Primer Parcial de PRG - ETSInf
Fecha: 7 de mayo de 2012. Duración: 2 horas y 30 minutos

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

1. (2 puntos) Escribe un método **recursivo** que reciba como argumento un valor entero no negativo y que escriba en la salida estándar las secuencias descendente y ascendente de valores enteros desde el número inicial hasta el cero. Cada secuencia en una línea diferente.

   Por ejemplo, si el método solicitado se ejecuta con el valor 14, entonces escribirá por la salida estándar lo siguiente:

   ```
   14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
   ```

   Nota que los números de cada secuencia aparecen todos seguidos en una misma línea.

   Solución:

   ```
   public class Problema1 {
       public static void main( String[] args ) {
           secuencias( 14 );
           System.out.println();
       }

       /** n>=0 */
       private static void secuencias( int n ) {
           if ( n > 0 ) {
               System.out.print( n + " " );
               secuencias( n-1 );
               System.out.print( " " + n );
           } else {
               System.out.print( "0\n0" );
           }
       }
   }
   ```

2. (2 puntos) Escribe un método **recursivo** que dado un array de enteros `v`, una posición inicial `ini>=0`, una posición final `fin<v.length` y un valor entero `x` determine si la suma de los elementos simétricos es `x`, es decir, los extremos de los subarrays formados por las posiciones en el intervalo `[ini+i, fin-i]` para `0<=i<=(ini+fin)/2` cumplen que `v[ini+i]+v[fin-i] == x`.

   Por ejemplo:

   Si `x = 10, ini = 1, fin = v.length-2` y `v = {1,2,3,4,5,6,7,8,9}` devuelve `true`.

   Si `x = 10, ini = 1, fin = v.length-2` y `v = {1,2,4,4,5,6,7,8,9}` devuelve `false`.

   Si `x = 6, ini = 0, fin = v.length-1` y `v = {1,2,3,3,4,5}` devuelve `true`.

   Solución:

```
public class Problema2 {
    public static void main( String[] args ) {
        int[] a = {1,2,3,4,5,6,7,8,9};
        int[] b = {1,2,4,4,5,6,7,8,9};
        int[] c = {1,2,3,3,4,5};

        System.out.println( metodo( a, 1, a.length-2, 10 ) );
        System.out.println( metodo( b, 1, b.length-2, 10 ) );
        System.out.println( metodo( c, 0, c.length-1,  6 ) );
    }

    public static boolean metodo( int[] v, int ini, int fin, int x ) {
        if ( ini > fin ) return true;
        else
            if ( ini == fin )
                return (2*v[ini])==x;
            else if ( v[ini]+v[fin] != x )
                    return false;
                 else
                    return metodo( v, ini+1, fin-1, x );
    }
}
```

3. (2 puntos) Dado el siguiente método:

```
/** n>=0 */
public static void binario( int n ) {
    if ( n > 0 ) binario( n/2 );
    System.out.print( n % 2 );
}
```

Se pide:

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

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

c) Escribe 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. Resuélvela por sustitución.

d) Expresa el resultado anterior usando notación asintótica.

Solución:

1. La talla del problema es el valor del argumento del método, esto es, `n`.
2. No hay instancias significativas.
3. Planteamos la ecuación de recurrencia: $T(n) = \begin{cases} T(n/2) + k & \text{si } n > 0 \\ k' & \text{si } n = 0 \end{cases}$

   Resolviendo por sustitución: $T(n) = T(n/2)+k = T(n/2^2)+2k = \ldots = T(n/2^i)+ik$. Al llegar al caso base $i \approx log_2 n$ y se cumple que $T(n/2^i) \to T(0) = k'$. Con lo que $T(n) \approx k' + klog_2 n$.
4. En notación asintótica: $T(n) \in \Theta(\log n)$, es decir, el coste temporal es logarítmico con la talla del problema.

4. (3 puntos) Sea una matriz cuadrada de enteros, `m`, con todos sus valores no negativos. Para comprobar si los elementos de su diagonal principal suman más que cierto valor `val`, no negativo, se consideran los dos métodos siguientes:

```
public class Problema4 {
    /** Para todo i, j: 0<=i<m.length, 0<=j<m.length, m[i][j]>=0 y val>=0 */
    public static boolean metodo1( int[][] m, int val ) {
        int s = 0;
        for( int i=0; i < m.length; i++ ) s += m[i][i];
        return s > val;
    }

    /** Para todo i, j: 0<=i<m.length, 0<=j<m.length, m[i][j]>=0 y val>=0 */
    public static boolean metodo2( int[][] m, int val ) {
        int s = 0;
        for( int i=0; i < m.length && s <= val; i++ ) {
            for( int j=0; j <= i && s <= val; j++ ) {
                if ( i == j ) s += m[i][j];
            }
        }
        return s > val;
    }
}
```

Se pide:

a) **Para cada método:**

1. Indica cuál es el tamaño o talla del problema, así como la expresión que lo representa.
2. Identifica, caso de que las hubiere, las instancias del problema que representan el caso mejor y peor del algoritmo.
3. Elige una unidad de medida para la estimación del coste (pasos de programa, instrucción crítica) y acorde con ella, obtén una expresión matemática, lo más precisa posible, del coste temporal del programa, a nivel global o en las instancias más significativas si las hay.
4. Expresa el resultado anterior utilizando notación asintótica.

b) Describe brevemente las diferencias entre ambos métodos.

c) ¿Cómo modificarías el primero parar mejorarlo?

