PRG - ETSInf. TEORÍA. Curso 2015-16. Recuperación Parcial 1.
GII. GIINF - GADE
17 de junio de 2016. Duración: 2 horas.

1. [4 puntos] Dado un array `a` de `int` donde `a.length` $\geq$ `1`, escribir un método **recursivo** que determine si los elementos del array representan una sucesión de Fibonacci, es decir, si el valor de un elemento corresponde a la suma de los dos elementos inmediatamente anteriores (siendo los dos primeros elementos 0 y 1), tal y como se muestra en el siguiente ejemplo: {`0,1,1,2,3,5,8,13,21,...`}.

   Se pide:

   a) (0.75 puntos) Perfil del método, con los parámetros adecuados para resolver recursivamente el problema.

   b) (1.25 puntos) Caso base y caso general.

   c) (1.5 puntos) Implementación en Java.

   d) (0.5 puntos) Llamada inicial para que determine si se cumple la sucesión sobre todo el array.

Solución:

a) Una posible solución consiste en definir un método con el siguiente perfil:

```
/** Precondición: a.length >= 1 y 0 <= pos <= a.length - 1. */
public static boolean esFibo(int[] a, int pos)
```

de modo que determine si los elementos de `a[0..pos]` representan una sucesión de Fibonacci, siendo `0` $\leq$ `pos` $\leq$ `a.length - 1`.

b)
- Caso base, `pos = 0`: Verifica que `a[0] = 0`.
- Caso base, `pos = 1`: Verifica que `a[1] = 1` y que `a[0] = 0`.
- Caso general, `pos` $>$ `1`: Subarray de tres o más elementos. Verifica que `a[pos] = a[pos - 1] + a[pos - 2]`.

c)
```
/** Determina si los elementos de a[0..pos] representan una sucesión de Fibonacci.
 *  Precondición: a.length >= 1 y 0 <= pos <= a.length - 1. */
public static boolean esFibo(int[] a, int pos) {
    if (pos == 0) { return a[0] == 0; }
    else if (pos == 1) { return a[1] == 1 && a[0] == 0; }
    else { return a[pos] == a[pos - 1] + a[pos - 2] && esFibo(a, pos - 1); }
}
```

d) Para un array `a`, la llamada `esFibo(a, a.length - 1)` resuelve el problema del enunciado.

2. [3 puntos] Mediante el siguiente método se comprueba si todas las filas de la matriz `m`, cuadrada, de `double`s, suman siempre un mismo valor. Se sabe que `m` es al menos de orden 2 (tiene un número de filas y columnas mayor o igual que 2).

```
/** PRECONDICIÓN: m es cuadrada de orden mayor o igual que 2. */
public static boolean sumanIgual(double[][] m) {
    // Se calcula la suma de la primera fila:
    int sum0 = 0;
    for (int i = 0; i < m.length; i++) { sum0 += m[0][i]; }

    boolean sumIgual = true;
    // Se calcula la suma de cada fila posterior:
    for (int i = 1; i < m.length && sumIgual; i++) {
        int sumFil = 0;
        for (int j = 0; j < m.length; j++) { sumFil += m[i][j]; }
        sumIgual = (sum0 == sumFil);
    }

    return sumIgual;
}
```

Se pide:

a) (0.25 puntos) Indicar cuál es el tamaño o talla del problema, así como la expresión que la representa.

b) (0.75 puntos) Indicar si existen diferentes instancias significativas para el coste temporal del algoritmo e identificarlas si es el caso.

c) (1.50 puntos) Elegir una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y de acuerdo con ella obtener una expresión matemática, lo más precisa posible, del coste temporal del método, distinguiendo el coste de las instancias más significativas en caso de haberlas.

d) (0.50 puntos) Expresar el resultado anterior utilizando notación asintótica.

Solución:

a) La talla del problema es la dimensión de la matriz `m`, es decir, `m.length`. De ahora en adelante, denominaremos a este número n. Esto es, n=`m.length`.

b) Sí que existen diferentes instancias ya que se trata de la búsqueda de la primera fila de la matriz `m`, si existe, cuya suma sea diferente de la de la primera.

