PROGRAMACIÓN 2. Curso 2017-18. Grupo de tarde

2ª prueba voluntaria de evaluación

Esta es la segunda prueba voluntaria que se plantea en la asignatura ***Programación 2*** para la evaluación de los alumnos matriculados en el grupo de tarde. Tiene un valor de **10 puntos** y debe ser resuelta individualmente.

Primer problema [5.0 puntos]

El diseño de la función genérica **`organizar`**`(v,n)` se presenta a continuación. Se han etiquetado las líneas de su código con las primeras letras del alfabeto (**`a`**, **`b`**, ..., **`i`** y **`j`**). Se asume que el coste de ejecutar el código de cada una de dichas líneas es constante, es decir $\boldsymbol{O}(1)$, y va a ser denominado t_a, t_b, ..., t_i y t_j, respectivamente.

```
/*
 * Pre: n > 0
 * Post: No es necesario conocer este predicado para resolver el problema
 */
template <typename Dato>
void organizar (Dato v[], const int n) {
    for (int i = n - 2; i >= 0; --i) {                    // a
        Dato d = v[i];                                    // b
        int iLimite = n - 1;                              // c
        int j = i;                                        // d
        while (j != iLimite) {                            // e
            if (d > v[j+1]) {                             // f
                v[j] = v[j+1];                            // g
                j = j + 1;                                // h
            }
            else {
                iLimite = j;                              // i
            }
        }
        v[j] = d;                                         // j
    }
}
```

Se pide analizar el coste en tiempo de ejecutar **`organizar`**`(v,n)` en función del valor que tome el parámetro **`n`**, según el siguiente guión:

1. Explicar y justificar cuáles son las disposiciones iniciales de los **`n`** primeros elementos del vector **`v`** que proporcionan los casos asintóticamente peores y mejores en cuanto al coste en tiempo de **`organizar`**`(v,n)` o, en su caso, justificar si no cabe distinguir casos diferenciados ya que hay un único caso general para caracterizar asintóticamente el coste en tiempo.

2. Determinar la función de coste en tiempo, $t(n)$, en los casos extremos identificados en el apartado anterior o, si ha lugar, en el caso único general. Dicha función o funciones de coste se expresarán en función de los costes en tiempo t_a, t_b, ..., t_i y t_j de las diferentes líneas del algoritmo. Se deberán detallar y, en su caso, justificar los diferentes pasos dados para calcular la función o funciones de coste pedidas.

3. Caracterizar asintóticamente el orden de cada una de las funciones de coste en tiempo, $\boldsymbol{O}(t(n))$, determinadas en el apartado anterior.

Segundo problema [5.0 puntos]

Sea $t_{peor}(n)$ la función que determina el coste en tiempo necesario para ejecutar una invocación **elevarRec**(base,n,1,1.0,base) a la función cuyo diseño se presenta más adelante, en los casos peores que puedan presentarse. Y sea $t_{mejor}(n)$ la función que determina el coste en tiempo necesario para ejecutar una invocación **elevarRec**(base,n,1,1.0,base) a la misma función en los casos mejores que puedan presentarse.

En este problema se pide realizar un análisis del coste en tiempo de una invocación **elevarRec**(base,n,1,1.0,base) para valores de **n** suficientemente grandes que responda con claridad y detalle las siguientes preguntas:

1. Justificar por qué las funciones de coste en tiempo $t_{peor}(n)$ y $t_{mejor}(n)$ de una invocación **elevarRec**(base,n,1,1.0,base) dependen del valor del argumento **n** y no dependen del valor de los restantes argumentos.

2. Explicar y justificar para qué valores del argumento **n** se presentan los casos con costes en tiempo asintóticamente peores y mejores de una invocación **elevarRec**(base,n,1,1.0,base).

3. Asumiremos que el coste de ejecutar el código de cada una de las líneas del código de la función es constante es igual a t_a (el coste del código de la línea **a**), a t_b (el coste del código de la línea **b**), a t_c (el coste del código de la línea **c**), a t_d (el coste del código de la línea **d**) y a a t_e (el coste del código de la línea **e**). Se pide determinar la función de coste en tiempo $t_{peor}(n)$ de una invocación **elevarRec**(base,n,1,1.0,base) detallando los cálculos realizados y su justificación.

