PRG - ETSInf. TEORIA. Curs 2015-16. Recuperació Parcial 1.
17 de juny de 2016. Durada: 2 hores.

1. [4 punts] Donat un array `a` de `int` on `a.length` $\geq$ `1`, escriure un mètode **recursiu** que determine si els elements de l'array representen una successió de Fibonacci, és a dir, si el valor d'un element correspon a la suma dels dos elements immediatament anteriors (sent els dos primers elements 0 i 1), tal com es mostra en el següent exemple: {`0,1,1,2,3,5,8,13,21,...`}.

   Es demana:

   a) (0.75 punts) Perfil del mètode, amb els paràmetres adequats per a resoldre recursivamente el problema.

   b) (1.25 punts) Cas base i cas general.

   c) (1.5 punts) Implementació en Java.

   d) (0.5 punts) Crida inicial per a que determine si es compleix la successió sobre tot l'array.

   Solució:

   a) Una possible solució consisteix a definir un mètode amb el següent perfil:

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

   de manera que determine si els elements de `a[0..pos]` representen una successió de Fibonacci, sent `0` $\leq$ `pos` $\leq$ `a.length - 1`.

   b)
   - Cas base, `pos = 0`: Verifica que `a[0] = 0`.
   - Cas base, `pos = 1`: Verifica que `a[1] = 1` i que `a[0] = 0`.
   - Cas general, `pos` $>$ `1`: Subarray de tres o més elements. Verifica que `a[pos] = a[pos - 1] + a[pos - 2]`.

   c)
   ```
   /** Determina si els elements de a[0..pos] representen una successió de Fibonacci.
    *  Precondició: a.length >= 1 i 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) Per a un array `a`, la crida `esFibo(a, a.length - 1)` resol el problema de l'enunciat.

2. [3 punts] Mitjançant el següent mètode es comprova si totes les files de la matriu `m`, quadrada, de `double`s, sumen sempre un mateix valor. Se sap que `m` és almenys d'ordre 2 (té un nombre de files i columnes major o igual que 2).

   ```
   /** PRECONDICIÓ: m és quadrada d'ordre major o igual que 2. */
   public static boolean sumenIgual(double[][] m) {
       // Es 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;
       // Es 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;
   }
   ```

   Es demana:

   a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

   b) (0.75 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (1.50 punts) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és la dimensió de la matriu `m`, és a dir, `m.length`. D'ara endavant, anomenarem a aquest nombre n. Açò és, n=`m.length`.

b) Si que existeixen diferents instàncies ja que es tracta de la cerca de la primera fila de la matriu `m`, si existeix, que la seua suma siga diferent de la de la primera.

El cas millor és quan la suma dels elements de la primera fila de `m` és diferent de la suma dels de la segona (amb la qual cosa es recorreran solament els elements de la primera i segona files), mentre que el cas pitjor es donarà quan la suma dels elements de totes les files de la matriu `m` val el mateix (i es recorreran per complet totes les files).

c) Triant com a unitat de mesura el pas de programa, es té:

- Cas millor:

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

- Cas pitjor:

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

Triant com a unitat de mesura la instrucció crítica i considerant com tal la instrucció `sum0 += m[0][i];` del primer bucle i la instrucció `sumFil += m[i][j];` del segon bucle (ambdues de cost unitari), es té:

- Cas millor:

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

- Cas pitjor:

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

d) En notació asimptòtica: $T^m(n) \in \Theta(n)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(n)$ i $T(n) \in O(n^2)$.

3. 3 punts Donada la següent implementació d'un algorisme recursiu que resol la multiplicació *a la russa* de dos nombres naturals `a` i `b`:

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

Es demana: analitzar el cost temporal d'aquest algorisme. En concret:

a) (0.25 punts) Indicar quina és la talla del problema, i quina expressió la defineix.

b) (0.5 punts) Identificar, cas que les hi haguera, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

c) (1.5 punts) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, de l'equació de recurrència del cost temporal en funció de la talla (una única equació si no hi haguera instàncies significatives; dues equacions, corresponents als casos millor i pitjor, en el cas que el problema tinguera instàncies significatives). Resoldre-la(les) per substitució.

d) (0.75 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla és el valor del segon argument, l'enter `b`, que anomenarem n d'ara endavant. L'enter `a`, no influeix en el nombre de crides recursives que es generen, ni en el cost temporal de les operacions no recursives, per la qual cosa no té cap efecte en el cost de l'algorisme.

b) Atès que es tracta d'un algorisme que *recorre* la seqüència de valors $b, \frac{b}{2}, \frac{b}{4}, \frac{b}{8} \ldots$ no existeixen instàncies significatives.

c) El cost de les operacions que s'executen en el cas base ($n = 0$) és constant, suposem que igual a $c_0 \in \Re^+$. El cost de les operacions no recursives que s'executen en el cas general ($n > 0$) és també constant, suposem que igual a $c_1 \in \Re^+$. En el cas general es realitza una crida recursiva en la qual la talla del problema (n) es redueix a la meitat. En conseqüència, es pot escriure la següent equació de recurrència:

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

expressada en passos de programa (*p.p.*).

Resolent per substitució: $T(n) = c_1 + T(\frac{n}{2^1}) = 2c_1 + T(\frac{n}{2^2}) = 3c_1 + T(\frac{n}{2^3}) = \ldots = kc_1 + T(\frac{n}{2^k})$. En el cas base, la talla és 0, i el paràmetre de la funció T hauria de prendre aquest valor, donant lloc a la igualtat: $\frac{n}{2^k} = 0$. Per a poder trobar el valor de k, canviem aquesta igualtat a $\frac{n}{2^k} = 1$ i obtenim el valor de $k = \log_2 n$. Substituint el valor de k en el terme general $kc_1 + T(\frac{n}{2^k})$, obtenim:

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

d) En notació asimptòtica: $T(n) \in \Theta(\log n)$, és a dir, el cost temporal de l'algorisme depèn logarítmicament de la talla del problema.