Adrián Borges Cano
Marco González Martínez
Laboratorio: Miércoles 15:00-17:00

Algoritmia

ÍNDICE

Tabla de contenido

PROBLEMA 3

Analizar la eficiencia del siguiente código:

```
fun Calculo(x,y,z: entero) dev valor:entero
var i,j,t: entero
  valor ← 0
  Desde i ← x hasta y Hacer valor ← valor + i fdesde
  si (valor ÷ (x+y)) <= 1 entonces Devolver z
  si no
      t ← x + ((y-x) ÷ 2)   { ÷ es la división entera }
      Desde i ← x hasta y Hacer
            Desde j ← (3*x) hasta (3*y) Hacer
                  valor ← valor + Minimo(i,j)
            fdesde
      fdesde
      valor ← valor + 4*Calculo(t,y,valor)
      Devolver valor
  fsi
ffun
```

Coste del algoritmo:

```
1: 1 Declaración (1 ejecución)
2: 3 Declaraciones (1 ejecución)
3: 1 Asignación (1 ejecución)
4: 1 Bucle for + 1 Asignación (1 ejecución)
   CUERPO DEL BUCLE: 1 Asignación + 1 Suma (n ejecuciones)
5: 1 Condicional + 1 SUMA + 1 División Entera + 1 Comparación (1 ejecución)
   CUERPO DEL CONDICIONAL-T: 1 Devolución
   CUERPO DEL CONDICIONAL-F:
   7: 1 Resta + 1 División Entera + 1 Suma + 1 Asignación(1 ejecución)
   8: 1 Bucle for + 1 Asignación (1 ejecución)
         CUERPO DEL BUCLE: (n ejecuciones)
         9: 1 Bucle for + 2 Mult + 1 Asignación (n ejecuciones)
            CUERPO DEL BUCLE: (3n^2 ejecuciones)
            10: 1 Función + 1 Suma + 1 Asignación (3n^2 ejecuciones)
   13: 1 Función Recursiva + 1 Suma + 1 Asignación (1 ejecución)
   14: 1 Devolución (1 ejecución)

Coste = 1+3+1+(1+2n)+(3+max(1,4+(1+3n+(2+Coste(Minimo))n(3n))+T(Calculo)+2+1))
Coste = 17 + 5n + 3n^2 · (2+Coste(Minimo)) + T(n/2)
Coste = 17 + 5n + 12n^2 + T(n/2)
```

Para calcular el coste de la llamada recursiva a la función, es necesario analizar que parámetros toman juego en las iteraciones en los bucles de la función. En este caso, la variable *z* no aparece, en ninguna condición de los bucles. El número de iteraciones en este caso depende de la diferencia entre los parámetros (y-x), que llamaremos n por comodidad. Al llamar recursivamente a la función, se introduce como parámetro *x* la variable *t*, a la que se le ha asignado *x + (y-x)/2* así que el valor de *t* pasará a ser la media entre *x* e *y*. De esto se puede deducir que *(y-t) = (x-t)/2*. Y la *n* de la que depende la cantidad de iteraciones pasará a ser n/2 al aumentar el nivel de la recursión.

```
def minimo(x,y):
   if x>=y:       # 1 Comparación
      return y    # 1 Devolución
   else:
      return x    # 1 Devolución

# Coste = 1 + max(1,1)
# Coste = 2
```

Cálculo de la recursividad del coste final:

$$T(n) = T(n/2) + 17 + 5n + 12n^2$$

$\Rightarrow$ Realizaremos un cambio de variable, para facilitar el cálculo en el caso de $n/2$.

$$\boxed{n = 2^m}$$

$$T(2^m) = T(2^{m-1}) + 17 + 5 \cdot 2^m + 12 \cdot 2^{2m}$$

$$x^m = x^{m-1} + 17 + 5 \cdot 2^m + 12 \cdot 2^{2m} \Rightarrow x^m - x^{m-1} = 17 + 5 \cdot 2^m + 12 \cdot 2^{2m}$$

• Parte Homogénea:

$$x^m - x^{m-1} = 0 \Rightarrow x^{m-1}(x-1) = 0 \quad \begin{cases} x = 1 \\ \cancel{x = 0} \end{cases}$$

se descarta porque sería una raíz nula.

