Primer Parcial de PRG - ETSInf
Data: 7 de maig de 2012. Duració: 2 hores i 30 minuts

NOTA: Cal contestar en fulles apart. No és necessari entregar aquesta fulla.

1. (2 punts) Escriu un mètode **recursiu** que reba com argument un valor enter no negatiu i escriga en l'eixida estàndard les seqüències descendent i ascendent de valors enters des del número inicial fins el zero. Cada seqüència en una línia diferent.

   Per exemple, si el mètode sol.licitat s'executa amb el valor 14, aleshores escriurà per l'eixida estàndard el següent:

   ```
   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 els números de cada seqüència apareixen tots seguits en una mateixa línia.

Solució:

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

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

2. (2 punts) Escriu un mètode **recursiu** que donat un array d'enters `v`, una posició inicial `ini>=0`, una posició final `fi<v.length` i un valor enter `x` determine si la suma dels elements simètrics és `x`, és a dir, els extrems dels subarrays formats per les posicions en l'interval `[ini+i, fi-i]` per a `0<=i<=(ini+fi)/2` compleixen que `v[ini+i]+v[fi-i] == x`.

   Per exemple:

   Si `x = 10, ini = 1, fi = v.length-2` i `v = {1,2,3,4,5,6,7,8,9}` retorna `true`.

   Si `x = 10, ini = 1, fi = v.length-2` i `v = {1,2,4,4,5,6,7,8,9}` retorna `false`.

   Si `x = 6, ini = 0, fi = v.length-1` i `v = {1,2,3,3,4,5}` retorna `true`.

Solució:

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

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

3. (2 punts) Donat el següent mètode:

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

Es demana:

a) Indica quina és la talla del problema i quina expressió la defineix.

b) Determina si existeixen instàncies significatives. Si n'hi ha, identifica les que representen els casos millor i pitjor de l'algorisme.

c) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

d) Expressa el resultat anterior fent servir notació asimptòtica.

Solució:

1. La talla del problema és el valor de l'argument del mètode, açò és, `n`.
2. No hi ha instàncies significatives.
3. Plantegem l'equació de recurrència: $T(n) = \begin{cases} T(n/2)+k & \text{si } n > 0 \\ k' & \text{si } n = 0 \end{cases}$

   Resolent per substitució: $T(n) = T(n/2)+k = T(n/2^2)+2k = \ldots = T(n/2^i)+ik$. En arribar al cas base $i \approx log_2 n$ i es compleix que $T(n/2^i) \to T(0) = k'$. Amb el que $T(n) \approx k' + klog_2 n$.
4. En notació asimptòtica: $T(n) \in \Theta(\log n)$, és a dir, el cost temporal és logarítmic amb la talla del problema.

4. (3 punts) Siga una matriu quadrada d'enters, `m`, amb tots els seus valors no negatius. Per tal de comprovar si els elements de la seua diagonal principal sumen més que cert valor `val`, no negatiu, es consideren els dos mètodes següents:

```
public class Problema4 {
    /** Per a tot i, j: 0<=i<m.length, 0<=j<m.length, m[i][j]>=0 i val>=0 */
    public static boolean metode1( int[][] m, int val ) {
        int s = 0;
        for( int i=0; i < m.length; i++ ) s += m[i][i];
        return s > val;
    }