El caso mejor es cuando la suma de los elementos de la primera fila de `m` es diferente de la suma de los de la segunda (con lo cual se recorrerán solamente los elementos de la primera y segunda filas), mientras que el caso peor se dará cuando la suma de los elementos de todas las filas de la matriz `m` valgan lo mismo (y se recorrerán por completo todas las filas).

c) Eligiendo como unidad de medida el paso de programa, se tiene:

- Caso mejor:

$$T^m(n) = 1 + \sum_{i=0}^{n-1} 1 + \sum_{j=0}^{n-1} 1 = 2n + 1 \ p.p.$$

- Caso peor:

$$T^p(n) = 1 + \sum_{i=0}^{n-1} 1 + \sum_{i=1}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + n + \sum_{i=1}^{n-1}(1 + n) = 1 + n + (n-1)(n+1) = n^2 + n \ p.p.$$

Eligiendo como unidad de medida la instrucción crítica y considerando como tal la instrucción `sum0 += m[0][i];` del primer bucle y la instrucción `sumFil += m[i][j];` del segundo bucle (ambas de coste unitario), se tiene:

- Caso mejor:

$$T^m(n) = \sum_{i=0}^{n-1} 1 + \sum_{j=0}^{n-1} 1 = 2n \ i.c.$$

- Caso peor:

$$T^p(n) = \sum_{i=0}^{n-1} 1 + \sum_{i=1}^{n-1}\sum_{j=0}^{n-1} 1 = n + \sum_{i=1}^{n-1} n = n + n(n-1) = n^2 \ i.c.$$

d) En notación asintótica: $T^m(n) \in \Theta(n)$ y $T^p(n) \in \Theta(n^2)$. Por tanto, $T(n) \in \Omega(n)$ y $T(n) \in O(n^2)$.

3. 3 puntos Dada la siguiente implementación de un algoritmo recursivo que resuelve la multiplicación *a la rusa* de dos números naturales `a` y `b`:

```
/** PRECONDICIÓN: a >= 0 y b >= 0. */
public static int productoRuso(int a, int b) {
    if (b == 0) { return 0; }
    else {
        if (b % 2 == 0) { return productoRuso(a * 2, b / 2); }
        else { return a + productoRuso(a * 2, b / 2); }
    }
}
```

Se pide: analizar el coste temporal de este algoritmo. En concreto:

a) (0.25 puntos) Indicar cuál es la talla del problema, y qué expresión la define.

b) (0.5 puntos) Identificar, caso de que las hubiera, las instancias del problema que representan el caso mejor y peor del algoritmo.

c) (1.5 puntos) Elegir una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y de acuerdo con ella obtener una expresión matemática, lo más precisa posible, de la ecuación de recurrencia del coste temporal en función de la talla (una única ecuación si no hubiera instancias significativas; dos ecuaciones, correspondientes a los casos mejor y peor, en el caso de que el problema tuviera instancias significativas). Resolverla(s) por sustitución.

d) (0.75 puntos) Expresar el resultado anterior utilizando notación asintótica.

Solución:

a) La talla es el valor del segundo argumento, el entero `b`, que llamaremos n de aquí en adelante. El entero `a`, no influye en el número de llamadas recursivas que se generan, ni en el coste temporal de las operaciones no recursivas, por lo que no tiene efecto alguno en el coste del algoritmo.

b) Dado que se trata de un algoritmo que *recorre* la secuencia de valores b, $\frac{b}{2}$, $\frac{b}{4}$, $\frac{b}{8}$ ... no existen instancias significativas.

c) El coste de las operaciones que se ejecutan en el caso base ($n = 0$) es constante, supongamos que igual a $c_0 \in \Re^+$. El coste de las operaciones no recursivas que se ejecutan en el caso general ($n > 0$) es también constante, supongamos que igual a $c_1 \in \Re^+$. En el caso general se realiza una llamada recursiva en la que la talla del problema (n) se reduce a la mitad. En consecuencia, se puede escribir la siguiente ecuación de recurrencia:

$$T(n) = \begin{cases} c_0 & \text{si } n = 0 \\ c_1 + T(\frac{n}{2}) & \text{si } n > 0 \end{cases}$$

expresada en pasos de programa ($p.p.$).

Resolviendo por sustitución: $T(n) = c_1 + T(\frac{n}{2^1}) = 2c_1 + T(\frac{n}{2^2}) = 3c_1 + T(\frac{n}{2^3}) = \ldots = kc_1 + T(\frac{n}{2^k})$. En el caso base, la talla es 0, y el parámetro de la función T tendría que tomar ese valor, dando lugar a la igualdad: $\frac{n}{2^k} = 0$. Para poder despejar k, cambiamos esta igualdad a $\frac{n}{2^k} = 1$ y obtenemos el valor de $k = \log_2 n$. Sustituyendo el valor de k en el término general $kc_1 + T(\frac{n}{2^k})$, obtenemos:

$$T(n) = c_1 \log_2 n + T(1) = c_1 \log_2 n + (c_1 + T(0)) = c_1 \log_2 n + c_1 + c_0 \; p.p.$$

d) En notación asintótica: $T(n) \in \Theta(\log n)$, es decir, el coste temporal del algoritmo depende logarítmicamente de la talla del problema.