	<p style="text-align: center;">UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS</p>	<p>Código: ST245</p> <hr/> <p>Estructura de Datos 1</p>
---	--	---

Laboratorio Nro. I: Recursión

Isabella Arango Restrepo

Universidad Eafit
Medellín, Colombia
iarangor1@eafit.edu.co

Juan David Rengifo Castro

Universidad Eafit
Medellín, Colombia
jdrengifoc@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. Expliquen con sus propias palabras cómo funciona el ejercicio GroupSum5

El ejercicio GroupSum5 funciona con recursive backtracking. El código busca encontrar una forma de sumar los elementos del arreglo desde cierta posición hasta que alcancen un target siguiendo dos restricciones: si uno de los elementos es 5 y el que sigue es 1, entonces no los cuenta en la suma. Este funciona de la siguiente manera: primero revisa si el número start ingresado es menor que el tamaño del arreglo, de no ser así retornara falso si el target no es cero. Después, revisa que el número del arreglo que se encuentra en la posición start es igual a 5 y en ese caso mira si en la siguiente posición es un 1. Si esto pasa, entonces hará un llamado recursivo, pero incrementando el start+2, pues el 5 y el 1 no contarán ni afectarán al target. De lo contrario, el llamado recursivo se hará con el start+1 y restándole al target el 5. Luego, si el arreglo en la posición start es diferente de 5, entonces hay dos opciones, que el número en esa posición se tome o no se tome, por lo tanto, hay dos opciones de llamados: con start+1 y restándole el numero al target, o con start+1, pero con el target sin ser modificado. Finalmente, el start llegará a ser mayor que el tamaño del arreglo y por lo tanto se retornara falso si no se cumple que $target == 0$ o verdadero si lo hace.

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

2. Calculen la complejidad de los Ejercicios en Línea de los numerales 2.1 y 2.2 y agréguela al informe PDF

2.1.

- a) **changeXY**: $T(n) = T(n-1) + C$
- b) **Array6**: $T(n) = T(n-1) + C$
- c) **stringClean**: $T(n) = T(n-1) + C$
- d) **nestParen**: $T(n) = T(n-2) + C$
- e) **count8**: $T(n) = T(n/10) + C$

2.2

- a) **groupSum6**: $T(n) = 2T(n-1) + C$
- b) **groupNoAdj**: $T(n) = T(n-2) + T(n-1) + C$
- c) **groupSum5**: $T(n) = 2T(n-1) + C$
- d) **groupSumClump**: $T(n) = 2T(n-1) + C$
- e) **splitArray**: $T(n) = 2T(n-1) + C$

3. Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del ejercicio anterior

2.1.

- f) **changeXY**: n es la longitud de la cadena en cuestión.
- g) **Array6**: n es la longitud del arreglo de enteros.
- h) **stringClean**: n es la longitud de la cadena en cuestión.
- i) **nestParen**: n es la longitud de la cadena en cuestión.
- j) **count8**: n es el valor del entero

2.2

- f) **groupSum6**: n es la longitud del arreglo de enteros.
- g) **groupNoAdj**: n es la longitud del arreglo de enteros.
- h) **groupSum5**: n es la longitud del arreglo de enteros.

i) **groupSumClump**: n es la longitud del arreglo de enteros.

j) **splitArray**: n es la longitud del arreglo de enteros.

4. ¿Qué aprendieron sobre Stack Overflow? ¿Por qué sucede este error?

El Stack Overflow es un error común cuando se trabaja la recursión en programación. Este error consiste principalmente en que se excede el número de llamados de una clase o un método recursivo, llenando de esta manera todo el Stack y generando el StackOverflowError.

5. ¿Cuál es el valor más grande que pudo calcular para Fibonacci? ¿Por qué? ¿Por qué no se puede ejecutar Fibonacci con 1 millón?

El valor más grande que pudimos calcular para Fibonacci fue el de la posición 55, es decir, cuando el parámetro n tomaba el número 54, obteniendo 368225352 como resultado. Después de esta cantidad no pudimos calcular más valores para Fibonacci, pues el programa ya se tomaba demasiado tiempo para retornar una cifra. No se puede ejecutar Fibonacci con un millón porque se genera el java.lang.StackOverflowError, pues aunque tenga el caso base bien definido, hacer el llamado un millón de veces hace que el Stack se llene completamente y no haya espacio suficiente para el resto de llamadas.

6. ¿Qué concluyen sobre la complejidad de los problemas de CodingBat Recursion 1 con respecto a los de Recursion 2?

Es claro que la dificultad de los ejercicios de Recursion 2 son más difíciles de resolver que los de Recursion 1. Esta dificultad, además de que es empírica para el programador, se evidencia a la hora de calcular la complejidad de estos problemas. La complejidad de los ejercicios de la parte 2 es superior y en el mayor de los casos el doble. En efecto, en recursión 2 en ocasiones es necesario llamar a un método auxiliar, existen más variables y siempre se requiere poner un return compuesto por una disyunción.


4) Simulacro de Parcial

1. (Opcional) Start+1, nums, target
2. B
3. 3.1.
3.2.
3.3.
4. (Opcional) e) La suma de los elementos del arreglo a y es $O(n)$.
5. 5.1. Línea 2. return n;
Línea 3. n-1
Línea 4. n-2
5.2. b
6. 6.1 sumaAux(n , i+2)
6.2 sumaAux(n, i+1);
7. (Opcional) 7.1 S, i + 1, t-S[i]
7.2 S, i+1, t
8. 8.1. return 0;
8.2 ni+nj;

5) Lectura recomendada (opcional)

RECURSIÓN

En este capítulo el autor comienza hablando sobre la recursión y menciona que cualquier función que se llame a si misma es recursiva y que está ejecuta sus tareas dividiéndolas en subtareas hasta que llega a un “caso base” donde la función se detiene. Paralela a la recursión está la iteración la cual aunque es similar tiene sus diferencias y cada proceso tiene sus ventajas y desventajas. Por su parte, la recursión termina cuando el caso base es alcanzado, cada llamado recursivo requiere un espacio en el stack frame (la memoria), por lo tanto la

	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Código: ST245
		Estructura de Datos 1

recursión tiene un límite, sin embargo, algunos problemas se formulan más fácil recursivamente. Por el contrario, la iteración se detiene cuando la condición se convierte en falsa y como ventaja cada iteración no requiere espacio extra, por lo que pueden ser infinitos, sin embargo, las soluciones de este tipo en ocasiones no son tan obvias como las recursivas.

Finalmente el autor habla sobre el “backtraking”, el cual podría definirse como una “fuerza bruta 2.0”, es decir, en vez de probar con todas las combinaciones posibles, elimina las opciones que claramente no son posibles.