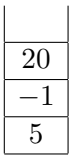
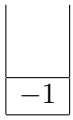
PRG - ETSInf. TEORIA. Curs 2014-15. Recuperació.
23 de juny de 2015. Duració: 2 hores i 15 minuts.

1. 2.5 punts Donada una `PilaIntEnla p` qualsevol, que pot estar buida, es vol escriure un mètode estàtic **recursiu** (en una classe distinta de `PilaIntEnla`), tal que elimine els elements positius de la pila, deixant la resta en el mateix ordre en què es trobaven inicialment. El mètode ha de retornar la suma dels elements eliminats.

   Exemple:

   La pila  ha de quedar  , i s'ha de retornar 25.

   Es demana:

   a) Perfil del mètode.

   b) Cas base i cas general.

   c) Implementació en Java.

Solució:

a)
```
/** Elimina els elements positius de p, sent p.talla() >= 0,
  * i torna la seua suma.*/
public static int filtrarPos(PilaIntEnla p)
```

b)
- Cas base, `p` està buida: No hi ha cap element que eliminar. Trivialment, se torna 0 com suma dels elements eliminats.
- Cas general, `p` no està buida, té al menys un element: Es redueix el problema desempilant el cim de `p`. Recursivament, s'eliminen els elements positius de la pila reduïda, obtenint-se la suma d'aquests elements. Si l'element que s'havia desempilat és ≤ 0 se torna a empilar en `p`, i si no el seu valor incrementa la suma a tornar.

c)
```
public static int filtrarPos(PilaIntEnla p) {
    if (p.esBuida()) return 0;
    else {
        int x = p.desempilar();
        int sumaResult = filtrarPos(p);
        if (x <= 0) p.empilar(x)
        else sumaResult += x;
        return sumaResult;
    }
}
```

2. 3 punts El següent mètode rep un array `a` els components del qual són llistes amb PI d'enters, i busca si l'enter `x` apareix en alguna d'aquestes llistes:

```
/** Busca en quina llista a[i], 0 <= i < a.length, apareix x.
 *  Si no se troba en cap, torna -1.
 */
public static int buscar(LlistaPIIntEnla[] a,int x) {
    boolean trobat = false;
    int i = 0;
    while (i < a.length && !trobat) {
        LlistaPIIntEnla l = a[i];
        l.inici();
```

```
            while (!l.esFi() && l.recuperar() != x)
                l.seguent();
            if (!l.esFi()) trobat = true; else i++;
        }
        if (trobat) return i; else return -1;
    }
```

Es desitja analitzar el cost temporal del mètode, tenint en compte que totes les operacions de la classe `LlistaPIIntEnla` són $\Theta(1)$. Per simplificar, suposarem que totes les llistes d'`a` tenen `a.length` elements.

Es demana:

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

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

c) 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 si és el cas.

d) Expressar el resultat anterior utilitzant notació assimptòtica.

Solució:

a) La talla és $n =$ `a.length`, que determina el volum d'elements a tractar en el problema.

b) Es tracta d'un problema de cerca, llista a llista, i en cada llista, element a element. Per tant, per a una mateixa talla sí que presenta instàncies distintes.

El cas millor es dóna quan `x` es troba en la primera posició de la primera llista revisada, `a[0]`.

El cas pitjor quan tots els elements de totes les llistes components d'`a` són diferents de `x`.

c) Mesura del cost prenent com unitat de mesura el pas de programa:

Totes les instruccions que s'executen en el mètode (excepte els bucles) tenen un cost de l'ordre d'un pas de programa. Així doncs, el cost del cas millor és $T^m(n) = 1$ *p.p.* El cost del cas pitjor és $T^p(n) = 1 + n \cdot (1 + n) = 1 + n + n^2$ *p.p.*, donat que en aquest cas el bucle més extern realitza n passades, totes elles del mateix cost $1 + n$ *p.p.*

Mesura del cost prenent com unitat de mesura la instrucció crítica:

