RESUM DE RECURSIÓ EN JAVA (C.Herrero, grup 1A PRG 2020)

DEFINICIÓ D'ALGORISME: és un conjunt prescrit d'instruccions o regles ben definides, ordenades i finites que permet dur a terme una activitat mitjançant passos successius que no generen dubtes a qui haja de fer aquesta activitat i que daus: estat inicial i dades d'entrada, seguint els passos successius s'arriba a un estat final i a una solució o eixida.

DEFINICIÓ D'ALGORISME RECURSIU: algorisme (**i per tant finit**) que expressa la solució d'un problema en termes d'una invocació a ell mateix.

FUNCIONAMENT INTERN DE LA RECURSIO: Els algorismes recursius solen anar situats dins de mètodes per a permetre les denominades cridades recursives. Cada crida a un mètode (siga o no recursiu) reserva, en una zona especial de memòria, una porció de la mateixa denominada "Registre d'Activació" que conté:

- l'adreça on el comptador de programa ha de tornar en acabar el mètode,
- la còpia per valor dels paràmetres del mètode com a variables locals,
- les variables que es defineixen dins del mètode explícitament,
- el valor de tornada (si ho hi ha) que prendrà el mètode en fer el `return`.

En la recursió, les successives crides al mètode des de dins d'ell mateix, generen l'aparició de nous registres d'activació quasi idèntics (però amb diferents valors interns) que s'apilen en la denominada "Pila de Recursió". Així, cada crida recursiva utilitza variables noves ("fresques") sense utilitzar les de la crida o crides anteriors. Quan una crida a mètode finalitza el registre d'activació associat desapareix, no sense abans copiar el resultat (si ho hi ha) en el lloc adequat de la crida prèvia.

A diferència dels algorismes iteratius, el consum de memòria en els algorismes recursius és significatiu i ha de ser tingut en compte.

ETAPES DEL DISSENY D'UN MÈTODE RECURSIU:

- enunciat del problema (Capçalera, paràmetres, precondició i resultat esperat),
- anàlisi per casos (identificar cas base i cas general i comprovar que son complementaris i excloents),
- transcripció algorísmica (implementació en Java del mètode),
- Validació del disseny (comprovació de la terminació i correcció)

GARANTIA DE TERMINACIÓ D'UN ALGORISME RECURSIU: Tot algorisme ha de finalitzar i per tant un algorisme recursiu no pot resoldre's infinitament invocant a casos més xicotets. Sempre ha d'haver-hi un cas més xicotet que tots els altres. Per a identificar-ho primer s'ha de concretar què és el que defineix la grandària del problema. Una vegada estiga definit aquest s'ha de trobar el cas o els casos més xicotets (CAS BASE o DIRECTE) para el que no siga necessari cridar a altres problemes més xicotets (al contrari que la resta de casos, que correspondrien al CAS GENERAL o RECURSIU) doncs la seua solució és calculable directament. Cada invocació d'un algorisme recursiu ha de garantir que es realitza sobre un problema més xicotet, amb el que, ineludiblement, arribe al mínim en algun moment.

GARANTIA DE CORRECCIÓ D'UN ALGORISME RECURSIU: La correcció de l'algorisme, és a dir, que per a qualssevol dades de entrada vàlides que es donen, la solució de l'algorisme és correcta (sol requerir una demostració matemàtica per inducció sobre algun paràmetre de l'algorisme, i demostrant que la solució del cas base es correcta).

PARTS D'UN MÈTODE RECURSIU: Encara que admeten moltes variants, tots els mètodes recursius tenen almenys aquestes components:

- Capçalera del mètode, conté com tots els mètodes:
  - tipus de tornada (pot ser *`void`*),
  - identificador del mètode,
  - paràmetres. Els mètodes recursius tenen d'especial que els paràmetres han de (de manera habitual i excepte excepcions) incloure allò que siga necessari perquè es reduïsca la grandària del problema, i que canvie de crida a crida, de manera que cada vegada estiga més prop de ser el cas base.
- Cos del mètode, a més de declaració de variables i altres elements comuns ha de contenir:
  - instrucció Condicional, que la seua guarda discrimine si es troba en el cas base o no,
  - instruccions de resolució del cas base,
  - instruccions de resolució del cas recursiu:
    - reducció del problema,
    - crida o cridades recursives al mateix mètode amb el problema reduït,
    - combinació del resultat de la crida recursiva amb les dades locals (opcional),
  - devolució del resultat de la crida al mètode (si és que no és de tipus *`void`*).

