Primer Parcial de PRG - ETSInf
Fecha: 21 de marzo de 2011. Duración: 1 hora y 30 minutos

NOTA: Se debe responder en hojas aparte. No es necesario entregar esta hoja.

1. (3 puntos) El siguiente método **iterativo** calcula el producto de un vector fila `x` por una matriz cuadrada `a`, obteniendo como resultado otro vector fila. Por tratarse de una matriz cuadrada tanto el vector `x` como el vector resultado tendrán el mismo tamaño, es decir, el número de filas de la matriz.

```
/**  x.length = a.length  y a es una matriz cuadrada */
static double [] vectorPorMatriz( double x[], double a[][] )
{
    double r[] = new double [ a.length ];

    for( int i=0; i < r.length; i++ ) {
        r[i] = 0.0;
        for( int j=0; j < x.length; j++ )
            r[i] += x[j] * a[j][i];
    }
    return r;
}
```

Por ejemplo, para $x = (2, 1, 3)$, un vector fila de dimensión 1×3, y $a = \begin{pmatrix} 4 & 3 & 2 \\ 2 & 5 & 1 \\ 1 & 0 & 3 \end{pmatrix}$, una matriz de dimensión 3×3, el producto $x \times a$ es $r = (13, 11, 14)$, un vector fila de dimensión 1×3.

Se pide:

a) Indicar cuál es el tamaño o talla del problema, así como la expresión que lo representa.

b) Identificar, caso de que las hubiere, las instancias del problema que representan el caso mejor y peor del algoritmo.

c) Elegir una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y acorde con ella, obtener una expresión matemática, lo más precisa posible, del coste temporal del programa, a nivel global o en las instancias más significativas si las hay.

d) Expresar el resultado anterior utilizando notación asintótica.

Solución:

1. La talla o tamaño del problema es el número de filas de la matriz, `a.length`, que a su vez coincide con el de columnas de la propia matriz y con el número de componentes del vector `x`. En adelante, llamaremos a ese número n, es decir, $n =$ `a.length`.

2. Se trata de un problema de recorrido, por tanto, para una misma talla del problema no hay instancias significativas.

3. Si se toma como unidad de medida el paso de programa, se puede expresar el coste del algoritmo de la forma siguiente: $T(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(1+n) = 1 + n + n^2$.

   Si se elige como unidad de medida para la estimación del coste una instrucción crítica, en concreto, seleccionando la asignación más interna `r[i] += x[j] * a[j][i]`, obtenemos la siguiente función de coste temporal: $T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n^2$.

4. En notación asintótica: $T(n) \in \Theta(n^2)$.

2. (3.5 puntos) Para obtener la posición del último elemento impar de un array `v` de enteros, se propone el siguiente método **iterativo**:

```
static int posUltimoImpar( int[] v )
{
    int i = v.length-1;
    while( i >= 0 && v[i]%2 == 0 ) i--;
    return i;
}
```

Se pide:

a) Indicar cuál es el tamaño o talla del problema, así como la expresión que lo representa.

b) Identificar, caso de que las hubiere, las instancias del problema que representan el caso mejor y peor del algoritmo.

c) Elegir una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y acorde con ella, obtener una expresión matemática, lo más precisa posible, del coste temporal del programa, a nivel global o en las instancias más significativas si las hay.

d) Expresar el resultado anterior utilizando notación asintótica.

Solución:

a) El tamaño o talla del problema es el número de elementos del array `v` y la expresión que lo representa es `v.length`. En adelante, denominaremos a ese número n. Esto es, $n =$ `v.length`.

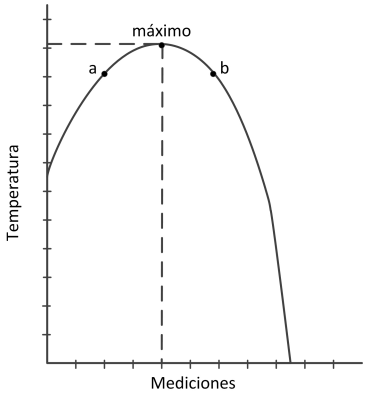
b) Se trata de un problema de búsqueda (secuencial iterativa descendente) y, por tanto, para una misma talla sí que presenta instancias distintas. El *caso mejor* se da cuando el último elemento del array es impar (`v[v.length-1]%2==1`). El *caso peor* ocurre cuando todos los elementos del array son pares (∀`i`, `0`≤`i`<`v.length`, `v[i]%2==0`).

