PRG - ETSInf. TEORIA. Curs 2012-13. Parcial 1. Recuperació
17 de juny de 2013. Duració: 1h 50min.

1. 2.5 punts Per a determinar si cert nombre enter no negatiu `n` pot estar expressat en una base determinada `b` ($2 \leq b \leq 10$), és suficient que tots els dígits del nombre tinguen un valor estrictament menor que la base `b`.

   Per exemple, el nombre 453123 pot representar un valor en base 6, 7, 8, 9 i 10 ja que tots els seus dígits són estrictament inferiors als valors d'aquestes possibles bases.

   Es demana: Implementar en Java un **mètode recursiu** que, donats `n` i `b`, resolga el problema plantejat, especificant els casos base i general de la recursió.

   Solució:

```
/** Torna si és o no possible que n estiga en base b.
  * PRECONDICIÓ: 2<=b<=10 i n >=0 */
public static boolean basePossible(int n, int b) {
    if (n==0) return true;                              // cas base
    else {                                              // cas general
        int ultDig = n%10;
        if (ultDig<b) return basePossible(n/10,b);
        else return false;
    }
}
```

2. 2.0 punts Donat un array `a` de valors reals i cert valor real `x`, es vol determinar, **recursivament**, el nombre d'elements d'`a`, des de la posició `pos` inclosa fins a la final (`a[pos..a.length-1]`), que tinguen valor més gran que x. Per a això, partint del perfil:

   `public static int numMajors(double[] a, double x, int pos)`

   Es demana:

   - Implementar el **mètode recursiu** demanat, especificant els casos base i general de la recursió.
   - Escriure la crida inicial per obtenir el nombre d'elements més grans que `x` de tot un array.

   Solució:

```
/**
  * Calcula el nombre d'elements amb valor major que un donat des de
  * la posició pos.
  */
public static int numMajors(double[] a, double x, int pos) {
    if (pos>=a.length) return 0;                        // cas base
    else if(a[pos]>x) return 1+numMajors(a,x,pos+1);    // cas general
    else return numMajors(a,x,pos+1);                   // cas general
}
```

   Crida inicial: `int num=numMajors(v,x,0);` complint `v` i `x` les condicions donades per la capçalera del mètode.

3. [3.0 punts] Donada certa matriu `m` quadrada de valors reals, el següent mètode determina si aquesta és triangular inferior (és a dir, si tots els elements superiors a la diagonal principal són iguals a 0):

```
/** Determina si una matriu quadrada és triangular inferior, això és:
  * si tots els elements superiors a la diagonal principal valen 0.
  * /
public static boolean esInferior(double[][] m) {
    boolean esInf = true;
    for(int i=0; i<m.length && esInf; i++)
        for(int j=i+1; j<m.length && esInf; j++)
            esInf = m[i][j]==0;
    return esInf;
}
```

Es demana: estudiar el cost temporal del mètode, per el que has de:

a) Indicar quina és la mida o talla del problema, així com l'expressió que la representa.

> **Solució:** La talla del problema és n=`m.length`, és a dir, la dimensió de la matriu.

b) Identificar, cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

> **Solució:** El mètode és un problema de recerca i, per tant, per a una mateixa talla sí que presenta instàncies diferents.
> *Cas millor:* Quan `m[0][1]!=0` amb el que ambdós bucles finalitzaran en acabar la seva primera iteració. *Cas pitjor:* Quan la matriu és triangular inferior i tots els elements superiors a la diagonal principal valen 0. En aquest cas, s'efectuen totes les iteracions possibles.

c) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i acord amb ella, obtenir una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si n'hi ha.

> **Solució:**
>
> - Si optem per escollir com a unitat de mesura el pas de programa, s'obté:
>   - *Cas millor:* $T^m(n) = 1$ pas de programa.
>   - *Cas pitjor:* $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=i+1}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(n-i) = n^2/2 + n/2 + 1$ passos.
> - Si optem per escollir la instrucció crítica com a unitat de mesura per a l'estimació del cost:
>   - *Caso mejor:* $T^m(n) = 1$ instrucció crítica.
>   - *Caso peor:* $T^p(n) = \sum_{i=0}^{n-1}(\sum_{j=i+1}^{n-1} 1) = \sum_{i=0}^{n-1}(n-i-1) = n^2/2 - n/2$ instruccions crítiques.

d) Expressar el resultat anterior utilitzant notació asimptòtica.

> **Solució:**
> 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 $T(n) \in O(n^2)$.

4. 2.5 punts El següent mètode recursiu comprova si dos `String` que tenen la mateixa longitud són simètriques. Dues `String` són simètriques quan el primer element de la primera és igual a l'últim de la segona i així successivament.

   Per exemple, les `String`: `"HOLA"` i `"ALOH"` són simètriques, mentre que `"HOLA"` i `" ALHA"` no ho són.

```
/**
  * Determina si a i b són o no simètriques
  * PRECONDICIÓ: a.length()==b.length()
  */
public static boolean simetriques(String a, String b) {
    if(a.length()==0) return true;
    else {
        int ult = b.length()-1;
        return a.charAt(0)==b.charAt(ult) &&
               simetriques(a.substring(1), b.substring(0,ult));
    }
}
```

   Es demana: estudiar el cost temporal del mètode anterior, sabent que **totes les operacions de la classe** `String` que s'apliquen a alguna `String` en l'algorisme **tenen un cost constant** (això és, el seu cost no depèn de la llargària de la `String` a la qual s'aplique) per a això:

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

   > **Solució:** La talla és la longitud `n` de cada `String` (`a.length()`).

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

   > **Solució:** Sí que hi ha instàncies significatives. El *cas millor* es dóna quan el primer i últim caràcter d'ambdues `String` no concorden. El *cas pitjor* es dóna quan ambdues `String` són simètriques.

   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. Resol-la per substitució.

   > **Solució:**
   >
   > - Cas millor: $T^m(n) = 1$ passos.
   > - Cas pitjor: Es dóna el següent: $T^p(n) = 1$ si $n = 0$ i $T^p(n) = T^p(n-1) + 1$ si $n > 0$.
   >
   >   Resolent per substitució: $T^p(n) = T^p(n-1) + 1 = T^p(n-2) + 2 = \ldots = T^p(n-i) + i = \ldots = T^p(0) + n = n + 1$ passos.

   d) Expressa el resultat anterior usant notació asimptòtica.

   > **Solució:**
   > En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$.