• Parte No Homogénea:

$$12 \cdot 2^{2m} + 5 \cdot 2^m + 17$$

$$(17 \cdot 1^m) m^0$$

$$(12 \cdot 4^m + 5 \cdot 2^m) m^0 = 12 \cdot 4^m \cdot m^0 + 5 \cdot 2^m \cdot m^0$$

$$\underbrace{17 \cdot 1^m \cdot m^0 + 12 \cdot 4^m \cdot m^0 + 5 \cdot 2^m \cdot m^0}_{(x-1) \quad (x-4) \quad (x-2)}$$

$\Rightarrow$ Combinación de la parte Homogénea con la No Homogénea:

$$(x-1)^2, (x-4), (x-2)$$

$$17 + C_1 m + 12 \cdot 4^m + 5 \cdot 2^m \qquad n = 2^m \Rightarrow m = \log_2 n$$

$$\boxed{T(n) = 12n^2 + 5n + C_1 \log_2 n + 17} \Rightarrow \boxed{O(n) = n^2}$$

(coste final)

PROBLEMA 7

Realiza un programa que pida un número positivo al usuario (N) y le diga cuantos primos hay entre 1 y ese número N, y cuantos perfectos hay entre 1 y ese número N. Realiza un análisis de eficiencia y de complejidad.

Código del algoritmo:

```
def divisores(n):
    if n==1: return [1]
    i=1
    k=n//i
    divisores = []
    while i<k:
        if n%i==0:
            divisores.append(i)
            if i!=n/i:
                divisores.append(n//i)
        k=n//i
        i+=1
    return divisores

def esprimo(n):
    ldivisores=divisores(n)
    return len(ldivisores)==2 and ldivisores[0]!=ldivisores[1]

def esperf(n):
    return sum(divisores(n))==2*n

n = int(input('Introduce un número positivo: '))

tot_primos=0
tot_perfectos=0

for j in range(1,n):
    tot_primos += esprimo(j)
    tot_perfectos += esperf(j)

print(f"Hay {tot_primos} números primos entre {1} y {n}")
print(f"Hay {tot_perfectos} números perfectos entre {1} y {n}")
```

Coste del algoritmo:

Asumiendo Coste(append) = 1, Coste(len) = 1, Coste(sum) = n

divisores:
6: 1 Condicional + 1 Comparación (1 ejecución)
CUERPO DEL CONDICIONAL-T: 1 Devolución (1 ejecución)
CUERPO DEL CONDICIONAL-F: (1 ejecución)
7: 1 Asignación (1 ejecución)
8: 1 División entera (1 ejecución)
9: 1 Asignación (1 ejecución)
10: Bucle-while (1 ejecución)
CUERPO DEL BUCLE: 1 Comparación ($\sqrt{(n)}$ ejecuciones)
11: 1 Condicional + 1 Módulo + 1 Comparación ($\sqrt{(n)}$ ejecuciones)
CUERPO DEL CONDICIONAL-T: ($\sqrt{(n)}$ ejecuciones)
12: 1 Coste(append) ($\sqrt{(n)}$ ejecuciones)
13: 1 Condicional + 1 División + 1 Comparación ($\sqrt{(n)}$ ejecuciones)
CUERPO DEL CONDICIONAL-T: ($\sqrt{(n)}$ ejecuciones)
14: 1 Coste(append)+ 1 División Entera ($\sqrt{(n)}$ ejecuciones)
15: 1 Asignación + 1 División Entera ($\sqrt{(n)}$ ejecuciones)
16: 1 Asignación + 1 Suma ($\sqrt{(n)}$ ejecuciones)
17: 1 DEVOLUCIÓN (1 ejecución)