c) Optamos por elegir la instrucción crítica como unidad de medida para la estimación del coste. La única instrucción que podemos considerar como crítica es la guarda del bucle: `i>=0 && v[i]%2==0` (se ejecuta siempre una vez más que cualquiera de las de dentro del bucle). En el *caso mejor* sólo se ejecutará una vez y la función de coste temporal en este caso será $T^m(n) = 1$. En el *caso peor* se repetirá tantas veces como elementos tenga el array más una (cuando se hace falsa) y la función de coste será $T^p(n) = \sum_{i=0}^{n} 1 = n + 1$.

d) En notación asintótica: $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(n)$. Por tanto, $T(n) \in \Omega(1)$ y $T(n) \in O(n)$, es decir, el coste temporal está acotado inferiormente por una función constante y superiormente por una función lineal con la talla del problema.

3. (3.5 puntos) Se dispone de una serie de mediciones de temperatura que forman parte de una curva parabólica (positiva) en un array `v` de reales. El siguiente método **recursivo** obtiene el máximo valor de la curva representada por el array `v`.

Para este tipo de curvas, dado un valor se sabe que si el anterior y el posterior son menores que él, nos hallamos en el punto máximo medido. Si por el contrario el anterior es mayor y el posterior es menor, estamos a la derecha del máximo, y si el anterior es menor y el posterior mayor, estamos a la izquierda del máximo. Otra situación es imposible en este tipo de curva. En la curva de la figura, se puede observar que los puntos `a` y `b` (anterior y posterior al punto máximo, respectivamente) son menores que el punto `máximo`.




```
static double maximaTemp( double[] v, int ini, int fin )
{
    if ( ini == fin )
        return v[ini];
    else if ( ini == fin-1 )
        return Math.max( v[ini], v[fin] );
    else {
        int mitad = (fin+ini)/2;
        if ( v[mitad] > v[mitad-1] && v[mitad] > v[mitad+1] )
            return v[mitad];
        else if ( v[mitad] < v[mitad-1] && v[mitad] > v[mitad+1] )
            return maximaTemp( v, ini, mitad );
        else
            return maximaTemp( v, mitad, fin );
    }
}
```

La llamada inicial será: `double maximo = maximaTemp( v, 0, v.length-1 );`

Se pide:

a) Indicar cuál es la talla del problema y qué expresión la define.

b) Determinar si existen instancias significativas. Si las hay, identificar las que representan los casos mejor y peor del algoritmo.

c) Escribir la ecuación de recurrencia del coste temporal en función de la talla para cada uno de los casos si hubiera varios, o una única ecuación si sólo hubiera un caso. Resolverla por sustitución.

d) Expresar el resultado anterior usando notación asintótica.

Solución:

1. La talla del problema es el número de elementos del array `v` entre las posiciones `ini` y `fin`. Es decir, la expresión de la talla es $n =$ `fin-ini+1`.

2. Se trata de un problema de búsqueda (binaria), por lo que para una misma talla sí se distinguen instancias. El *caso mejor* se da cuando el punto máximo se encuentra en `v[(fin+ini)/2]` siendo `ini=0` y `fin=v.length-1`. El *caso peor* se da cuando para encontrar el punto máximo se hacen el mayor número de llamadas posible, es decir, cuando la talla del problema se reduce hasta llegar a la talla del caso base.

3. En el *caso mejor*, el coste es constante, $T^m(n) = c$. Para obtener el coste en el *caso peor*, planteamos la ecuación de recurrencia: $T^p(n) = \begin{cases} T^p(n/2) + k & \text{si } n > 2 \\ k' & \text{si } n = 1 \text{ o } n = 2 \end{cases}$

   Resolviendo por sustitución: $T^p(n) = T^p(n/2) + k = T^p(n/2^2) + 2k = \ldots = T^p(n/2^i) + ik$. Al llegar al caso base, para talla 1, $n/2^i = 1 \rightarrow i = log_2 n$; y para talla 2, $n/2^i = 2 \rightarrow n = 2^i * 2 \rightarrow i = log_2 n + log_2 2 = log_2 n + 1$. Con lo que $T^p(n) \approx k' + k log_2 n$.

4. En notación asintótica: $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(log_2 n)$. Por tanto, $T(n) \in \Omega(1)$ y $T(n) \in O(log_2 n)$, es decir, el coste temporal está acotado inferiormente por una función constante y superiormente por una función logarítmica con la talla del problema.