Laboratorio Nro. 1: Recursión

Mauricio Castaño Uribe
Universidad Eafit
Medellín, Colombia
mcastanou@eafit.edu.co

Jose Miguel Gil Valencia
Universidad Eafit
Medellín, Colombia
jmgilv@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. **EXPLICACION GROUPSUM5**

   En este ejercicio el cual consiste en elegir un grupo de números el cual sumados entre ellos de un numero objetivo dado, siempre y cuando se escojan todos los números que son múltiplos de 5, y si aparece un número "1" después de un "5", este se debe de escoger obligatoriamente.

   Entonces para el algoritmo se implementó como condición de parada cuando se ha recorrido todo el arreglo; y se pregunta si el numero obtenido y el numero objetivo es el mismo, esto se hace ya que cada vez que se escoge un número, este número se le reste al número objetivo, haciendo que el numero objetivo llegue a cero si un grupo de números del arreglo logra sumar el objetivo.
   Después de la condición de parada se tienen los parámetros recursivos. El primero se trata de elegir obligatoriamente los múltiplos de "5" luego si se escogió un número "5", se revisa si el siguiente es un número "1" para agregarlo. Y el parámetro final de la recursión es escoger y no escoger el número que sigue, haciendo tosas las posibles combinaciones que cumplan con los parámetros dados

2. **2.1**

COMPLEJIDAD bunnyEars:

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

T (n) =c_2*n + c_1
O(n)
Donde "n" es el número de conejos

COMPLEJIDAD sumDigits:

$$T(n) \begin{cases} c_1 \text{ if } 0 < x < 10 \\ T(n/10) + c_2 \text{ if } x >= 10 \end{cases}$$

T(n)= c_2*Log(n)/Log (10) + c_1
O (Log n)
Donde "n" es el entero que se ingresó como parámetro

COMPLEJIDAD powerN:

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

T(n)=c_2*n+c_1
O(n)
Donde "n" es la potencia a la que se elevó la base

COMPLEJIDAD array6:

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

T(n)=c_2*n+c_1
O(n)
Donde "n" es el número de posiciones en el arreglo que falta por revisar

COMPLEJIDAD array11:

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

T(n)=c_2*n+c_1
O(n)
Donde "n" es el número de posiciones en el arreglo que faltan por revisar

2.2

COMPLEJIDAD groupSum6:

$$T(n) \begin{cases} c_1 \text{ if } x = 0 \\ 2 * T(n-1) + c_2 \text{ if } x >= 1 \end{cases}$$

T(n)=c_1*2^(n-1)+c_2*(2^n -1)
O (2^n)
Donde "n" es el número de posiciones que aún no se han revisado del arreglo

COMPLEJIDAD groupSum5:

$$T(n) \begin{cases} c_1 \text{ if } x = 0 \\ 2 * T(n-1) + c_2 \text{ if } x >= 1 \end{cases}$$

T(n)=c_1*2^(n-1)+c_2*(2^n -1)
O (2^n)
Donde "n" son las posiciones en el arreglo que aún no se han revisado del arreglo

COMPLEJIDAD splitArray:

$$T(n) \begin{cases} c_1 \text{ if } x = 0 \\ 2 * T(n-1) + c_2 \text{ if } x >= 1 \end{cases}$$

T(n)=c_1*2^(n-1)+c_2*(2^n -1)
O (2^n)
Donde "n" es el número de posesiones que faltan por ser revisadas en el arreglo

COMPLEJIDAD splitOdd10:

$$T(n) \begin{cases} c_1 \text{ if } x = 0 \\ 2 * T(n-1) + c_2 \text{ if } x >= 1 \end{cases}$$ T(n)=c_1*2^(n-1)+c_2*(2^n -1)

O (2^n)
Donde "n" es el número de posesiones que faltan por ser revisadas en el arreglo

COMPLEJIDAD split53:

$$T(n) \begin{cases} c_1 \text{ if } x = 0 \\ 2 * T(n-1) + c_2 \text{ if } x >= 1 \end{cases}$$

T(n)=c_1*2^(n-1)+c_2*(2^n -1)
O (2^n)
Donde "n" es el número de posesiones que faltan por ser revisadas en el arreglo

3. **Stack Overflow**

   Esta definición aplicada en la recursión se trata de ir haciendo llamados recursivos y se guardan en cierto orden, a medida que se llama uno nuevo, esta acción se pone en lista encima de las anteriores. Haciendo una pila de llamados. Y luego al terminar los llamados recursivos, se empieza a devolver en orden, solucionando el último llamado y terminando en el primero. Es decir, stack overflow es ordenar de primero a último y resolver de último a primero.

4. **Limite Fibonacci**

   Al calcular un número muy grande los llamados recursivos ocupan demasiado espacio y el número del resultado da tan grande que la memoria asignada no da para que el número se guarde por lo que el problema es la memoria

5. **Comparar complejidad**

   Podemos concluir que todos los algoritmos realizados en la sección de recursión 2 tiene un grado de complejidad considerablemente mayor a los algoritmos de recursión 1, ya que en los algoritmos de recursión 2 estos revisan muchos más casos posibles para llegar a la solución del problema mientras que los de recursión 1 llegan de una forma más directa y sencilla a la respuesta

4) Simulacro de Parcial

1. (start+ , nums , target)

2. a) T(n)= T(n/2) + c

3. linea 4: n-a, a, b, c
linea 5: res, solucionar(n-b, a, b, c)
linea 6: res, solucionar(n-c, a, b, c)

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

6. sumaAux(n, i+2)
sumaAux(n, i+1)

7. Línea 9: (S, i+1, t-s[i])
Línea 10: (S, i+1, t)

8. Linea 9: return 0;
Linea 13: ni+nj;

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