    /** Per a tot i, j: 0<=i<m.length, 0<=j<m.length, m[i][j]>=0 i val>=0 */
    public static boolean metode2( 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;
    }
}
```

Es demana:

a) **Per a cada mètode:**

1. Indica quina és la grandària o talla del problema, així com l'expressió que la representa.
2. Identifica, en cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.
3. Tria una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella, obté una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si les hi ha.
4. Expressa el resultat anterior utilitzant notació asimptòtica.

b) Descriu breument les diferències entre ambdós mètodes.

c) Com modificaries el primer per tal de millorar-lo?

Solució:

a) Per a cada mètode:

1. En ambdós mètodes, la talla o grandària del problema és el número de files de la matriu, `m.length`, que al mateix temps coincideix amb el de columnes. D'ara endavant, anomenarem a aquest nombre n. És a dir, $n =$ `m.length`.
2. El primer mètode és un problema de recorregut, per tant, per a una mateixa talla del problema no hi ha instàncies significatives. El segon mètode és un problema de cerca i, per tant, per a una mateixa talla sí que presenta instàncies distintes. El *cas millor* es dóna quan el primer element de la matriu és major que `val`, és a dir, `m[0][0]>val`. El *cas pitjor* ocorre quan la suma de tots els elements de la diagonal principal de la matriu és menor o igual que `val`.

3. • Si optem per triar com unitat de mesura el pas de programa, s'obté:
Per al primer mètode, com a funció de cost temporal: $T(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ passos.
Per al segon mètode, en el seu *cas millor*, quan tan sols s'avalua una única vegada la guarda del bucle més intern, podem dir que $T^m(n) = 1$ pas, mentre que per al *cas pitjor* es tindria: $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i} 1) = 1 + \sum_{i=0}^{n-1}(i + 2) = (n^2 + 3n)/2 + 1$ passos.

   • Si optem per triar la instrucció crítica com unitat de mesura per a l'estimació del cost, es tindria que:
En el primer mètode, si es selecciona l'assignació `s += m[i][i]`, obtenim la següent funció de cost temporal: $T(n) = \sum_{i=0}^{n-1} 1 = n$.
En el segon mètode, considerem com instrucció crítica la comparació: `(i == j)`. En el *cas millor* només s'executarà una vegada i la funció de cost temporal en aquest cas serà $T^m(n) = 1$. En el *cas pitjor* es repetirà el número màxim de vegades possible i la funció de cost serà $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(i + 1) = (n^2 + n)/2$.

4. Per al primer mètode, en notació asimptòtica, $T(n) \in \Theta(n)$, és a dir, el cost temporal és lineal amb la talla del problema.

   Per al segon mètode, en notació asimptòtica, $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n^2)$, és a dir, el cost temporal està acotat inferiorment per una funció constant i superiorment per una funció quadràtica amb la talla del problema.

b) Per tal de determinar si els elements de la diagonal de la matriu quadrada `m` sumen més que `val`, el primer mètode únicament visita aquests elements mentre que el segon mètode visita també tots els elements per baix de la diagonal. El primer mètode, al tractar-se d'un un recorregut, realitza la suma de tots els elements de la diagonal. No obstant, el segon mètode, al tractar-se d'una cerca, quan la suma dels elements de la diagonal visitats en un moment donat és major que `val`, ja no segueix sumant.

c) El primer mètode es pot millorar si es resol com una cerca, només canviant la guarda del bucle per `i < m.length && s <= val`.

5. (1 punt) La taula següent mostra els temps d'execució, en mil·lisegons, de cert algorisme per als valors de la talla del problema que es mostren en la primera columna.

```
#    Talla    Temps (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) Des d'un punt de vista asimptòtic, quina creus que és la funció de cost temporal que millor aproxima els valors de la columna `Temps`?

b) De forma aproximada, quant de temps creus que tardaria l'algorisme anterior en executar-se per a una talla de `20000` elements?

Solució:

La funció de cost temporal que més s'aproxima als valors de la taula és una funció quadràtica amb la talla del problema. Si anomenem n a aquesta talla, $T(n) \in \Theta(n^2)$. Es pot comprovar que quan la talla es duplica (de n a $2n$), el temps es multiplica per 4 (de n^2 a 2^2n^2). Per exemple, per a $n = 2000$ el temps és 202.33 ms i per a $n = 4000$ el temps és 804.03 ms, aproximadament 4 vegades el de $n = 2000$.

Per a una talla de `20000` elements, el temps d'execució seria aproximadament $4 \times 5277.99 \approx 20000$ mil·lisegons.