Coste = 2 + max(1, (1+1+1+1+ $\sqrt{(n)}$·(1+3+1+3+2+2+2)+1))
Coste = 7 + 14· $\sqrt{(n)}$

esprimo:
24: 1 Asignación + Coste(divisores) (1 ejecución)
25: 1 Devolución + Coste(len)+ 2 Comparaciones (1 ejecución)

Coste = 1 + 7 + 14 $\sqrt{(n)}$ + 1 + 1 + 2
Coste = 11 + 14 $\sqrt{(n)}$ + 1

esperf:
32: 1 Devolución + Coste(divisores) + Coste(sum) + 1 Comparación + 1 Mult (1 ejecución)

Coste = 3 + n + 7 + 14 $\sqrt{(n)}$
Coste = 10 + 14$\sqrt{(n)}$ + n

34: 1 Asignación (1 ejecución)

36: 1 Asignación (1 ejecución)
37: 1 Asignación (1 ejecución)
39: 1 Bucle for (n ejecuciones)

CUERPO DEL BUCLE (n ejecuciones)
  40: 1 Asignación + 1 Suma + T(esprimo) (n ejecuciones)
  41: 1 Asignación + 1 Suma + T(esperf) (n ejecuciones)

Coste = 1 + 1 + 1 + n·(2 + 11 + 14log2(n) + n + 2 + 10 + 14$\sqrt{(n)}$ + n)
Coste = 3 + 25n + 14n$\sqrt{(n)}$ +2n^2 => O(n) = n^2

En cada iteración del bucle de la función divisores, a la variable k se le asigna el valor de n//i, y posteriormente i es incrementada. De esta forma, la condición del bucle (i<k) se romperá cuando el valor de i>=k, que como k=n/i el mínimo valor que lo puede llegar a romper es $\sqrt{(n)}$, por lo que el número de veces que se ejecuta el bucle es $\sqrt{(n)}$.

PROBLEMA 9

Realizar una función recursiva que calcule el siguiente sumatorio: S= 1+2+3+4+….+n-1+n. Realiza un análisis de eficiencia y de complejidad.

Código del algoritmo:

```
def sumatorio (n):
    """
    int --> int
    OBJ: calcular el sumatorio de 1 hasta n
    PRE: n>=1
    """
    # Caso de partida: El sumatorio de 1 hasta 1, trivialmente es 1
    if n==1:                    # 1 Comparación
        return 1                # 1 Devolución
    # Caso general: El sumatorio de 1 hasta n es equivalente a la suma de n más
el sumatorio de 1 hasta n - 1
    # Se puede desglosar según la progresión de la suma de los n primeros de la
siguiente forma:
    # 1 + 2 + ... + n-1 + n
    else:
        return n+(sumatorio(n-1))   # 2 + T(n-1)

# T(n) = 3 + T(n-1)
```

<u>Cálculo de la recursividad del coste final:</u>

$T(n) = 3 + T(n-1)$

$x^n = 3 + x^{n-1} \Rightarrow x^n - x^{n-1} = 3$

- Parte Homogénea:

$x^n - x^{n-1} = 0 \Rightarrow x^{n-1}(x-1) = 0$ — $x = 1$; $\cancel{x = 0}$ Descartada

- Parte No Homogénea:

$3 = P(n) \cdot a^n \Rightarrow 3 = (3 \cdot n^0) 1^n$ $\quad 0+1$

$3 = (x-1)^1$

$x - 1 = 0 \Rightarrow \boxed{x = 1}$

$\Rightarrow$ Combinación de la Homogénea con la No Homogénea:

$(x-1)(x-1) = (x-1)^2$

$1^n \cdot C_1 \cdot n^0 + 1^n \cdot C_2 \cdot n \Rightarrow \boxed{T(n) = C_1 + C_2 \cdot n} \rightarrow \boxed{O(n) = n}$

$\boxed{C_1, C_2 \in \mathbb{R}}$

→ Coste final:

$\boxed{T(n) = 3 + C_1 + C_2 \cdot n}$ $\boxed{O(n) = n}$