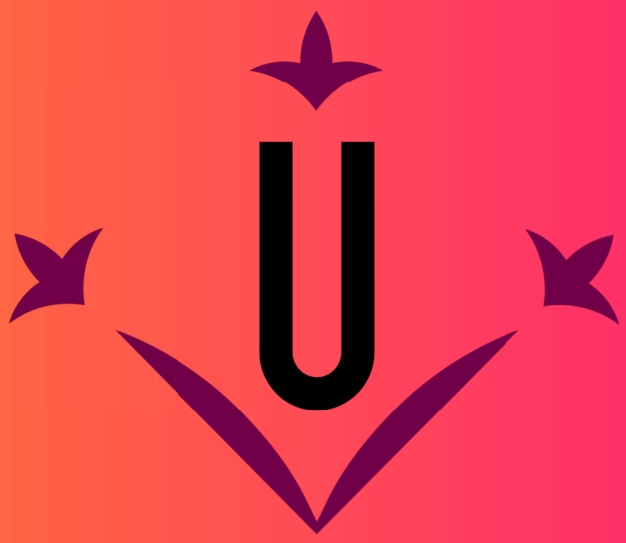
PROGRAMACIÓ II

Pràctica 1

Polinomis



Universitat de Lleida

ARNAU RODRÍGUEZ GONZÁLEZ

20 / 03 / 2021

ÍNDEX

INTRODUCCIÓ

POLINOMIS

La pràctica 1 de Programació 2 tracta sobre polinomis, diferents formes d'estructurar-los i operar amb ells.

Un polinomi és una expressió que consta de constants i variables que són operats únicament amb suma resta i de forma exponencial (els exponents només poden ser números no negatius). Podríem visualitzar un polinomi de la següent manera: una cadena de monomis encadenats per sumes i restes. Un monomi és una constant multiplicada per una o més variables on cada variable té un exponent independent.

Els polinomis se solen escriure de la forma ordenada següent: s'ordena d'esquerra a dreta els monomis amb l'exponent més gran; si té diferents variables, sempre conta la que té l'exponent més gran. En la pràctica solament operarem amb polinomis d'una sola variable.

El grau d'un polinomi és l'exponent més gran. Si tenim ordenats els monomis dins el polinomi, l'exponent més gran sempre serà a l'esquerra de tot. D'aquesta forma és molt fàcil saber de quin grau és el polinomi amb què tractem.

Exemple de monomis:

$3x^0=3$

$-5x^1=-5x$

$7x^4$

$-9x^6$

Exemple de polinomis:

$4x^1+5x^0=4x+5$

$8x^2+1x^1-3x^0=8x^2+1x-3$

$-5x^7+3x^5-2x^4$

Hem de tenir en compte que, si operem dos monomis on els dos monomis són iguals, però amb el signe canviat de la constant, el monomi resultant és zero.

$2x^8-3x^7+3x^7+5x^3=2x^8+0+5x^3=2x^8+5x^3$

REPRESENTACIÓ ESTESA D'UN POLINOMI

La representació estesa d'un polinomi és una representació que recull tots els monomis que hi ha en un polinomi, inclús els que tenen la constant igual a zero. Una 'array' tindrà *n+1* elements sent el grau del polinomi igual a *n*, ja que hem de contar el monomi de la dreta de tot, que té l'exponent igual a zero. En la 'array' es guardaran els valors, acord amb el seu exponent. En la posició *i* de la 'array' hi anirà la constant que està multiplicada per la variable amb l'exponent *i*.

Exemple representació estesa d'un polinomi:

$4x^2-5x+7$

7	-5	4

$3x^6+2x^4+1$

1	0	0	0	2	0	3

Mai tindrem elements amb zeros per la dreta, si és que no hi ha elements més endavant amb valors diferents de zero (constants). Aquesta propietat es veu molt bé si sumem dos polinomis, en què el resultat hauria de tenir zeros en les posicions finals. Sense tenir elements més endavant diferents de zero, els elements amb zeros seran eliminats.

Exemple de suma en un cas que s'ha d'eliminar zeros:

$-5x^3+2x$

0	2	0	-5

$5x^3+2x$

0	2	0	5

Resultat:

$2x$

0	2

També ens trobarem amb 'arrays' de mida igual a zero. Això vol dir que és un polinomi buit, sense cap monomi.

REPRESENTACIÓ COMPRIMIDA D'UN POLINOMI

La representació comprimida d'un polinomi és una representació que només recull les constants i els exponents quan les constants del monomi són diferents de zero. Els valors es guarden emparellats en una 'array' de 'arrays' (matrius). La 'array' principal serà de mida tan gran com el nombre de constants diferents de zero en el polinomi. I a cada element tindrà una 'array' de mida dos, on a la primera posició, hi haurà l'exponent de la variable, i a la segona posició, la constant que la multiplica. La 'array' estarà ordenada com en la forma estesa, dels exponents més petits als més grans.

