Laboratorio Nro. 1: Recursión

Camila Barona Cabrera
Universidad Eafit
Medellín, Colombia
cbaronac1@eafit.edu.co

Mariana Gómez Piedrahita
Universidad Eafit
Medellín, Colombia
mgomezp10@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1
En el ejercicio nos dan 3 variables, la primera en un entero que guarda la posición desde donde se empieza a recorrer el arreglo, la segunda es un arreglo tipo entero y la ultima es un entero el cual nos trae el valor de la suma que debemos conseguir. En general, lo que se requiere es saber si sí se puede con los valores que hay dentro del arreglo sumar la cantidad que nos da una variable; pero para esto hay unas condiciones que se deben cumplir, primeramente, se deben meter a la suma los números múltiplos de 5 y si el valor que le sigue al múltiplo de 5 es 1, no cuenta

Explicación:
La decisión de parada es que cuando la variable que recorre el arreglo (se podría decir que como un contador) sea igual o mayor a la longitud del arreglo, la suma sea cero, es decir que ya hayamos encontrado los números de la suma total.
Hay una primera decisión en la que se hacen dos cosas:
El llamado recursivo, en el que aumentamos en 1 el “contador”, restamos lo que haya en la posición a la suma total (target) sí y solo sí la llamada al método(checkOne) resulta true, no da true si el valor de la posición en la que estamos parados es múltiplo de 5 y la posición siguiente es diferente de 1, es verdadero si el valor de la posición en la que estamos parados es múltiplo de 5 y el valor de la siguiente posición es diferente de 1.
Luego, hacemos un segundo llamado recursivo, primero, verificamos si la posición en la que estamos parados es diferente de 5 y luego hacemos el llamado recursivo pasándole el contador+1, el arreglo y la suma.
El programa sigue haciendo todo esto, hasta que encontremos los números dentro del arreglo que suman lo pedido por el “target”, es decir, cuando la decisión de parada se cumpla.

3.2
1.BunnyEars2:

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ c_2 + T(n-1) & \text{if } n > 0 \end{cases}$$

Respuesta
T(n)= c_2*n+c_1
Notación O
O(c_2*n)
O(n)

2. Triangle:

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ c_2 + T(n-1) & \text{if } n > 0 \end{cases}$$

Respuesta
T(n)= c_2*n+c_1
Notación O
O(c_2*n)

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

O(n)

3. SumDigits:

$$T(n) = \begin{cases} c_1 & \text{if } n < 10 \\ c_2 + T(n/10) & \text{if } n >= 10 \end{cases}$$

Respuesta
$T(n) = \frac{c_2 \log(n)}{\log(10)} + c_1$ (c_1 is an arbitrary parameter)

Notación O
O($\frac{c_2 \log(n)}{\log(10)}$)
O(log(n))

4. CountHi:

$$T(n) = \begin{cases} c_1 & \text{if } n < 2 \\ T(n-1) & \text{if } n >= 2 \end{cases}$$

Notación O
O(c1)
O(1)

Respuesta
T(n)=c1

5. Count7:

$$T(n) = \begin{cases} c_1 & \text{if } n < 2 \\ T(n-1) & \text{if } n >= 2 \end{cases}$$

Notación O
O($\frac{c_2 \log(n)}{\log(10)}$)
O(log(n))

Answer

$T(n) = \frac{c_2 \log(n)}{\log(10)} + c_1$ (c_1 is an arbitrary parameter)

RESOURCE 2:

1.GroupSum6:

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n-1) + c_2 & \text{if } n > 1 \end{cases}$$

Notación O
O ($c_1\, 2^{n-1}$)
O(2^{n-1})

Respuesta

$T(n) = c_1\, 2^{n-1} + c_2\,(2^n - 1)$ (c_1 is an arbitrary parameter)

2.GroupNoAdj:

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ T(n-1) + T(n-2) + c_2 & \text{if } n > 1 \end{cases}$$

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Respuesta
$T(n) = c_1 F_n + c_2 L_n - c_2$ (c_1 . c_2 are arbitrary parameters)

Notación O
O ( )
O( )

3.SplitArray

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n-1) + c_2 & \text{if } n > 1 \end{cases}$$

Respuesta
$T(n) = c_1 F_n + c_2 L_n - c_2$ (c_1 . c_2 are arbitrary parameters)

Notación O
O ( )
O( )

3.SplitArray

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n-1) + c_2 & \text{if } n > 1 \end{cases}$$

Notación O
O ($c_1 2^{n-1}$)
O(2^{n-1})

Respuesta

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

$T(n) = c_1\,2^{n-1} + c_2\,(2^n - 1)$ (c_1 is an arbitrary parameter)

Code took from: http://gregorulm.com/codingbat-java-recursion-2/

4.SplitOdd10:

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n-1) + c_2 & \text{if } n > 1 \end{cases}$$

Notación O
O ($c_1\,2^{n-1}$)
O(2^{n-1})

Respuesta
$T(n) = c_1\,2^{n-1} + c_2\,(2^n - 1)$ (c_1 is an arbitrary parameter)

3.3 En el cálculo de complejidad la n representa la variable principal, es decir, la variable que cambia constantemente durante todo el algoritmo, es esta misma la que cumple la condición de parada en algún momento.
De igual manera representamos el llamado recursivo con la letra T.
Las letras C, simbolizan constantes que las podemos encontrar a lo largo de todo el algoritmo.