4. De un modo similar, se pide determinar la función de coste en tiempo $t_{mejor}(n)$ de una invocación **elevarRec**(base,n,1,1.0,base) detallando los cálculos realizados y su justificación.

5. Explicar cuál es la diferencia, si es que existe, entre las funciones de coste de la invocación **elevarRec**(base,n,1,1.0,base) en los casos peores y mejores.

```
/*
 * Pre: i >= 0 AND x^N = r * (x^i)^e  AND pot = x^e
 * Post: elevarRec(x,i,e,r,pot) = x^N
 */
double elevarRec (const double x, const int i, const int e, const double r, const double pot) {
    if (i != 0) {                                                          // a
        if (i % 2 != 0) {                                                  // b
            return elevarRec(x, i / 2, 2 * e, pot * r, pot * pot);         // c
        }
        else {
            return elevarRec(x, i / 2, 2 * e, r, pot * pot);               // d
        }
    }
    else {
        return r;                                                          // e
    }
}
```

Modo de entrega de la prueba

Cada alumno preparará un documento en papel, manuscrito o impreso, con la resolución de los dos problemas anteriores. El documento será entregado al profesor de la asignatura (al final de clase o en su despacho) con fecha límite el **viernes 4 de mayo de 2018**. No se admitirán trabajos presentados de otro modo, ni entregas fuera de plazo, ni trabajos plagiados total o parcialmente.

Una solución del primer problema

El coste en tiempo de ejecutar **`organizar`**`(v,n)` depende del número **`n`** de elementos del vector **`v`** a tratar pero, fijado un valor de **`n`** ***suficientemente grande*** (estamos haciendo un análisis asintótico del coste), también dependerá de la disposición de los **`n`** primeros elementos de **`v`**.

- **Caracterización de los casos mejores**. Se presentan cuando los datos inicialmente almacenados en **`v`** están ya ordenados de menor a mayor valor. En tales casos el bloque interior del bucle **`while`** solo se ejecuta una vez, en la que se no satisface la condición de la línea **`f`** que provoca la ejecución de la instrucción de la línea **`i`** y la consiguiente terminación del bucle al satisfacerse entonces que $j = iLimite$. El coste de ejecutar el bucle interior se limita a la siguiente suma de costes $t_e + t_f + t_i + t_e$.

  Función de coste en tiempo en los casos mejores. Se va a determinar la función de coste como suma de costes de las diferentes líneas a ejecutar:

  $$t_{caso\ mejor}(n) = t_{inv} + t_a + (\Sigma\alpha \in [1, n-1].\ t_b + t_c + t_d + t_e + t_f + t_i + t_e + t_j + t_a)$$
  $$t_{caso\ mejor}(n) = t_{inv} + t_a + (t_b + t_c + t_d + 2 \times t_e + t_f + t_i + t_j + t_a) \times (n-1)$$

  Caracterización asintótica de la función de coste en tiempo en los casos mejores.

  $$\boldsymbol{O}(t_{caso\ mejor}(n)) = \boldsymbol{O}(t_{inv} + t_a + (t_b + t_c + t_d + 2 \times t_e + t_f + t_i + t_j + t_a) \times (n-1))$$
  $$\boldsymbol{O}(t_{caso\ mejor}(n)) = \boldsymbol{O}((t_b + t_c + t_d + 2 \times t_e + t_f + t_i + t_j + t_a) \times (n-1))$$
  $$\boldsymbol{O}(t_{caso\ mejor}(n)) = \boldsymbol{O}(n-1) = \boldsymbol{O}(n)$$

- **Caracterización de los casos peores**. Se presentan cuando los **`n`** primeros elementos de **`v`** están ordenados de mayor a menor valor. En tales casos en cada iteración del bucle **`for`** hay que insertar el dato **`v`**`[i]` en la ubicación `v[n-1]` (línea **`j`**), tras haber desplazado cada uno de los elementos de **`v`**`[i+1..n-1]` a la posición anterior (línea **`g`**). El bloque interior del bucle **`while`** se ejecuta **`n-1-i`** veces y el coste de cada una de estas iteraciones es $t_f + t_g + t_h + t_e$. La ejecución del bucle interior **`while`** concluye cuando se satisface que $j = n - 1$.