Exemple representació comprimida d'un polinomi:

$-6x^4+8x^2$

2	8
4	-6

$x^{20}+2x^{10}-3x^5$

5	-3
10	2
20	1

Mai tindrem elements amb 'arrays' amb zeros a la posició u. Aquesta propietat es veu molt bé si sumem dos polinomis, en que el resultat hauria de tenir zero en la posició u de la 'array' del monomi. Els elements amb 'arrays' on a la posició u tenen un zero, són eliminats.

Exemple de suma en un cas que s'ha d'eliminar elements:

$8x^{32}-5x^{16}+4x^8$

8	-4
16	5
32	8

$3x^{24}+5x^{16}+2x^{8}$

6	2
16	5
24	3

Resultat:

$8x^{32}+3x^{24}+6x^{8}$

8	6
24	3
32	8

També ens trobarem amb ‘arrays’ de mida igual a zero. Això vol dir que és un polinomi buit, sense cap monomi.

MILLOR FORMA DE CODIFICAR UN POLINOMI

Tenint dos formes diferents de representar un polinomi la pregunta que es planteja és: Quina és la millor forma de representar un polinomi? La resposta no és sempre la mateixa, ja que hi ha polinomis que per poder ser representats de la forma estesa poden ocupar molt espai a la memòria, mentre que de forma comprimida ocupen molt poc.

ex. : $8x^{32}$

Aquest polinomi ocupa molt poc espai de forma comprimida, ja que només ocuparia un espai de dos enters, mentre que de forma estesa ocuparia 32+1 espais d'enters.

Però en canvi, hi ha polinomis que de forma estesa ocupen la meitat que de forma comprimida.

ex. : $2x^2+2x+2$

Aquest polinomi, de forma comprimida, ocupa sis espais d'enters a la memòria, mentre que de forma estesa només n'ocupa tres.

En el codi hi ha la funció 'bestType()' que a partir d'un polinomi d'entrada determina si la forma de representar-lo és la més eficient. En l'apartat 'FUNCIONS' es tracta aquesta funció amb més profunditat. No es una funció que s'utilitzi en la pràctica, peró ha sigut afegitda en cas que en un futur aquest codi fos utilitzat mes endavant, per tenir l'espai de memòria mes optimitzat i efectuar menys operacions al operar amb les 'arrays'.

FUNCIONS

Les funcions desenvolupades en el codi no totes són donades en l'enunciat. Hi ha funcions que no demanen ser creades en l'enunciat, sigui per millorar la netedat del codi, per posar a prova més a fons el codi o per optimitzar l'espai de memòria en cas que aquest codi s'utilitzés en projectes futurs.

En les funcions s'utilitza les variables constants declarades amb l'enunciat, per anar a buscar en les 'arrays' de tipus comprimit l'exponent i la constant. Així el codi es veu més clar, no hi ha zeros i uns sinó 'DEGREE' i 'COEFFICIENT' amb els valors zero i u respectivament.

expandedSize()

INTRODUCCIÓ

- La funció 'expandedSize()' retorna a partir d'un polinomi representat de forma comprimida, la mida que hauria de tenir una 'array' de tipus estès per representar el mateix polinomi.

DISSENY

- Per determinar quin hauria de ser la mida d'una 'array' de tipus estès, se segueix el protocol establert. Sabent que en l'últim índex de la 'array' 'compressed', en la posició zero hi ha el grau del polinomi, podem utilitzar aquesta informació per determinar la mida de la 'array' de tipus estès. La mida serà, el grau del polinomi més u. Ja que el grau ens diu el màxim exponent, només li hem de sumar u, per contar el monomi de l'exponent igual a zero.
- També s'ha de tenir en compte que podem rebre una 'array' buida. És per això, que abans de buscar el grau del polinomi, mirem si la 'array' 'compressed' té una mida superior a zero. En cas que no ho miréssim ens donaria error al buscar en un índex que no existeix.

CODI

- Paràmetres:
  - compressed → ‘array’ de ‘arrays’ d’enters, representa un polinomi de forma comprimida.

- Primer es mira si la mida de la ‘array’ és superior a zero, i en cas de ser-ho retorna un zero. En cas de ser superior a zero, retornem el grau del polinomi més u. El grau es troba en la posició zero en l’últim element de la ‘array’ ‘compressed’, i li sumem u pel motiu ja mencionat (sumar l’element que representa el polinomi de l'exponent zero).
- El codi consta de les línies següents:

```
if (compressed.length == 0) {
      return 0;
}
else {
      return compressed[compressed.length - 1][DEGREE] + 1;
}
```

ALTRES ASPECTES IMPORTANTS

- Per anar a buscar l’últim element de la ‘array’ ‘compressed’ s’utilitza la propietat ‘.length’ que ens retorna la quantitat d’elements de la ‘array’. Com que els índexs dels elements d’una ‘array’ comencen per zero, se li resta u al valor retornat per ‘.length’, si no s’estaria buscant fora de la ‘array’, en un índex que no existeix.

