Laboratorio Nro. 1: Recursión

Cristian Camilo Rendón Cardona
Universidad Eafit
Medellín, Colombia
crendo11@eafit.edu.co

Jhesid Steven Suerez Berrio
Universidad Eafit
Medellín, Colombia
jssuarezb@eafit.edu.co

2) Solución de ejercicios en CodingBat

1. A continuación se presentan los ejercicios resueltos de CodingBat y cuál es su funcionamiento
   - ***factorial():*** Este algoritmo calcula el factorial de un número recursivamente.
   - ***BunnyEars():***Va sumando de 2 en 2 hasta repetir el ciclo un número X de veces, dando como resultado el número de "orejas" que hay dado un número de "conejos".
   - ***sumDigits():***Suma cada número de un número más grande dando como resultado la suma de c/u de sus dígitos (*e.g.* 123->1+2+3=6).
   - ***powerN():***Realiza la potencia de un número base, multiplicandolo un número X de veces.
   - ***Bunnyears2():***Dado un número X suma 2 por cada número par que se encuentra hasta llegar a ese número X, asi mismo suma 3 por cada número impar, dando como resultado la suma de "orejas" y "patas" que tienen de manera visible el número "x" de conejos.
   - ***groupSum5():*** Dice si es posible sumar los números de un arreglo de tal forma que den como resultado el parámetro traget. Con la restricción de que si un número del arreglo es múltiplo de 5 se debe sumar obligatoriamente, y si después del múltiplo hay un 1, éste se debe omitir en la suma.
   - ***groupSum6():***Dice si es posible sumar los números de un arreglo de tal forma que den como resultado el parámetro traget. Con la restricción de que todo los 6's que hayan en el arreglo se deben sumar.
   - ***groupSumClump():***Dice si es posible sumar los números de un arreglo de tal forma que den como resultado el parámetro traget. Con la restricción de que si hay números repetidos adyacentes entre si se deben sumar todos u omitir todos.
   - ***splitArray():*** Dice si es posible dividir en dos un arreglo de enteros cuyas suman sean iguales.
   - ***groupNoAdj():***Dice si es posible sumar los números de un arreglo de tal forma que den como resultado el parámetro traget. Con la restricción de que si se suma un número en siguiente se debe omitir.

2. EL algoritmo *groupSum5()* evalúa si en la posición actual del arreglo hay un múltiplo de 5 y en la siguiente hay un 1, de ser así el algoritmo se llama recursivamente aumentando el parámetro start 2 unidades para omitir la siguiente posición. Si sólo hay un múltiplo de 5 pero no hay un 1 en la siguiente posición entonces se llama sumando el valor de 5. Si no ocurren ninguno de estos dos casos entonces el algoritmo se comporta como un *groupSum()*.
3. Complejidad:

   - Complejidad ***factorial()***

$$T(n) = \begin{cases} c_3 \; n \leq 1 \\ c_1 + c_2 + c_4 + T(n-1) \;\; n > 1 \end{cases}$$

Solucionando se obtiene

$$T(n) = (c_1 + c_2 + c_4)\, n + c_1 \;\therefore \\ T(n) es \; O(n)$$

- Complejidad ***bunnyEars()***

$$T(n) = \begin{cases} c_4 \; n \leq 0 \\ c_1 + c_2 + c_3 + c_5 + T(n-1) \;\; n > 0 \end{cases}$$

Solucionando se obtiene

$$T(n) = (c_1 + c_2 + c_3 + c_5)\, n + c_1 \;\therefore \\ T(n) es \; O(n)$$

- Complejidad ***sumDigits()***

$$T(n) = \begin{cases} c_3 \; n \leq 0 \\ c_1 + c_2 + c_4 + T(n/10) \;\; n > 0 \end{cases}$$

Solucionando se obtiene

$$T(n) = (c_1 + c_2 + c_3 + c_5)\, n + c_1 \;\therefore \\ T(n) es \; O(n)$$

- Complejidad ***powerN()***

Presenta la misma complejidad que *sumDigits()* y *bunnyEars()* Es $O(n)$

- Complejidad ***bunnyears2()***

$$(n) = \begin{cases} c_4 \; n \leq 0 \\ c_1 + c_2 + c_3 + c_5 + c_6 + c_7 + T(n-1) \;\; n > 0 \end{cases}$$

Solucionando se obtiene

$$T(n) = (c_1 + c_2 + c_3 + c_5 + c_6 + c_7)\, n + c_1 \;\therefore \\ T(n) es \; O(n)$$