  Función de coste en tiempo en los casos peores. Se va a determinar la función de coste como suma de costes de las diferentes líneas a ejecutar:

  $$\begin{aligned} t_{caso\ peor}(n) = {} & t_{inv} + t_a + (\Sigma\alpha \in [1, n-1].\ t_b + t_c + t_d + t_e + \\ & (\Sigma\beta \in [1, n-1-i].\ t_f + t_g + t_h + t_e) + t_j + t_a) + t_j + t_a) \end{aligned}$$
  $$\begin{aligned} t_{caso\ peor}(n) = {} & t_{inv} + t_a + (t_b + t_c + t_d + t_e + t_j + t_a) \times (n-1) + \\ & (t_f + t_g + t_h + t_e) \times (\Sigma\alpha \in [1, n-1].\ n-1-i) \end{aligned}$$

  Si tenemos en cuenta que $\alpha = n - 1 - i$ entonces:

  $$\begin{aligned} t_{caso\ peor}(n) = {} & t_{inv} + t_a + (t_b + t_c + t_d + t_e + t_j + t_a) \times (n-1) + \\ & (t_f + t_g + t_h + t_e) \times (\Sigma\alpha \in [1, n-1].\ \alpha) \end{aligned}$$
  $$t_{caso\ peor}(n) = t_{inv} + t_a + (t_b + t_c + t_d + t_e + t_j + t_a) \times (n-1) + (t_f + t_g + t_h + t_e)\tfrac{1}{2}n(n-1)$$
  $$t_{caso\ peor}(n) = (t_f + t_g + t_h + t_e)\tfrac{1}{2}n(n-1) + (t_b + t_c + t_d + t_e + t_j + t_a) \times (n-1) + t_{inv} + t_a$$

  Caracterización asintótica de la función de coste en tiempo en los casos peores.

  $$\begin{aligned} \boldsymbol{O}(t_{caso\ peor}(n)) = \boldsymbol{O}((t_f + t_g + t_h + t_e)\tfrac{1}{2}n(n-1) + \\ (t_b + t_c + t_d + t_e + t_j + t_a) \times (n-1) + t_{inv} + t_a) \end{aligned}$$
  $$\boldsymbol{O}(t_{caso\ peor}(n)) = \boldsymbol{O}((t_f + t_g + t_h + t_e)\tfrac{1}{2}n(n-1))$$
  $$\boldsymbol{O}(t_{caso\ peor}(n)) = \boldsymbol{O}(n(n-1)) = \boldsymbol{O}(n \times n) = \boldsymbol{O}(n^2)$$

Una solución del segundo problema

Se van a responder las preguntas planteadas sobre el coste, en tiempo, de una invocación **`elevarRec`**`(base,n,1,1.0,base)`.

1. El coste, en tiempo, de una invocación **`elevarRec`**`(base,n,1,1.0,base)` depende del valor que toma el parámetro **`i`** (el valor determinado por el argumento **`n`**) ya que de él depende el número de veces que se invoca recursivamente la función **`elevarRec`**`(...)`. Asímismo, el valor de **`i`** en cada invocación recursiva determina si ha de ejecutarse el código de la línea **`c`**, ligeramente más costoso por el cálculo del valor del cuarto argumento, $pot * r$, o el de la línea **`d`** cuyo cuarto argumento es r.

2. Los casos asintóticamente peores y mejores hay que analizarlos para un valor del argumento **`n`** suficiente grande.

   Los casos mejores en tiempo se presentan cuando en todas en las invocaciones recursivas se ejecuta la línea **`d`**, que es ligeramente menos costosa que la línea **`c`**. Esto sucede cuando el valor de **`n`** es una potencia de 2, es decir, $n = 2^K$.

