Diseño de algoritmos

Jesús Bermúdez de Andrés. UPV-EHU
Ejercicios: Análisis de algoritmos

Curso 2008-09

1. Con un algoritmo de función de coste temporal $f(n) = n^3$ resolvemos problemas de tamaño K en una hora. ¿Hasta qué tamaño podremos resolver, en el mismo tiempo, con una máquina 1000 veces más rápida? ¿Y si la función de coste fuese $f(n) = 2^n$?

2. Disponemos de dos algoritmos A y B para resolver el mismo problema, con implementaciones que realizan $8n^2$ y $64n \lg n$ operaciones elementales, para entradas de tamaño n, respectivamente. Determine para qué tamaños de la entrada, el algoritmo A es más rápido que B.

3. Determine para qué tamaños de entrada, en la misma máquina, es más rápido un algoritmo con función de coste $100n^2$ que otro con función de coste 2^n.

4. Analice el siguiente algoritmo, definiendo la función de coste pertinente y especificando claramente qué es lo que se está considerando como tamaño de la entrada.

```
func Es_Equilibrado (V, inicio, fin) return integer
  for i in inicio .. fin loop
    izq ← 0;
    for k in inicio .. i − 1 loop izq ← izq+V(k); end loop;
    der ← 0;
    for k in i .. fin loop der ← der+V(k); end loop;
    if izq = der then return i;
  end loop;
  return 0;
```

5. El siguiente algoritmo busca la primera aparición de un string $B(1..k)$ en el string $A(1..n)$; devuelve *true* y el índice de A donde comienza B, si lo encuentra; y *false* en caso contrario. El valor $n - k + 1$ es la posición más a la derecha en A donde podría comenzar B.

```
func StringSearch (A, B: String) return (boolean, natural)
  N ← A.length; K ← B.length; Inicio ← A.firstIndex;
```

```
Encontrado ← false;
Limite ← N-K+1;
while not Encontrado and Inicio ≤ Limite loop
  I ← Inicio;
  J ← B.firstIndex;
  while J ≠ K+1 and then (A(i) = B(j)) loop
    I ← I+1; J ← J+1;
  end loop ;
  Encontrado ← (J=K+1);
  if not Encontrado then Inicio ← Inicio+1;
end loop ;
return (Encontrado, Inicio)
```

¿Cuántas veces se ejecuta la comparación $A(i) = B(j)$ en el peor caso? ¿Qué entradas dan lugar al peor caso? Obsérvese que el operador booleano *and then* hace que la comprobación sólo se realice cuando sea verdadera la condición J $\neq$ K+1.

6. El siguiente algoritmo realiza una búsqueda secuencial de un número X en una tabla *T(I..F)*. Determine cuántas comparaciones, del número X con un elemento de la tabla T, se realizan en el caso peor y en el caso medio.

```
func BUSQUEDA_SECUENCIAL (T: Tabla; I, F: integer; X: integer)
    return boolean
  Actual ← I
  while Actual ≤ F and then T(Actual) ≠ X loop
    Actual ← Actual+1
  end loop
  if Actual > F then return false
  else return true
```

7. Un número natural $n \geq 1$ es *triangular* si es la suma de una sucesión ascendente no nula de naturales consecutivos que comienza en 1. Por tanto, los cinco primeros números triangulares son 1, $3 = 1+2$, $6 = 1+2+3$, $10 = 1+2+3+4$ y $15 = 1+2+3+4+5$.

   a) Escriba un algoritmo que, dado un entero positivo $n \geq 1$, decida si éste es un número triangular.

   b) Analice su algoritmo.

