PRG - ETSInf. TEORÍA. Curso 2016-17. Recuperación Parcial 1.
26 de junio de 2017. Duración: 2 horas.

Nota: El examen se evalúa sobre 10 puntos pero su peso específico en la nota final de PRG es de **3 puntos**.

1. 3 puntos Un número triangular es aquel que puede recomponerse en la forma de un triángulo equilátero (consideraremos que el primer número triangular es el 0). Los primeros 10 números triangulares son: 0, 1, 3, 6, 10, 15, 21, 28, 36, 45.

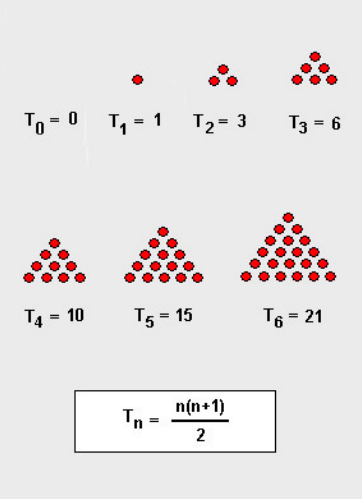

Figura 1: Representación gráfica de los 7 primeros números triangulares

Cada número triangular se puede calcular mediante:

$$t(n) = \begin{cases} 0 & n = 0 \\ n + t(n-1) & n > 0 \end{cases}$$

Siendo $t(n)$ el enésimo número triangular y $t(n-1)$ su anterior.

Se quiere implementar, usando recursión, un algoritmo que permita instanciar un array de un cierto tamaño n, y que contenga los n primeros términos de esta sucesión de los números triangulares. Se pide implementar los siguientes métodos, en los que además se deberá especificar las precondiciones necesarias:

- Un primer método, lanzadera, no recursivo, que reciba un único argumento, el número n, que será un valor entero positivo. Este método instanciará el array al tamaño adecuado, invocará al segundo método (encargado de rellenar el array), y devolverá el array.
- Un segundo método, recursivo, que reciba dos argumentos: el array de tamaño n y otro argumento, necesario para implementar el diseño recursivo. En este método se calcularán los valores de la sucesión, almacenándolos en las posiciones que corresponda en el array.

Por ejemplo, la llamada al método lanzadera con $n = 5$ deberá devolver el array $\{0, 1, 3, 6, 10\}$, mientras que con $n = 1$ devolverá el array $\{0\}$.

Solución:

Una posible solución consiste en calcular cada número triangular, en el caso general, mediante la ecuación $t(n) = n + t(n-1)$.

```
/** Precondición: n>0 */
public static int[] creaSucesionTriangulares(int n) {
    int[] a = new int[n];
    creaSucesionTriangulares(a, n - 1);
    return a;
}
```

```
/** Precondición: n>=0 && n<a.length */
private static void creaSucesionTriangulares(int[] a, int n) {
    if (n == 0) { a[n] = 0; }
    else {
        creaSucesionTriangulares(a, n - 1);
        a[n] = n + a[n - 1];
    }
}
```

Otra posible solución consiste en calcular cada número triangular mediante la ecuación $t(n) = n*(n+1)/2$ mostrada en la Figura 1. En este caso se presenta, además, una solución recursiva basada en descomposición ascendente.

```
/** Precondición: n>0 */
public static int[] creaSucesionTriangulares(int n) {
    int[] a = new int[n];
    creaSucesionTriangulares(a, 0);
    return a;
}

/** Precondición: n>=0 && n<a.length */
private static void creaSucesionTriangulares(int[] a, int n) {
  if (n < a.length) {
    a[n] = n * (n + 1) / 2;
    creaSucesionTriangulares(a, n + 1);
  }
}
```

2. 4 puntos El siguiente método, dado un array `a` ordenado ascendentemente, devuelve el número de veces que aparece en el array el primer número repetido (el de valor más pequeño) o 0 si no hay repetidos. Por ejemplo, si `a = {2, 5, 5, 5, 8, 8, 9}` devuelve `3` (el primer número que aparece repetido es el `5` y se repite `3` veces); si `a = {2, 5, 8, 9}` devuelve `0`.

```
/** Precondición: a ordenado ascendentemente */
public static int contarPrimerRepetido(int[] a) {
    int cont = 0, i = 0;
    boolean repe = false;
    while (i < a.length - 1 && !repe) {
        int j = i + 1;
        repe = (a[j] == a[i]);
        while (j < a.length && a[j] == a[i]) { j++; }
        if (repe) { cont = j - i; }
        i++;
    }
    return cont;
}
```

Se pide:

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

e) (1 punto) Teniendo en cuenta que el algoritmo de ordenación por *inserción directa* visto en clase tiene un coste lineal en el mejor caso y cuadrático en el peor, ¿cuál es el coste de ordenar `a` por inserción directa y a continuación aplicar el método `contarPrimerRepetido(int[])`?