ALTRES POSSIBILITATS

- Es podria posar un sol ‘return’ i que retornes el valor d’una variable. Aquesta variable anteriorment se li posaria el valor de la mida de la suposada ‘array’. Per netedat del codi i per mantenir-ho simple no s’ha fet. En cas que hi hagués més condicionals i més ‘returns’ s’utilitzaria la variable per no perdre el fil del programa.

compressedSize()

INTRODUCCIÓ

- La funció ‘compressedSize()’ retorna a partir d’un polinomi representat de forma estesa, la mida que hauria de tenir una ‘array’ de tipus comprimit per representar el mateix polinomi.

DISSENY

- Per determinar quin hauria de ser la mida d’una ‘array’de tipus comprimit, se segueix el protocol establert. Només seran contats tots els valors diferents de zero per determinar la mida.
- També s’ha de tenir en compte que podem rebre una ‘array’ buida. No fa falta posar condicional en aquest cas, ja que en cas de rebre una ‘array’ de mida igual a zero, el bucle no fa cap iteració i es retorna zero igualment.

CODI

- Paràmetres:
  - expanded → ‘array’ d’enters, representa un polinomi de forma estesa.

- Variables:
  - count → enter, comptador, s’inicialitza igual a zero.

- En cas que la mida de la ‘array’ d’entrada fos igual a zero, no s’entraria al bucle i retornaria zero. Per cada element de la ‘array’ d’entrada, s’incrementa la variable ‘count’ (comptador) mentre trobi valors diferents de zero (les constants iguals a zero no han d’estar dins la forma comprimida d’un polinomi).
- Finalment retorna el valor de la variable ‘count’ (comptador).
- El codi consta de les línies següents:

```
int count = 0;
```

```
for (int element : expanded) {
      if (element != 0) {
            count++;
      }
}

return count;
```

ALTRES ASPECTES IMPORTANTS

- Si d'entrada es té una 'array' de mida igual a zero, no es farà cap iteració.

ALTRES POSSIBILITATS

- Podríem posar un condicional pel cas que d'entrada rebéssim una 'array' de mida igual a zero, però en cas de rebre una 'array' de mida igual a zero, el bucle no fa cap iteració i es retorna zero igualment.

createExpanded()

INTRODUCCIÓ

- La funció 'createExpanded()' retorna una 'array' de tipus estès amb la mida indicada per l'enter entrat com a paràmetre. No es declara la 'array' sinó, que es dóna l'espai de memòria.

DISSENY

- Es retorna l'espai de memòria d'una 'array' amb la mida indicada per l'enter entrat com a paràmetre ('expandedSize').

CODI

- Paràmetres:
  - expandedSize → enter, representa la mida d'una 'array' estesa.

- Es retorna l'espai de memòria que ocupa una 'array' amb la mida indicada per l'enter entrat com a paràmetre ('expandedSize').
- El codi consta de la línia següent:

```
return new int[expandedSize];
```

ALTRES ASPECTES IMPORTANTS

- No es declara la 'array' sinó, només l'espai de memòria que ocupa. La 'array' s'haurà de declarar en un futur.

ALTRES POSSIBILITATS

- Podríem declarar la 'array' al mateix moment, però per netedat del codi no es fa. Tampoc es fa perquè en cridar la funció per assignar el valor retornat a una 'array', seria una 'array' apuntant cap a una altra 'array'. No és necessari.

createCompressed()

INTRODUCCIÓ

- La funció 'createCompressed()' retorna una 'array' de tipus comprimit amb la mida indicada per l'enter entrat com a paràmetre ('compressedSize'). No es declara la 'array' sinó, que es dóna l'espai de memòria.

DISSENY

- Es retorna l'espai de memòria d'una 'array' amb la mida indicada per l'enter entrat com a paràmetre ('compressedSize').

CODI

- Paràmetres:
  - compressedSize → enter, representa la mida d'una 'array' comprimida.

- Es retorna l'espai de memòria que ocupa una 'array' de 'arrays' (matriu) de mida definida per l'enter passat com a paràmetre ('compressedSize') per dos.
- El codi consta de la línia següent:

```
return new int[compressedSize][2];
```

ALTRES ASPECTES IMPORTANTS

- No es declara la 'array' sinó, només l'espai de memòria que ocupa. La 'array' s'haurà de declarar en un futur.