8. Calcule la función de coste temporal de los siguientes algoritmos:

   a)
   ```
   func SERIE (n, m: natural) return natural
     if n ≤ 1 then return m
     else return m + SERIE(n − 1, 2 × m)
   ```

b) **func** TOTAL (n: natural) **return** natural
  if $n \leq 1$ **then return** 1
  else return TOTAL$(n-1) + 2 \times$ PARCIAL$(n-1)$

siendo

func PARCIAL (k: natural) **return** natural
  if $k \leq 1$ **then return** 1
  else return $2 \times$ PARCIAL$(k-1)$

9. Dado el algoritmo siguiente, que determina si una cadena C es palíndromo:

func PAL (C, i, j) **return** booleano
  if $i \geq j$ **then return** verdadero
  elsif $C(i) \neq C(j)$ **then return** falso
  else return PAL$(C, i+1, j-1)$

Analice la evaluación de PAL$(C, 1, n)$ en el caso peor y en el caso medio, suponiendo equiprobabilidad de todas las entradas y siendo $\{a, b\}$ el alfabeto que forma las cadenas.

10. Imagine un robot situado sobre unos raíles sin fin, que tiene la posibilidad de moverse un paso a la derecha o a la izquierda al ejecutarse la operación *posición* $\leftarrow$ *posición* $+1$. La dirección del movimiento depende del valor del parámetro *sentido*. La operación *cambiar(sentido)* modifica el valor de *sentido* de *derecha* a *izquierda* y viceversa; y esos son los únicos valores del parámetro *sentido*. Además, el robot tiene la posibilidad de advertir la presencia de un determinado objeto (frente a la posición determinada por el parámetro *posición*) al ejecutarse la operación booleana *detecto_en(posición)*. El siguiente algoritmo mueve pendularmente al robot en busca del objeto citado, partiendo de un origen que marcamos con el valor 0.

límite $\leftarrow$ 1
sentido $\leftarrow$ *derecha*
loop
  posición $\leftarrow$ 0
  while *posición* $<$ *límite* **loop**
    posición $\leftarrow$ *posición* $+1$
    if *detecto_en(posición)* **then return** *(posición, sentido)*
  end loop
  sentido $\leftarrow$ *cambiar(sentido)*
  for i **in** 1..*límite* **loop**
    posición $\leftarrow$ *posición* $+1$
  end loop
  límite $\leftarrow 2 \times$ *límite*
end loop

Asumiendo que el objeto será encontrado, analice el número de veces que se realizará la operación *posición* $\leftarrow$ *posición* $+1$ en función de la distancia inicial del objeto al origen.

11. Cuando los números tienen muchos dígitos, hay que cuestionarse si las operaciones aritméticas pueden considerarse operaciones elementales.

    a) Consideremos los algoritmos de suma y multiplicación, realizados con lápiz y papel, aprendidos en la escuela. Denominamos *elemental* a la multiplicación, respectivamente suma, de dos dígitos decimales. Justifique que el algoritmo típico que usa para multiplicar, con lápiz y papel, dos números enteros A y B realiza $\Theta(mn)$ multiplicaciones elementales y $\Theta(mn)$ sumas elementales, siendo m y n el número de dígitos de A y B respectivamente.

    b) Las operaciones $A \times 10$, $\lfloor B/10 \rfloor$ y $B\ mod\ 10$ pueden considerarse de $O(1)$ (consisten, respectivamente, en añadir, eliminar y seleccionar una cifra decimal; por lo tanto, para realizarlas no necesitamos multiplicaciones ni sumas).

    Analice el número de multiplicaciones elementales, y separadamente el número de sumas elementales, que se realizan con el siguiente algoritmo de multiplicación. Los símbolos $\oplus$ y $\otimes$ representan, respectivamente, las operaciones de suma y multiplicación de números naturales realizadas con el algoritmo típico considerado en el apartado anterior. Fíjese que las operaciones $A \times 10$, $\otimes$ y MULTIPLICAR se realizan con algoritmos diferentes.

    func MULTIPLICAR (A,B: natural) **return** natural
      if B=0 **then return** 0
      else return A$\otimes$($B\ mod$ 10) $\oplus$ MULTIPLICAR($A \times 10$, $\lfloor B/10 \rfloor$)