Solución:

a) La talla del problema es el número de elementos del array `a` y la expresión que la representa es `a.length`. De ahora en adelante, llamaremos a este número n. Esto es, $n =$ `a.length`.

b) Sí que existen diferentes instancias ya que se trata de una búsqueda del primer valor repetido que contiene otra búsqueda para contar cuántas veces se repite este valor.

El caso mejor se da cuando `a[0]` está repetido en `a[1]`, esto es, `a[0] == a[1]`.

El caso peor se da cuando no hay elementos repetidos en el array y también cuando `a[0]` está repetido en todo el array, esto es, $\forall$`i:1` $\leq$ `i <a.length: a[0] == a[i]`.

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

- Caso mejor: $T^m(n) = 3$ *p.p.*
- Caso peor: $T^p(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ *p.p.*

Escogiendo como unidad de medida la instrucción crítica y considerando como tal la guarda del bucle interno `j < a.length && a[j] == a[i]` (de coste unitario), se tiene:

- Caso mejor: $T^m(n) = 2$ *i.c.*
- Caso peor: $T^p(n) = \sum_{i=0}^{n-2} 1 = n - 1$ *i.c.*

d) En notación asintótica: $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(n)$. Por lo tanto, $T(n) \in \Omega(1)$ y $T(n) \in O(n)$.

e) La resolución del problema aplicando primero inserción directa y después el método `contarPrimerRepetido` tendrá la siguiente complejidad:

- En el caso mejor, el array está ordenado ascendentemente y `a[0]` está repetido en `a[1]`. Aplicar la ordenación por inserción directa tendrá un coste $\Theta(n)$ y aplicar el método `contarPrimerRepetido` será $\Theta(1)$. Por lo tanto, el coste en el caso mejor será lineal con la talla del problema.
- En el caso peor, los elementos del array están ordenados descendentemente y no hay repetidos. El coste de la ordenación por inserción directa será $\Theta(n^2)$ y aplicar el método `contarPrimerRepetido` será $\Theta(n)$. Por lo tanto, el coste en el caso peor será cuadrático con la talla del problema.

3. 3 puntos El siguiente método calcula la suma de los bits de la representación en binario de un entero no negativo `num`.

```
/** Precondición: num >= 0 */
public static int sumaBits(int num) {
    if (num <= 1) { return num; }
    else {
        int ultBit = num % 2;
        return sumaBits(num / 2) + ultBit;
    }
}
```

Se pide:

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

b) (0.5 puntos) Indicar si existen diferentes instancias significativas para el coste temporal del algoritmo e identificarlas si es el caso.

c) (1.5 puntos) Escribir la ecuación de recurrencia del coste temporal en función de la talla para cada uno de los casos si hay más de uno, o una única ecuación si hay un solo caso. Se debe resolver por sustitución.

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

Solución:

a) La talla del problema es `num`. De ahora en adelante, llamaremos a este parámetro n. Esto es, $n = num$.

b) No existen instancias significativas; para una misma talla `n`, el método siempre realiza el mismo número de operaciones.

c) La ecuación de recurrencia se define como:

$$T(n) = \begin{cases} T(n/2) + 1 & \text{si } n > 1 \\ 1 & \text{si } n \leq 1 \end{cases}$$

Resolviendo por sustitución:

$T(n) = T(n/2) + 1 = T(n/4) + 2 = T(n/8) + 3 = \ldots = T(n/2^i) + i$.

Si $n/2^i = 1 \rightarrow i = log_2 n$, con lo que $T(n) = T(n/2^{log_2 n}) + log_2 n = T(1) + log_2 n = 1 + log_2 n$ *p.p.*

d) En notación asintótica: $T(n) \in \Theta(log_2 n)$

Una solución alternativa consiste en tomar como talla el número de cifras de la representación en base 2 del valor *num*. En este caso:

a) $m =$ número de cifras de la representación en base 2 de *num*.

b) No existen instancias significativas; para una misma talla `m`, el método siempre realiza el mismo número de operaciones.

c) La ecuación de recurrencia se define como:

$$T(m) = \begin{cases} T(m-1) + 1 & \text{si } m > 1 \\ 1 & \text{si } m \leq 1 \end{cases}$$

Resolviendo por sustitución:

$T(m) = T(m-1) + 1 = T(m-2) + 2 = T(m-3) + 3 = \ldots = T(m-i) + i$.

Si $m - i = 1 \rightarrow i = m - 1$, con lo que $T(m) = T(1) + m - 1 = 1 + m - 1 = m$ *p.p.*

d) En notación asintótica: $T(m) \in \Theta(m)$

Como se puede observar, el coste temporal del método es el mismo, aunque expresado en función de tallas distintas. Recuerda que el número de cifras de la representación en base 2 de un número entero n es $1 + \lfloor log_2 n \rfloor$, esto es, m.