ALTRES POSSIBILITATS

- Podríem declarar la 'array' al mateix moment, però per netedat del codi no es fa. Tampoc es fa perquè en cridar la funció per assignar el valor retornat a una 'array', seria una 'array' apuntant cap a una altra 'array'. No és necessari.

copyTo() #1

INTRODUCCIÓ

- La funció 'copyTo()' no retorna res. A partir de dues 'arrays' d'entrada passa els valors d'una 'array' cap a l'altra 'array'. Aquesta és una funció amb sobrecàrrega, i en aquest cas passa els valors d'una 'array' de tipus estès cap a una 'array' de tipus comprimit. Es dóna per suposat que les 'arrays' tenen la mida adequada pel mateix polinomi que representen.

DISSENY

- Es recorre tota la 'array' 'fromExpanded', i tots els valors diferents de zero els guarda en la 'array' 'toCompressed'. Per cada valor diferent de zero en la 'array' 'fromExpanded' guarda els valors en la 'array' 'toCompressed' de la forma següent: en la posició de l'índex *n* i posició zero, es guarda l'índex on estava el valor en la 'array' 'fromExpanded'; i en la posició de l'índex *n* i posició u el valor en si. L'índex *n* s'actualitza cada vegada guardats una parella de valors per accedir en la posició següent.

ex. :

CODI

- Paràmetres:
  - fromExpanded → 'array' d'enters, representa un polinomi de forma estesa.
  - toCompressed → 'array' de 'arrays' d'enters, representa un polinomi de forma comprimida.

- Variables:
  - index → enter, índex on escriure el valor a la 'array' toCompressed, s'inicialitza igual a zero.

- La variable 'index' determina en quin espai toca escriure el següent valor. El bucle recorre tota la 'array' 'fromExpanded', i per cada pas de la iteració, si es troba un valor diferent de zero: en la posició de l'índex 'index' i posició zero de la 'array' 'toCompressed', es guarda l'índex on estava el valor en la 'array' 'fromExpanded' (exponent, DEGREE); i en la posició de l'índex 'index' i posició u de la 'array' 'toCompressed' el valor en si (constant, COEFFICIENT). S'actualitza la variable 'index' per poder accedir a més posicions en la 'array' 'toCompressed'.
- El codi consta de les línies següents:

```
int index = 0;
for (int i = 0; i < fromExpanded.length; i++) {
      if (fromExpanded[i] != 0) {
            toCompressed[index][DEGREE] = i;
            toCompressed[index][COEFFICIENT] = fromExpanded[i];
            index++;
```

```
        }
    }
```

ALTRES ASPECTES IMPORTANTS

- No s'utilitza un condicional per si entren dos 'arrays' buides com a paràmetres, ja que en aquest cas el bucle no farà cap iteració, se segueix tenint la mateixa funcionalitat.

ALTRES POSSIBILITATS

- Podríem aplicar un condicional per si entren dos 'arrays' buides. No s'haurien de passar cap valor d'una 'array' cap a l'altra. Però com que el bucle no fa cap iteració en aquest cas, ja compleix la seva funcionalitat.

copyTo() #2

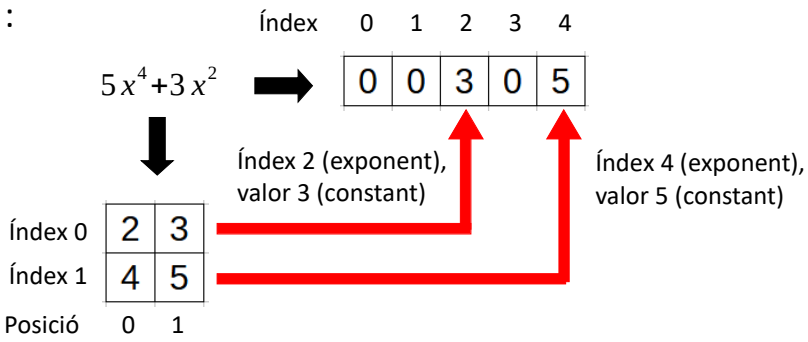
INTRODUCCIÓ

- La funció 'copyTo()' no retorna res. A partir de dues 'arrays' d'entrada passa els valors d'una 'array' cap a l'altra 'array'. Aquesta és una funció amb sobrecàrrega, i en aquest cas passa els valors d'una 'array' de tipus comprimit cap a una 'array' de tipus estès. Es dóna per suposat que les 'arrays' tenen la mida adequada pel mateix polinomi que representen.

