Departamento de Estadística, Matemáticas e Informática
Grado en Ingeniería Informática en Tecnologías de la Información

Metodología de la Programación y Algoritmia

Convocatoria de Junio 2017
SOLUCIÓN

1.- Describe un problema y propón un algoritmo que utilice una estrategia voraz para resolverlo. Explica detalladamente el funcionamiento del algoritmo, justifica por qué es voraz y haz una traza para un caso particular de forma que se vea claro el funcionamiento.

SOLUCIÓN:

En el tema de los algoritmos voraces hay diversos algoritmos que siguen esta estrategia: el problema de la devolución del cambio, la mochila con objetos fraccionables, planificación de tareas,… El problema y algoritmo de resolución que se utilice puede ser alguno de los que hayamos visto o cualquier otro, siempre y cuando estén bien definidos y se justifique correctamente por qué el algoritmo es voraz.

2.- Calcula la complejidad asintótica del siguiente algoritmo y **justifica** tu respuesta de forma que quede claro por qué sale dicha complejidad. La complejidad asintótica de calcular(n:real, m:entero):real es O(m^2).

```
          función junio17(A:real[x,y]):real
             i,j: entero
             v: real
(1)          v ← 0
(2)          i ← x
(3)          mientras i ≥ 1 hacer
(4)             j ← 1
(5)             mientras j < y hacer
(6)                v ← v + calcular(A_{i,j},y)
(7)                j ← j + 2
(8)             fmientras
(9)             i ← i / 2
(10)         fmientras
(11)         devolver v
          ffunción
```

SOLUCIÓN:

Las variables que indican el tamaño del problema son x e y. La complejidad vendrá dada en función de estas variables.

```
          función junio17(A:real[x,y]):real     Nivel 1    Nivel 2    Nivel 2
             i,j: entero
             v: real
(1)          v ← 0                              O(1)
(2)          i ← x                              O(1)
(3)          mientras i ≥ 1 hacer               O(logx)
(4)             j ← 1                                      O(1)
(5)             mientras j < y hacer                       O(y)
(6)                v ← v + calcular(A_{i,j},y)                        O(y^2)
(7)                j ← j + 2                                          O(1)
(8)             fmientras
(9)             i ← i / 2                                  O(1)
(10)         fmientras
(11)         devolver v                                    O(1)
          ffunción
```

En el bucle de la línea (3) el número de iteraciones varía en función del logx y en el de la línea (5) lo hace de forma lineal con la variable y. Como las instrucciones de la condición son del O(1), la complejidad de estas líneas son O(logx) y O(y), respectivamente

Para resolver las líneas (5) – (8), aplicamos la regla de la secuencia de instrucciones a las instrucciones que están dentro del bucle max{ O(y^2), O(1)} = O(y^2) y resolvemos la complejidad de este bucle aplicando la regla del producto O(y)*O(y^2) = O(y^3).

Metodología de la Programación y Algoritmia

Convocatoria de Junio 2017
SOLUCIÓN

```
       función junio17(A:real[x,y]):real    Nivel 1    Nivel 2
          i,j: entero
          v: real
(1)       v ← 0                             O(1)
(2)       i ← x                             O(1)
(3)       mientras i ≥ 1 hacer              O(logx)
(4)          j ← 1                                     O(1)
(5)          mientras j < y hacer                      O(y^3)
(6)             v ← v + calcular(A_i,j,y)
(7)             j ← j + 2
(8)          fmientras
(9)          i ← i / 2                                 O(1)
(10)      fmientras
(11)      devolver v                                   O(1)
       ffunción
```

Volvemos a realizar lo mismo con el bucle de la línea (3). Para las líneas (4) – (9) la regla de la secuencia de instrucciones: max{ O(1), O(y^3), O(1)} = O(y^3) y la regla del producto para el bucle O(logx)*O(y^3) = O(y^3logx).

```
       función junio17(A:real[x,y]):real    Nivel 1
          i,j: entero
          v: real
(1)       v ← 0                             O(1)
(2)       i ← x                             O(1)
(3)       mientras i ≥ 1 hacer              O(y^3logx)
(4)          j ← 1
(5)          mientras j < y hacer
(6)             v ← v + calcular(A_i,j,y)
(7)             j ← j + 2
(8)          fmientras
(9)          i ← i / 2
(10)      fmientras
(11)      devolver v
       ffunción
```

Finalmente, aplicamos la regla de la suma en el Nivel 1: max{ O(1), O(1). O(y^3logx} }= O(y^3logx).

Con lo cual la complejidad asintótica del algoritmo es O(y^3logx).

3.- Dado el siguiente algoritmo, que resuelve el problema del viajante de comercio, obtén su versión iterativa.

```
costeVóptimo ← +∞
V_1 ← 1
función Viajante(Coste:real+[n,n], V:&natural[n], Vóptimo:&natural[n],
                 costeVóptimo:&real, k:entero)
   costeV:real
   V_k ← 1
   mientras V_k ≠ n hacer
      V_k ← V_k + 1
      si Coste_Vk-1,Vk ≠ ∞  y  Ciclos(V, k) = FALSO
         si k = n
            si Coste_Vk,V1 ≠ ∞
               costeV ← CalcularCoste(Coste, V)
               si costeV < costeVóptimo
                  Vóptimo ← V
                  costeVóptimo ← costeV
               fsi
            fsi
         si no
            Viajante(Coste, V, Vóptimo, costeVóptimo, k+1)
         fsi
      fsi
   fmientras
ffunción
```

Metodología de la Programación y Algoritmia
Convocatoria de Junio 2017
SOLUCIÓN

SOLUCIÓN:

El algoritmo está visto en los ejercicios del tema de vuelta atrás. Para realizar la conversión a iterativo utilizamos el Esquema general iterativo de los algoritmos de vuelta atrás adecuándolo a este caso particular.

La solución es la siguiente:

```
función Viajante(Coste:real+[n,n])
   V,Vóptimo:natural[n]
   costeV,costeVóptimo:real
   k:entero

   costeVóptimo ← +∞
   V1 ← 1

   k ← 2
   mientras k>1 hacer
      si Vk ≠ n
         Vk ← Vk + 1
         si CosteVk-1,Vk ≠ ∞  y  Ciclos(V, k) = FALSO
            si k = n
               si CosteVk,V1 ≠ ∞
                  costeV ← CalcularCoste(Coste, V)
                  si costeV < costeVóptimo
                     Vóptimo ← V
                     costeVóptimo ← costeV
                  fsi
               fsi
            si no
               k ← k+1
               Vk ← 1
           fsi
         fsi
      si no
         k ← k-1
      fsi
   fmientras
ffunción
```