3.4
Aparece simplemente porque la memoria necesaria para el programa es mayor de la que configuro el entorno de ejecución en java, es decir un desbordamiento de pila; esto se produce principalmente por una mala condición de salida en el método recursivo.
3.5

Uno de los grandes problemas de la sucesión de Fibonacci, es que a partir de un valor el cálculo se demora más en realizarse, ya que, si el n va aumentando, igualmente el resultado; dicho esto, cada vez que ingresamos un n más grande el tamaño de la variable en el que guardamos el resultado debe tener el tamaño adecuado para poder guardar el número, el número más grande para calcular en Fibonacci es el 92, teniendo la variable del resultado en tipo **Long**, cuyo valor máximo es: *9.223.372.036.854.775.807.*
Fibonacci de 92 es: 7.540.113.804.746.346.429
Porque el Fibonacci de 93 ya se pasa: 12.200.160.415.121.876.738
Y no se puede ejecutar el Fibonacci de 1 millón ya que el valor del resultado no puede ser guardado en ninguna variable de tipo suficientemente largo.

Datos numéricos tomados de: https://medium.com/@williamkhepri/fibonacci-y-optimizaci%C3%B3n-f0b9590e989a

3.6

Una de las opciones es utilizar programación dinámica, para el problema de Fibonacci, cuando vamos a calcular un número, simplemente necesitamos los dos anteriores y esto ya lo habíamos calculado antes, la idea es comprobar si la solución ya existe, es decir, si ya lo habíamos hecho, si es así, lo utilizaremos para el nuevo problema.

3.7
Al principio, realizar codingbat recursión 1 es difícil, pero después de realizar varios ejercicios se hace más simple, en general y unos de los factores que más diferencian recursión 1 y recursión 2 son: en recursión 1 con un solo llamado recursivo y con un solo método en la mayoría de los casos se podían resolver fácilmente, en recursión 2 para que el problema funcione mejor, es necesario hacer varios llamados recursivos, e inclusive tener ayuda de otros métodos, así mismo en recursión 2 hay que tener en cuenta sin duda, muchísimos más detalles que son de vital importancia en las decisiones de parada y en el procedimiento, el número de parámetros aumenta considerablemente; en paralelo con recursión 1 todo lo anterior se tiene en cuenta, pero en un menor grado.

4) Simulacro de Parcial

1. *Start+1, nus, target*
2. D
3.
4. The unknown algorithm adds the numbers entered in an array
   Complexity:
   O(n)
5. *Line 1: Return 0;*
   Line2:Formas(n-1)
   Line3:Formas(n-2)

5.2 *In the worst case the algorithm executes*

T(n)=T(n-1)+T(n-2)+c

6. *Line10: return Suma(n,i+2)*
   Line12: SumaAux(n,i+1)
7. *Line9:return comb(S,i+1,t-S[i])*
   Line10:comb(S,i+1,t)
8. *Return 0*
   Return ni+nj

Explicación ejercicios en línea:

1. BunnyEars2:
Este código se basa en contar cuántas orejas tendrá un conejo recursivamente, sin embargo, si es un número par, es decir, conejitos% 2 == 0, se contarán 3 orejas, lo contrario de si es un número impar, dos será añadido. Por lo tanto, la condición de detención se realiza en el momento de no tener ningún conejo.

2. Triángulo:
Este código se basa en contar cuántos bloques tiene un triángulo, lo que nos da el número de filas, sabemos que la fila más alta tiene un solo bloque y los que le siguen lo incrementan en 1.

3. SumDigits:
Este código se basa en contar los dígitos de un número determinado, en cada llamada recursiva que nos ocupamos: sumando el número anterior tomado, y de obtener el nuevo número, este último que estamos haciendo por parejas, se logra dividiendo por 10 y para eliminar el último, usando el% mod.

4.CountHi:
Este código se basa en contar el número de veces "HI" en una cadena, pasamos por String y hacemos una subcadena de 0,2 comparando si coincide con "hi", si es así, agregamos 1 y devolvemos la cadena cortada para no devolver para Comenzar la búsqueda desde la misma parte.

5.Count7:
Este código se basa en contar la cantidad de veces que aparece "7" en una cadena, usamos la misma estructura que SumDigits, pero comparando si lo que tomamos de la Cadena es un número 7.

6.GroupsSum6:
Este código se basa en elegir un grupo que suma un int dado, elige los 6 que aparecen en el conjunto. Para eso cada vez que 6 están en la matriz, buscamos otra int que logre la suma.

7.SplitArray:
Este código se basa en dividir la matriz en dos grupos y encontrar si la suma de esos grupos es la misma. El algoritmo se detendrá cuando la suma del grupo 1 sea la misma en el grupo 2.

8.GroupNoAdj:

Este código se basa en elegir un número o un grupo de números de la matriz, con el propósito de obtener un valor (objetivo) con sus sumas, pero uno no puede seguir al otro en la matriz.

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co