Solución:

a) Para cada método:

1. En ambos métodos, la talla o tamaño del problema es el número de filas de la matriz, `m.length`, que a su vez coincide con el de columnas. En adelante, llamaremos a ese número n, es decir, $n =$ `m.length`.
2. El primer método es un problema de recorrido, por tanto, para una misma talla del problema no hay instancias significativas. El segundo método es un problema de búsqueda y, por tanto, para una misma talla sí que presenta instancias distintas. El *caso mejor* se da cuando el primer elemento de la matriz es mayor que `val`, es decir, `m[0][0]>val`. El *caso peor* ocurre cuando la suma de todos los elementos de la diagonal principal de la matriz es menor o igual que `val`.

- Si optamos por elegir como unidad de medida el paso de programa, tendremos:
  En el primer método, como función de coste: $T(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ pasos.
  Para el segundo método, en su *caso mejor*, cuando sólo se evalúa una única vez la guarda del bucle más interno, podemos decir que $T^m(n) = 1$ paso, mientras que en su *caso peor*, se tendría que: $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i} 1) = 1 + \sum_{i=0}^{n-1}(i+2) = (n^2+3n)/2+1$ pasos.
- Si, por lo contrario, se opta por elegir la instrucción crítica como unidad de medida para la estimación del coste, entonces:
  En el primer método, si se selecciona la asignación `s += m[i][i]`, obtenemos la siguiente función de coste temporal: $T(n) = \sum_{i=0}^{n-1} 1 = n$.
  En el segundo método, consideramos como instrucción crítica la comparación: `(i == j)`. 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 el número máximo de veces posible y la función de coste será $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(i+1) = (n^2+n)/2$.

Para el primer método, en notación asintótica, $T(n) \in \Theta(n)$, es decir, el coste temporal es lineal con la talla del problema.

Para el segundo método, en notación asintótica, $T^m(n) \in \Theta(1)$ y $T^p(n) \in \Theta(n^2)$. Por tanto, $T(n) \in \Omega(1)$ y $T(n) \in O(n^2)$, es decir, el coste temporal está acotado inferiormente por una función constante y superiormente por una función cuadrática con la talla del problema.

b) Para determinar si los elementos de la diagonal de la matriz cuadrada `m` suman más que `val`, el primer método únicamente visita dichos elementos mientras que el segundo método visita también todos los elementos por debajo de la diagonal. El primer método, al tratarse de un recorrido, realiza la suma de todos los elementos de la diagonal. Sin embargo, el segundo método, al tratarse de una búsqueda, en cuanto la suma de los elementos de la diagonal visitados en un momento dado es mayor que `val`, ya no sigue sumando.

c) El primer método se puede mejorar si se resuelve como una búsqueda, sin más que cambiar la guarda del bucle por `i < m.length && s <= val`.

5. (1 punto) La siguiente tabla muestra los tiempos de ejecución, en milisegundos, de cierto algoritmo para los valores de la talla del problema que se muestran en la primera columna.

```
#    Talla    Tiempo (ms)
#---------------------------------------
      1000       49.78
      2000      202.33
      3000      454.42
      4000      804.03
      5000     1270.28
      6000     1841.47
      7000     2506.30
      8000     3253.62
      9000     4141.05
     10000     5277.99
```

a) Desde un punto de vista asintótico, ¿cuál crees que es la función de coste temporal que mejor aproxima los valores de la columna `Tiempo`?

b) De forma aproximada, ¿cuánto tiempo crees que tardará en ejecutarse el algoritmo anterior para una talla de `20000` elementos?

Solución:

La función de coste temporal que más se aproxima a los valores de la tabla es una función cuadrática con la talla del problema. Si llamamos n a dicha talla, $T(n) \in \Theta(n^2)$. Se puede comprobar que cuando la talla se duplica (de n a $2n$), el tiempo se multiplica por 4 (de n^2 a 2^2n^2). Por ejemplo, para $n = 2000$ el tiempo es 202.33 ms y para $n = 4000$ el tiempo es 804.03 ms, aproximadamente 4 veces el de $n = 2000$.

Para una talla de `20000` elementos, el tiempo de ejecución sería aproximadamente $4 \times 5277.99 \approx 20000$ milisegundos.