DISSENY

- Es recorre tota la 'array' 'fromCompressed', i tots els valors els posa en la 'array' 'toExpanded'. Per cada element en la 'array' 'fromCompressed' es guarda els valors de la posició u en la 'array' 'toExpanded' de la forma següent: l'índex en què anirà el valor en la 'array' 'toExpanded', és el valor de la posició zero; i el valor que anirà en l'índex ja determinat, serà el valor de la posició u.

ex. :



CODI

- Paràmetres:
  - fromCompressed → 'array' de 'arrays' d'enters, representa un polinomi de forma comprimida.
  - toExpanded → 'array' d'enters, representa un polinomi de forma estesa.
- Per cada element de la 'array' 'fromCompressed', es guarda el valor de la posició u en la 'array' 'toExpanded', a l'índex que digui la posició zero de l'element.
- El codi consta de les línies següents:

```
for (int[] element : fromCompressed) {
      toExpanded[element[DEGREE]] = element[COEFFICIENT];
}
```

ALTRES ASPECTES IMPORTANTS

- No s'utilitza un condicional per si entren dos 'arrays' buides com a paràmetres, ja que en aquest cas el bucle no farà cap iteració, se segueix tenint la mateixa funcionalitat.

ALTRES POSSIBILITATS

- Podríem aplicar un condicional per si entren dos 'arrays' buides. No s'haurien de passar cap valor d'una 'array' cap a l'altra. Però com que el bucle no fa cap iteració en aquest cas, ja compleix la seva funcionalitat.

compress()

INTRODUCCIÓ

- La funció ‘compress()’ retorna a partir d’una ‘array’ de tipus estès, una ‘array’ de tipus comprimit representant el mateix polinomi.

DISSENY

- A partir de la ‘array’ de tipus estès, es crea una ‘array’ que podrà representar el mateix polinomi però de forma comprimida.
- S’aprofiten les funcions ja creades, ‘createCompressed()’ per crear una ‘array’ de tipus comprimit, i la funció ‘compressedSize()’ per determinar la mida que hauria de tenir la ‘array’ de tipus comprimit. La funció ‘createCompressed()’ retorna l’espai de memòria d’una ‘array’, només fa falta declarar la ‘array’.
- S’haurà de passar tots els valors, seguint el protocol marcat, de la ‘array’ de tipus estès a la ‘array’ de tipus comprimit. Es farà ús de la funció ‘copyTo()’.

CODI

- Paràmetres:
  - expanded → ‘array’ d’enters, representa un polinomi de forma estesa.

- Variables:
  - compressedArray → ‘array’ de ‘arrays’ d’enters, ‘array’ que serà retornada amb el polinomi representat de forma comprimida.

- Es declara la ‘array’ ‘compressedArray’ i l’espai de memòria es ve donat per la funció ‘createCompressed()’. El valor que determina la mida a la funció ‘createCompressed()’ és la funció ‘compressedSize()’, que per paràmetre d’entrada rep la ‘array’ ‘expanded’. Les funcions estan una dins de l’altra, d’aquesta forma quan la funció de dins retorna un valor, ja és agafat per l’altra funció (No fa falta declarar una variable per emmagatzemar el valor).

- En la funció ‘copyTo()’ es passaran els valors d’una ‘array’ a l’altra. La funció ‘copyTo()’ ja és explicada en el seu apartat, no es retorna cap valor, només actualitza les ‘arrays’.
- Finalment es retorna la ‘array’ de tipus comprimit.
- El codi consta de les línies següents:

```
int[][] compressedArray = createCompressed(compressedSize(expanded));

copyTo(expanded, compressedArray);

return compressedArray;
```

ALTRES ASPECTES IMPORTANTS

- La funció ‘compressedSize()’ retorna un valor que és directament passat com a paràmetre a la funció ‘createCompressed()’. No és un valor que es guardi en una variable.

ALTRES POSSIBILITATS

- Es podria posar un condicional pel cas que la ‘array’ que rebem com a paràmetre d’entrada fos de mida igual a zero. No es posa per mantenir el codi simple, ja que tampoc canvia la funcionalitat de la funció. És cert que es cridarà la funció ‘copyTo()’, però no efectuarà cap canvi en les ‘arrays’.

expand()

INTRODUCCIÓ

- La funció ‘expand()’ retorna a partir d’una ‘array’ de tipus comprimit, una ‘array’ de tipus estès representant el mateix polinomi.

DISSENY

- A partir de la ‘array’ de tipus comprimit, es crea una ‘array’ que podrà representar el mateix polinomi però de forma estesa.

- s'aprofiten les funcions ja creades, 'createExpanded()' per crear una 'array' de tipus comprimit, i la funció 'expandedSize()' per determinar la mida que hauria de tenir la 'array' de tipus estès. La funció 'createExpanded()' retorna l'espai de memòria d'una 'array', només fa falta declarar la 'array'.
- S'haurà de passar tots els valors seguint el protocol marcat de la 'array' de tipus estès a la 'array' de tipus comprimit. Es farà ús de la funció 'copyTo()'.