   Los casos peores se presentan cuando en todas en las invocaciones recursivas, excepto en la última, se ejecuta la línea **`c`**, que es ligeramente más costosa que la línea **`d`**. Esto sucede cuando el valor de **`n`** es una unidad inferior a una potencia de 2, es decir, $n = 2^K - 1$.

3. Función de coste en el caso peor, $t_{peor}(n)$. El caso peor se presenta cuando $n = 2^K - 1$. En tales casos, en todas las invocaciones a la función **`elevarRec`**`(...)` se ejecuta la línea **`c`**, excepto en la última invocación. Por lo tanto, podemos escribir la siguiente recurencia:

   $t_{peor}(n) = t_{inv} + t_a + t_b + t_c + t_{peor}(n/2)$

   Donde $t_{peor}(n/2)$ determina el coste de la invocación recursiva **`elevarRec`**$(x, i/2, 2 * e, pot * r, pot * pot)$ de la línea **`c`** y t_c el coste de ejecutar dicha línea **`c`** a excepción del coste de la invocación recursiva mencionada.

   Procedemos a resolver la recurrencia anterior aplicando el cambio de variable $m = log_2\ n$.

   $t_{peor}(n) - t_{peor}(n/2) = t_{inv} + t_a + t_b + t_c$

   $t_{peor}(m) - t_{peor}(m-1) = (t_{inv} + t_a + t_b + t_c) \times 1^m$

   Ecuación característica:

   $(x-1)(x-1) = 0$

   Sus raíces son:

   $x = 1$ (raíz doble)

   Podemos escribir la solución de la recurrencia en función de la variable **`m`**:

   $t_{peor}(m) = c_1 \times m \times 1^m + c_2 \times 1^m = c_1 \times m + c_2$

   Si deshacemos el cambio de variable, obtenemos la función de coste en tiempo en el caso peor dependiente del valor de **`n`**.

   $t_{peor}(n) = c_1 \times log_2\ n + c_2$

4. Función de coste en el caso mejor, $t_{mejor}(n)$. El caso peor se presenta cuando $n = 2^K$. En tales casos, en todas las invocaciones a la función **`elevarRec`**`(...)` se ejecuta la línea **`d`**. Por lo tanto:

   $t_{mejor}(n) = t_{inv} + t_a + t_b + t_d + t_{mejor}(n/2)$

   Donde $t_{mejor}(n/2)$ determina el coste de la invocación recursiva **`elevarRec`**$(x, i/2, 2 * e, r, pot * pot)$ de la línea **`d`** y t_d el coste de ejecutar dicha línea **`d`** a excepción del coste de la invocación recursva mencionada.

   Procedemos a resolver la recurrencia anterior de forma idéntica al caso peor, aplicando el cambio de variable $m = log_2\ n$.

$t_{mejor}(n) - t_{mejor}(n/2) = t_{inv} + t_a + t_b + t_d$

$t_{mejor}(m) - t_{mejor}(m-1) = (t_{inv} + t_a + t_b + t_d) \times 1^m$

Ecuación característica:

$(x-1)(x-1) = 0$

Sus raíces son:

$x = 1$ (raíz doble)

Podemos escribir la solución de la recurrencia en función de la variable **`m`**:

$t_{mejor}(m) = c_3 \times m \times 1^m + c_4 \times 1^m = c_3 \times m + c_4$

Si deshacemos el cambio de variable, obtenemos la función de coste en tiempo en el caso mejor dependiente del valor de **`n`**.

$t_{mejor}(n) = c_3 \times log_2\, n + c_4$

5. Las dos funciones de coste son logarítmicas en el valor del parámetro **`n`**. Su diferencia fundamental está en el valor del coeficiente del término logarítmico, ligeramente mayor en el caso peor, el coeficiente c_1, que el coeficiente c_3 de la función de coste en el caso mejor, por el menor coste de ejecutar la línea **`d`** respecto del de ejecutar la línea **`c`** en cada invocación recursiva.