- Complejidad ***groupNoAdj()***

$$T(n) = \begin{cases} c_3 & n \leq start \\ c_1 + c_2 + c_4 + 2T(n) & n > start \end{cases}$$

Solucionado se obtiene

$$T(n) = 2^{n-1} + (2^n - 1)(c_1 + c_2 + c_4) \;\therefore \\ T(n) \;\; es \;\; O(2^n)$$

- Complejidad ***groupSum5()***

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

$$T(n) = \begin{cases} c_3 & n \leq start \\ c_1 + c_2 + c_4 + c_5 + c_6 + 2T(n) & n > start \end{cases}$$

Solucionando se obtiene

$$T(n) = 2^{n-1} + (2^n - 1)(c_1 + c_2 + c_4 + c_5 + c_6) \therefore$$
$$T(n)\ es\ O(2^n)$$

- Complejidad ***groupSum6()***

$$T(n) = \begin{cases} c_3 & n \leq start \\ c_1 + c_2 + c_4 + c_5 + 2T(n) & n > start \end{cases}$$

Solucionando se obtiene

$$T(n) = 2^{n-1} + (2^n - 1)(c_1 + c_2 + c_4 + c_5) \therefore$$
$$T(n)\ es\ O(2^n)$$

- Complejidad ***groupSumClump()***

$$T(n) = \begin{cases} c_3\ n \leq start \\ c_1 + c_2 + c_4 + c_5 + (c_6 + c_7 + c_8) * n + c_9 + 2T(n-1)\ n > start \end{cases}$$

Solucionando se obtiene

$$T(n) = c_1 2^{n-1} + c_1(2^n - 1) + c_2(2^n - 1) + c_4(2^n - 1) + c_5(2^n - 1) + c_6(-n + 2^{n+1} - 2) + c_7(-n + 2^{n+1} - 2) + c_8(-n + 2^{n+1} - 2) + c_9(2^n - 1) \therefore$$

$$T(n)\ es\ O(2^{n+1})$$

- Complejidad ***splitArray()***

$$T(n) = \begin{cases} c_3 & n \leq start \\ c_1 + c_2 + c_4 + 2T(n) & n > start \end{cases}$$

Solucionado se obtiene

$$T(n) = 2^{n-1} + (2^n - 1)(c_1 + c_2 + c_4) \therefore$$
$$T(n)\ es\ O(2^n)$$

3) Simulacro de preguntas de sustentación de Proyectos

1. El stack over flow y el heap son los espacios de memoria que permiten grandes llamdos recursivos y arreglos congran cantidad de elementos respectivamente
2. Hay dos motivos por los cuales no se puede calcular el fibonacci de un millón, el primero sería por el tiempo de ejecución y el segundo porque el número que resulta no puede ser almacenado en una variable.
3. Otra forma de calcular el n-esimo termino de finobacci es con la ecuación del número de oro:

$$x_n = \frac{\varphi^n - (1 - \varphi)^n}{\sqrt{5}}$$

Pero se tendría el problema de que si no se usan suficientes decimales para φ el resultado no sería un número entero.
Otra forma es generar un arreglo que vaya almacenando el número anterior y así volver el algoritmo O(n).

```
FUNCTION fibonacci(ARRAY[n+1] terminos, INTEGER n)

     IF(n<2)

          RETURN n

     ELSE

          terminos[0]=0

          terminos[1]=1

          FOR(INTEGER i IN RANGE [2,n])

                terminos[i] = terminos[i-1] + terminos[i-2]

          RETURN T[n]
```

Gerson Lázaro (2015), Programando y reprogramando la sucesión de Fibonacci. https://gersonlazaro.com/programando-y-reprogramando-fibonacci/

4. Los algoritmos de recursión 1 son mucho más rápidos que los de recursión 2, debido a que los primero son O(n) y los otros son O(2^n) aumentando exponencialmente su tiempo de ejecución

4) Simulacro de Parcial

1. *start + 1, nums, target*
2. *a*
3. **3.1.** n-a, a, b, c
   3.2. res, solucionar(n-b, a, b, c)
   3.3. res, solucionar(n-c, a, b, c)
4. *e*
5. **5.1.** Linea 1**:** n

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

Linea 3: n-1
Linea 4: n-2

5.2. b.

6.

6.1. 0
6.2. sumaAux(n, i+ 1)

7.

7.1. comb(S, i + 1, t – S[i])
7.**2.** comb(S, i + 1, t)

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