Laboratorio Nro. 1: Recursión

Juliana Henao Arroyave
Universidad Eafit
Medellín, Colombia
jhenaoa4@eafit.edu.co

Gerónimo Zuluaga Londoño
Universidad Eafit
Medellín, Colombia
gzuluagal@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. El método recibe tres parámetros, uno es un contador que sirve para establecer una posición en el arreglo, el siguiente es el arreglo y lo otro al numero que se quiere llegar sumando. La condición de parada es cuando ya se haya recorrido todo el arreglo, si cuando pase esto el target es 0 es porque encontró un conjunto de números que sumaran exactamente el target y retorna true, si no es cero es porque los números no alcanzaron a ser el target o se pasaban. El ejercicio pedía que si un numero era múltiplo de 5 obligatoriamente había que usarlo, excepto en el caso en que después de este número hubiera un 1. Por esto primero se pregunta si el número en la posición que estamos no deja residuo al dividirlo por 5 y si eso es cierto se evalúa si en la siguiente posición hay un uno (especificando que si se encuentra en la última no hay que evaluar esto), si lo anterior es verdadero no se toman encuentra ninguno de estos valores saltando ya de a dos en el contador. Si no se cumple lo del 1sabremos que se debe restar el múltiplo de 5 de inmediato del target. Si el número en la posición indicada no es múltiplo de 5, se retornan dos llamados del propio método; en cada uno se le suma uno al start. Y al target, según si se cumpla o no que sumen lo deseado, de le resta el número del arreglo que esté en la posición de start.
2. **Recursion 1**
   Triangle:
   T(n) = t(n-1) + c1
   Ecuación recursiva :
   T(n) = c1 + c1n

   SumDigits:
   T(n) = t(n/10)+c1
   Ecuación recursiva :
   T(n) = [(c1)log(n)]/log(10)

   Count7:
   T(n) = t(n/10) + c1
   Ecuación recursiva :
   T(n) = [(c1)log(n)]/log(10)

   CountX:
   t(n) = C1 +t(n-1)
   Ecuación recursiva :

T(n) = c1 + c1n

Count8 :
T(n) = t(n/10) + c1
Ecuación recursiva :
T(n) = [(c1)log(n)]/log(10)

Recursión 2
Groupsum
t(n) = t(n-1) + t(n-1)
Ecuación recursiva
t(n)= 2^n-1

groupsum6
t(n) = t(n-1) + t(n-1**)**
Ecuación recursiva
t(n)= 2^n-1

groupNoAdj
t(n) = t(n-1) + t(n-2)
Ecuación recursiva
c1*F1 + c2*Ln

SplitArray
t(n) = t(n-1) + t(n-1)
Ecuación recursiva
t(n)= 2^n-1

groupsum5
t(n) = t(n-1) + t(n-1)
Ecuación recursiva
t(n)= 2^n-1

3. Para el cálculo de la complejidad de los algoritmos se tienen en cuenta varios aspectos que nos arroja una función.
En un algoritmo, la variable que tomamos como n es la cantidad de operaciones que se realiza en el algoritmo, pero hay operaciones que requieren mucho más esfuerzo computacional que otro, por ejemplo:
Un llamado recursivo requiere mucho más esfuerzo que una suma, así que se toma como n al llamado recursivo mas no a la suma, porque al esto llevarse a números grandes de operaciones la suma afectaría muy poco a la eficiencia del algoritmo
También se tienen en cuenta las constantes que las denotamos con c1, c2… etc. Estas constantes son el valor con el que se va a trabajar el algoritmo, por ejemplo:
Puede ser 1, 1000 o 10000000 o hasta un número más grande.
Estas constantes en la notación O también se omiten porque tampoco en dimensiones muy grandes no afectaría la complejidad del algoritmo

4. El *Stack Overflow* es un error que sucede porque el algoritmo no llega a un punto donde debe parar de hacer operaciones y la pila de recursiones y operaciones, como su nombre lo dice, rebasa el máximo de su capacidad. Esto ocurre cuando se hace un mal llamado recursivo que no conduce a una condición de

parada, o por ejemplo no hace ningún cambio en los parámetros, así que se llama a si mismo infinitamente. También cuando la condición de parada no es la adecuada y por tanto nunca llega a ese punto.

5. El valor mas grande que pudo calcular fue 45, ya que si se ponían números un poco más grandes que eso podía tardar varios minutos y luego horas. Esto sucede porque el método tiene dos llamados recursivos lo que hace a la complejidad de el algoritmo de orden exponencial así que el tiempo que se demora en realizar las cosas crece exponencialmente por todas las operaciones que debe hacer.
6. Opcional
7. Los problemas de recursión 2 son mucho más complejos que los de recursión 1, ya que estos primeros tenían mas de un llamado recursivo, lo que hace la complejidad de orden exponencial. A diferencia de los de recursión 1 que no eran tan complejos, siendo casi todos de orden lineal o logarítmico. Los de recursión 1 pueden tomar valores muy grandes y siguen funcionando casi igual de rápido, en cambio los de recursión 2 llegaba un punto donde el computador se queda procesando mucho tiempo y ya es como si el algoritmo dejara de servir por todo el tiempo que toma resolviendo un problema on un numero muy grande.

4) Simulacro de Parcial

1. *Start+1, nums, target*
2. *c*
3. **3.1**. n-1,a,b,c
   3.2. solucionar(n-a,a,b,c),solucionar(n-b,a,b,c)
   3.3. n-res,solucionar(n-c,a,b,c)
4. e
5. Linea 2. *Return n;*
   Linea 3. *n-1*
   Linea 4. *n-1*
   5.2 *c*
6. Linea 10. *sumaAux(substring(n,i+3),i+2);*
   Linea12. *sumaAux(substring(n,i+1),i+1);*
7. Opcional
8. Linea 9. *Return -1;*
   Linea 13. *ni +nj;*

5) Lectura recomendada (opcional)

a) Título
b) Ideas principales
c) Mapa de Conceptos

6) Trabajo en Equipo y Progreso Gradual (Opcional)

a) Actas de reunión
b) El reporte de cambios en el código
c) El reporte de cambios del informe de laboratorio

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