CODI

- Paràmetres:
  - compressed → 'array' de 'arrays' d'enters, representa un polinomi de forma comprimida.

- Variables:
  - expandedArray → 'array' d'enters, 'array' que serà retornada amb el polinomi representat de forma estesa.

- Es declara la 'array' 'expandedArray' i l'espai de memòria es ve donat per la funció 'createExpanded()'. El valor que determina la mida a la funció 'createExpanded()' és la funció 'expandedSize()', que per paràmetre d'entrada rep la 'array' 'compressed'. Les funcions estan una dins de l'altra, d'aquesta forma quan la funció de dins retorna un valor, ja és agafat per l'altra funció (No fa falta declarar una variable per emmagatzemar el valor).
- En la funció 'copyTo()' es passaran els valors d'una 'array' a l'altra. La funció 'copyTo()' ja és explicada en el seu apartat, no retorna cap valor, només actualitza les 'arrays'.
- Finalment es retorna la 'array' de tipus estès.
- El codi consta de les línies següents:

```
int[] expandedArray = createExpanded(expandedSize(compressed));

copyTo(compressed, expandedArray);

return expandedArray;
```

ALTRES ASPECTES IMPORTANTS

- La funció ‘expandedSize()’ retorna un valor que és directament passat com a paràmetre a la funció ‘createExpanded()’. No és un valor que es guardi en una variable.

ALTRES POSSIBILITATS

- Es podria posar un condicional pel cas que la ‘array’ que rebem com a paràmetre d’entrada fos de mida igual a zero. No es posa per mantenir el codi simple, ja que tampoc canvia la funcionalitat de la funció. És cert que es cridarà la funció ‘copyTo()’, però no efectuarà cap canvi en les ‘arrays’.

pow()

INTRODUCCIÓ

- La funció ‘pow()’ retorna la solució de calcular un número elevat a un altre.

DISSENY

- La funció rep dos nombres enters, i resol l’operació següent: el primer numero (base) elevat al segon (exponent).
- Per resoldre aquesta operació es fa un bucle que itera tantes vegades com marca l’exponent. Per cada pas de la iteració es multiplica el resultat del pas anterior per la base. En el primer pas de la iteració es multiplica u per la base.

CODI

- Paràmetres:
  - base → enter, base d’una operació exponencial.
  - exp → enter, exponent d’una operació exponencial.

- Variables:
  - result → enter, s'inicialitza a u. Resultat de l'operació duna base elevada a un exponent.

- En la variable 'result' s'inicialitza amb un u perquè és l'element neutre en la multiplicació (exprés pel primer pas de la iteració).
- En el bucle es fan tantes iteracions com indiqui la variable 'exp'. I per cada pas de la iteració es multiplica la variable 'result' per la variable 'base', i el resultat es guarda també en la variable 'result'.
- Finalment es retorna el valor de la variable 'result'.
- El codi consta de les línies següents:

```
int result = 1;

for (int i = 0; i < exp; i++) {
      result *= base;
}

return result;
```

ALTRES ASPECTES IMPORTANTS

- Dins el bucle hi ha la contracció '*=', que de forma estesa es podria expressar com: el resultat és igual al resultat per un número. D'aquesta forma s'estalvia text i queda el codi més net.

ALTRES POSSIBILITATS

- Dins el bucle es podria posar un condicional per al primer pas de la iteració. Es podria passar el valor de la variable 'base' a la variable 'result' i no efectuar una multiplicació de més. Però per mantenir el codi net i senzill no s'ha fet.

evaluate() #1

INTRODUCCIÓ

- La funció ‘evaluate()’ rep una ‘array’ que representa un polinomi de forma estesa i un enter que serà el valor que substituirà les variables en el polinomi per avaluar-los i obtenir un resultat. És una funció amb sobrecàrrega, i en aquest cas resol un polinomi codificat de forma estesa.

DISSENY

- Per resoldre el polinomi s’aplicarà l’algoritme de Horner

  (Algoritme de Horner).

- L’algoritme de Horner funciona de la següent manera:
  - Partint d’un polinomi $a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + a_2 x^2 + a_1 x + a_0$

    On a_0, a_1, a_2, ... a_{n-2}, a_{n-1}, a_n són les constants dels monomis del polinomi, i *n* és el grau del polinomi.