ERRORS MÉS FREQÜENTS:

- *`Stack Overflow`* (Desbordament de pila): es tracta d'un error que es produeix quan s'omple la zona de memòria de la pila de recursió per excés de registres d'activació.
- Recursió Infinita: es tracta d'un error greu que es produeix quan hi ha casos en els quals no es redueix el problema en cada crida o que el cas base

TAXONOMIA DE LA RECURSIO: Depenent de si hi ha de combinació del resultat de cada crida amb l'anterior i del nombre de crides:

Recursió Lineal Final Recursió Lineal No final Recursió Múltiple

Recursió amb *arrays*: La grandària del problema un mètode recursiu ve donada per algun dels paràmetres, i reduint aquesta grandària és com es pot fer una crida recursiva garantint que l'algoritme finalitze. Però si el paràmetre d'entrada es un *array*, i els *arrays* son de grandària constant i definida en el moment de la creació: com podem reduir la grandària del problema? Afegint paràmetres amb límits.
El recorregut i la cerca d'un array de manera recursiva es prou semblant a les versions iteratives, les diferències més obvies son, en general:

- Paràmetres afegits per a assenyalar els límits de la part tractada i la no tractada
- Absència de instrucció de iteració (bucle).
- Aparició de instrucció condicional per a distingir el cas base del cas recursiu.
- En cas de cerca, aparició de una segona instrucció condicional per a parar la recursió en cas de trobar-ho. (no en la mateixa condició que abans)
- Normalment hi ha que especificar quina es la crida inicial per a que l'algorisme siga equivalent al iteratiu
- MOLT IMPORTANT: En el cas de la cerca **NOMÉS FER LA CRIDA RECURSIVA SI NO S'HA TROBAT i, òbviament, no fer-la més vegades de les necessàries**

Exemple de recorregut d'arrays iteratius:

```
/** Precondició a.length > 0
  * Torna la posició del màxim de l'array
  */

public static int posMaxIteratiu(double[] a) {

    int posMax = 0;

    for (int i = 1; i < a.length; i++) {

        if (a[i] > a[posMax]) {

            posMax = i;

        }
    }

    return posMax;

}
```

Es compara la posició i amb la del màxim des de 0 fins a i-1

```
/** Precondició a.length > 0 ini <= a.length
  * Torna la posició del màxim de l'array des de
  * la posició ini endavant
  *      crida inicial posMaxRecursiu(a, 0)
  */
public static int posMaxRecursiu(double[] a, int ini) {

    int posMax;

    if (ini == a.length -1) { posMax = ini; }

    } else {

        posMax = posMaxRecursiu(a, ini + 1);

        if (a[ini] > a[posMax]) {posMax = ini; }

    }
    return posMax;
}
```

Es compara la posició ini amb la del màxim des de ini + 1 fins a a.length

Exemple de cerca d'arrays iteratius:

```
/** Precondició no hi ha
  *  torna la posició del primer negatiu
  *     i si no hi ha torna  -1
  */

public static int posNegIteratiu(double[] a) {

    int i = 0;

    while (i < a.length && a[i] >= 0){

        i++;

    }
    if (i == a.length) {

        i = -1;

    }

    return i;
```

En acabant si hem arribat a la fi, no ho hem trobat i tornem -1

```
/** Precondició a.length > 0 ini <= a.length
  *  torna la posició del primer negatiu des de la posició ini endavant
  *  i si no hi ha, torna -1
  *  crida inicial posNegRecursiu(a, 0)
  */
public static int posNegRecursiu(double[] a, int ini) {

    int pos;

    if (ini == a.length) { pos = -1; }

    else if (a[ini] < 0) { pos = ini; }

    else { pos = posNegRecursiu (a, ini + 1); }

    return pos;
}
```

El cas base es arribar a la fi, u per tant no ho hem trobat i tornem -1