/11/2015	• Reescritura del código en java debido a que no encuentro la forma de saber la complejidad del método splitter de guava. • No ha necesidad de métodos auxiliares, debido al código nuevo.
16/11/2015	• Completado el documento para entregar completando las partes de portfolio, análisis y complejidad y las pruebas y trazas. • Últimas pruebas con el algoritmo realizado para completar los ejercicios faltantes en el documento para entrega.
19/11/2015	• Realizado una batería de pruebas expuesto al final del documento y entrega de la práctica.

3 Implementación

Código del ejercicio 2 *valorNumericoRecursivo*:

```
public static Integer valorNumerico(String s) {

	return valorNumericoRecursivo(s, 0);
}

private static Integer valorNumericoRecursivo(String s, Integer n) {
	Integer res;

	if (n == s.indexOf('$')) {
		res = 0;
	} else {
		res = valor(s.charAt(n)) + valorNumericoRecursivo(s, n +
			1);
	}

	return res;
}
```

Código del ejercicio 3 *comprobarIntegridad*:

```
public static Boolean comprobarIntegridad(String s) {

	Boolean res;

	if (valorNumerico(s).equals(
			Integer.parseInt(s.substring(s.indexOf('$') + 1)))){
		res = true;
	} else
		res = false;
	return res;
}
```

Código del ejercicio 4 *valorNumericoFinal*:

```
public static Integer valorNumericoFinal(String s) {

	return valorNumericoRecursivoFinal(s, 0, 0);
}

private static Integer valorNumericoRecursivoFinal(String s, Integer
		n, Integer acum) {
	Integer res;

	if (n == s.indexOf('$')) {
		res = acum;
	} else {
		res = valorNumericoRecursivoFinal(s, n + 1,
				acum + valor(s.charAt(n)));
	}
	return res;
}
```

Código del ejercicio 5:

```
public static Integer valorNumericoIterativo(String s) {
	Integer res = 0;

	for (int i = 0; i < s.indexOf('$'); i++) {
		res += valor(s.charAt(i));
	}
	return res;
}
```

4 Análisis y complejidad

En los ejercicios anteriores hemos podido analizar el ***algoritmo recursivo no final*** y definir el orden de complejidad.

El orden de complejidad del ***algoritmo recursivo final*** tiene exactamente el mismo orden de complejidad pudiendo cambiar únicamente el factor multiplicativo por las constantes que podamos tener, pero se puede despreciar en estos casos.

El orden de complejidad del ***algoritmo iterativo*** lo analizamos a continuación:

Definimos n como: $n = s.indexOf('\$')$

$$T(n) = \sum_{i=0+1x}^{n} k = n$$

$$T(n) = n \rightarrow O(n)$$

Nuevamente obtenemos un orden de complejidad lineal $O(n)$.

5 Pruebas y trazas

Indicar las pruebas realizados a los algoritmos desarrollados y ejemplos de uso.

Traza del algoritmo ***recursivo no final***:

valorNumerico('Esto$2516')= 0 + valorNumericoRecursivo('Esto$2516', 0)= 0 + valorNumericoRecursivo('Esto$2516', 0)= 0 + valorNumericoRecursivo('sto$2516', 69)= 69 + valorNumericoRecursivo('to$2516', 115)= 69 + 115 + valorNumericoRecursivo('o$2516', 116)= 69 + 115 + 116 + valorNumericoRecursivo('$2516', 111)= 69 + 115 + 116 + 111 + valorNumericoRecursivo('$2516', 0)= 69 + 115 + 116 + 111 = 411

Traza del algoritmo ***recursivo final***:

valorNumerico('Esto$2516')= valorNumericoRecursivo('Esto$2516', 0)= valorNumericoRecursivo('Esto$2516', 0)= valorNumericoRecursivo('sto$2516', 69)= valorNumericoRecursivo('to$2516', 69 + 115)= valorNumericoRecursivo('o$2516', 69 + 115 + 116)= valorNumericoRecursivo('$2516', 69 + 115 + 116 + 111)= valorNumericoRecursivo('$2516', 411)

Traza del ***algoritmo iterativo***: En este caso comprobaremos como se actualizan las variables.

valorNumericoIterativo('Esto$2516')

Variable\Iteración	0	1	2	3	4
i	0	1	2	3	4
a	0	69	115	116	111
res	0	69	184	300	411

A continuación vamos a mostrar una serie de pruebas realizadas para comprobar que el código escrito funciona:

String de entrada en los algoritmos: “Esto es un text securizado$2516”

Texto para comprobar: Esto es un text securizado

Código recibido: 2516

Valor Numérico Recursivo: 2516

Valor Numérico Recursivo Final: 2516

Valor Numérico Iterativo: 2516

¿El texto es íntegro? true

String de entrada en los algoritmos: “Esto es un texto securizado$2516”

Texto para comprobar: Esto es un texto securizado

Código recibido: 2516

Valor Numérico Recursivo: 2627

Valor Numérico Recursivo Final: 2627

Valor Numérico Iterativo: 2627

¿El texto es íntegro? false

String de entrada en los algoritmos: “ $32”

Texto para comprobar:

Código recibido: 32

Valor Numérico Recursivo: 32

Valor Numérico Recursivo Final: 32

Valor Numérico Iterativo: 32

¿El texto es íntegro? True

String de entrada en los algoritmos: “ñ$2”

Texto para comprobar: ñ

Código recibido: 2

Valor Numérico Recursivo: 241

Valor Numérico Recursivo Final: 241

Valor Numérico Iterativo: 241

¿El texto es íntegro? False

String de entrada en los algoritmos: “No es lo mismo que$1632”

Texto para comprobar: No es lo mismo que

Código recibido: 1632

Valor Numérico Recursivo: 1632

Valor Numérico Recursivo Final: 1632

Valor Numérico Iterativo: 1632

¿El texto es íntegro? True

String de entrada en los algoritmos: “Ñ$1632”

Texto para comprobar: Ñ

Código recibido: 1632

Valor Numérico Recursivo: 209

Valor Numérico Recursivo Final: 209

Valor Numérico Iterativo: 209

¿El texto es íntegro? false