    La constant a_n és multiplicada *n* vegades per la variable ‘x’, una més que la constant a_{n-1}. La constant a_{n-1} és multiplicada una vegada més per la variable ‘x’ que la constant a_{n-2}. I així successivament fins a arribar a la a_0, que no és multiplicada per la variable ‘x’. Partint d’aquesta deducció, podem reformular el polinomi de la forma següent:
    $a_0 + x(a_1 + x(a_2 + x(\ldots(a_{n-2} + x(a_{n-1} + x(a_n))))))$
  - Exemple:

    Polinomi $5x^3 + 6x^2 + 7x + 8$

    substituir la variable ‘X’ per 2

    Algoritme de Horner $8 + 2(7 + 2(6 + 2(5)))$

- Si tenim una ‘array’ que conté totes les constants d’un polinomi ordenat (inclús constants que són igual a zero), es pot aplicar l’Algoritme de Horner. Per aplicar-lo s'ha de recórrer la ‘array’ del revés. En cada pas de la iteració primer es multiplica el resultat del pas de la iteració anterior per la variable, i a continuació se suma la constant. En el primer pas de la iteració es multiplica la variable per zero (segueix sent el resultat igual a zero), després al sumar la constant, el resultat final canvia (en el següent pas de la iteració no es multiplicarà per zero).

CODI

- Paràmetres:
  - expanded → ‘array’ d’enters, representa un polinomi de forma estesa.
  - x → enter, valor per substituir a la variable del polinomi.

- Variables:
  - total → enter, s’inicialitza a zero. Resultat de resoldre el polinomi.
  - len → enter, allargada de la ‘array’ ‘expanded’.

- S’itera del reves per tota la ‘array’ ‘expanded’, i en cada pas de la iteració, es guarda el resultat en la variable ‘total’, la multiplicació de la variable ‘x’ per la variable ‘total’; i després es guarda el resultat en la variable ‘total’, la suma de la variable ‘total’ més un valor de la ‘array’. En cada pas de la iteració, l’índex de la ‘array’, que ens marca el valor la qual sumar en cada pas de la iteració, va decreixent (es comença pel final i acaba en el principi de la ‘array’).
- Finalment es retorna el valor de la variable ‘total’.
- El codi consta de les línies següents:

```
int total = 0;
int len = expanded.length;

for (int i = len - 1; i >= 0; i--) {
      total *= x;
      total += expanded[i];
}

return total;
```

ALTRES ASPECTES IMPORTANTS

- El bucle recorre la ‘array’ de forma inversa (comença pel final).
- Dins el bucle ens trobem la contracció ‘+=’, que de forma estesa es podria expressar com: el resultat és igual al resultat més un número. D’aquesta forma s’estalvia text i queda el codi més net.

ALTRES POSSIBILITATS

- Es podria resoldre el polinomi amb la funció 'pow()', però és menys eficient, ja que ha de fer més multiplicacions (com més multiplicacions, més ineficient). Amb l'Algoritme de Horner s'efectuen el mínim de multiplicacions possibles.

evaluate() #2

INTRODUCCIÓ

- La funció 'evaluate()' rep una 'array' que representa un polinomi de forma comprimida i un enter que serà el valor que substituirà les variables en el polinomi per resoldre el polinomi. És una funció amb sobrecàrrega, i en aquest cas resol un polinomi codificat de forma comprimida.

DISSENY

- S'itera per tota la 'array', i per cada element multiplica la constant per la variable 'x' del monomi elevada l'exponent indicat. S'utilitza la funció 'pow()' per resoldre els monomis.
- ex. :

  $3x^4+5x^2+4$ ➡ 3 * pow(x,4) + 5 * pow(x,2) + 4 * pow(x,0)

CODI

- Paràmetres:
  - compressed → 'array' de 'arrays' d'enters, representa un polinomi de forma comprimida.
  - x → enter, valor per substituir a la variable del polinomi.

- Variables:
  - total → enter, s'inicialitza a zero. Resultat de resoldre el polinomi.

- En el bucle s'itera per tots els elements de la 'array'. Per cada element, suma a la variable 'total', el resultat de multiplicar el valor de la posició u de l'element (la constant), i la variable 'x' elevada al valor de la posició zero de l'element (l'exponent). Per fer l'operació exponencial s'utilitza la funció ja descrita 'pow()'.
- Finalment es retorna el valor de la variable 'total'.
- El codi consta de les línies següents:

```
int total = 0;

for (int[] element : compressed) {
      total += pow(x, element[DEGREE]) * element[COEFFICIENT];
}

return total;
```

ALTRES ASPECTES IMPORTANTS

- En el principi del codi es donen dos constants: 'DEGREE', sent igual a zero; i 'COEFFICIENT', sent igual a u. 'DEGREE' es refereix a l'exponent i 'COEFFICIENT' a la constant d'un monomi.