Totes les instruccions que s'executen en el mètode (excepte els bucles) tenen un cost del mateix ordre constant, per tant, es pot prendre com a instrucció crítica l'avaluació de la guarda `!l.esFi() && l.recuperar() != x`, que és la que més es repeteix. En el cas millor només s'executarà una vegada. En el cas pitjor es repetirà, per a cadascuna de les n llistes, un total de $n + 1$ vegades (tantes vegades com elements hi ha en la llista, més l'avaluació final que detecta el final de la llista). La funció de cost temporal, considerant la instrucció crítica de cost unitari, en el cas millor serà $T^m(n) = 1$ *i.c.* i en el cas pitjor $T^p(n) = n \cdot (n + 1) = n + n^2$ *i.c.*

d) En notació assimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n^2)$.

3. 1.5 punts Es defineix l'índex de massa corporal d'una persona (IMC) com el seu pes (kg) dividit por l'altura (m) al quadrat. Se disposa d'un fitxer de text en el que cada línia conté el dni (una cadena de caràcters), pes i altura d'una persona (ambdós nombres reals), separats per espais en blanc.

Es demana: Implementar un mètode que, a partir d'aquest fitxer, genere un altre en què aparega, a més de les dades anteriors, una quarta columna amb l'IMC. Tant el nom del fitxer origen, com el nom que es vol per al fitxer destí hauran de ser paràmetres (de tipus `String`) del mètode.

Suposarem que el format de les dades del fitxer d'entrada és correcte. En cas que hi haja algun problema en obrir els fitxers, s'ha d'escriure en l'eixida estàndard el missatge `"Error obrint fitxer"`.

Solució:

```
public static void imc(String fitxIn, String fitxOut) {
    try {
        Scanner sc = new Scanner(new File(fitxIn)).useLocale(Locale.US);
        PrintWriter pw = new PrintWriter(new File(fitxOut));
        while (sc.hasNext()) {
            String dni = sc.next();
            double pes = sc.nextDouble();
            double altura = sc.nextDouble();
            double imc = pes / (altura * altura);
            pw.println(dni + " " + pes + " " + altura + " " + imc);
        }
        sc.close();
        pw.close();
    } catch (FileNotFoundException e) {
        System.out.println("Error obrint fitxer");
    }
}
```

4. [3 punts] **Es demana:** En la classe `LlistaPIIntEnla`, implementar un mètode d'instància amb perfil:

```
public void esborrarEnrere()
```

que, amb un cost com a molt lineal, permeta l'esborrament cap enrere, açò és, que esborre l'element anterior al del punt d'interès.

En cas que la llista estiga buida o que, en general, el punt d'interès es trobe a l'inici, haurà de llançar l'excepció `NoSuchElementException` amb el missstge `"PI a l'inici"`.

En cas contrari, després de l'esborrament, el punt d'interès (`PI`) continua inalterat sobre el mateix element, i l'anterior al punt d'interès (`antPI`) retrocedeix una posició.

Exemple: Si la llista és `2 3 1 8 9` (el `PI` està sobre el 8), després de l'esborrament ha de quedar `2 3 8 9`.

NOTA: En la solució sols es permet accedir als atributs de la classe, quedant terminantment prohibit l'accès als seus mètodes.

Solució:

```
/** Elimina l'element anterior al punt d'interès. Si la llista està buida o el
 *  punt d'interès està a l'inici, llança NoSuchElementException. */
 public void esborrarEnrere() {
    if (antPI == null) throw new NoSuchElementException("PI a l'inici");
    if (antPI == primer) {
        primer = antPI.seguent;
        antPI = null;
    }
    else {
        NodeInt aux = primer;
        while (aux.seguent != antPI)
            aux = aux.seguent;
        aux.seguent = antPI.seguent;
        antPI = aux;
    }
    talla--;
 }
```

Notar que el cost depèn, a més de la talla n de la llista, de la posició del `PI`. En el cas pitjor, quan el `PI` està al final de la llista, $T^p(n) \in \Theta(n)$.