ALTRES POSSIBILITATS

- Es podria aplicar l'Algoritme de Horner i emplenar tots els monomis buits amb zeros. El programa seria més complex, perdent així la netedat del codi i la seva simplicitat i eficiència.

makeCopy() #1

INTRODUCCIÓ

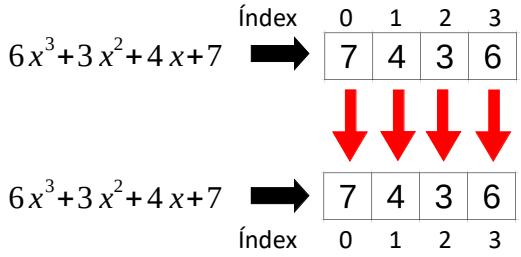
- La funció 'makeCopy()' rep dos 'arrays' que representa un polinomi de forma estesa. La funció no retorna cap valor, només fa una còpia entre 'arrays'. Per tots els elements de la primera 'array', entrada com a paràmetre, copia els elements a la segona 'array', entrada com a paràmetre. És una funció amb

sobrecàrrega, i en aquest cas copia una ‘array’ de tipus estès a una altra ‘array’ de tipus estès.

DISSENY

- S’itera per tota la ‘array’ ‘originExpanded’, i en cada pas de la iteració, es copia l’element de la ‘array’ ‘originExpanded’ a la ‘array’ ‘destinatonExpanded’ amb el mateix ordre.

  ex. :



CODI

- Paràmetres:
  - originExpanded → ‘array’ d’enters, representa un polinomi de forma estesa.
  - destinatonExpanded → ‘array’ d’enters, representa un polinomi de forma estesa.

- En el bucle s’itera per tots els elements de la ‘array’ ‘originExpanded’ i guarda cada element amb el mateix índex en la ‘array’ ‘destinatonExpanded’.
- El codi consta de les línies següents:

```
for (int i = 0; i < originExpanded.length; i++) {
      destinationExpanded[i] = originExpanded[i];
}
```

ALTRES ASPECTES IMPORTANTS

- Aquesta funció no retorna cap valor, ja que la seva funció és actualitzar els valors de les ‘arrays’.

ALTRES POSSIBILITATS

- Es podria posar un condicional per si tinguéssim ‘arrays’ de mida igual a zero. Però com que el bucle no faria cap iteració, no canvia la funcionalitat.
- No es permet utilitzar ‘System.arraycopy()’ en aquesta pràctica, però seria molt útil per passar els valors d’una ‘array’ a una altra.

makeCopy() #2

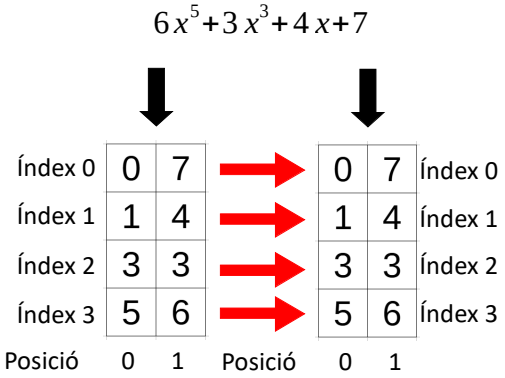
INTRODUCCIÓ

- La funció ‘makeCopy()’ rep dos ‘arrays’ que representa un polinomi de forma comprimida. La funció no retorna cap valor, només fa una còpia entre ‘arrays’. Per totes els elements de la primera ‘array’, entrada com a paràmetre, copia els elements a la segona ‘array’, entrada com a paràmetre. És una funció amb sobrecàrrega, i en aquest cas copia els valors d’una ‘array’ de tipus comprimit a una altra ‘array’ de tipus comprimit.

DISSENY

- S’itera per tota la ‘array’ ‘originCompressed’, i en cada pas de la iteració, es copien els elements de l’element de la ‘array’ ‘originCompressed’ amb el mateix ordre a la ‘array’ ‘destinationCompressed’.

ex. :



CODI

- Paràmetres:
  - originCompressed → ‘array’ de ‘arrays’ d’enters, representa un polinomi de forma comprimida.
  - destinatonCompressed → ‘array’ de ‘arrays’ d’enters, representa un polinomi de forma comprimida.

- En el bucle s’itera per tots els elements de la ‘array’ ‘originCompressed’ i guarda els valors de les dos posicions en la ‘array’ ‘destinatonCompressed’ mantenint l’ordre.
- El codi consta de les línies següents:

```
for (int i = 0; i < originCompressed.length; i++) {
      destinationCompressed[i][DEGREE] = originCompressed[i][DEGREE];
      destinationCompressed[i][COEFFICIENT] =
originCompressed[i[COEFFICIENT];
}
```

ALTRES ASPECTES IMPORTANTS

- Aquesta funció no retorna cap valor, ja que la seva funció és actualitzar els valors de les ‘arrays’.

ALTRES POSSIBILITATS

- Es podria posar un condicional per si tinguéssim ‘arrays’ de mida igual a zero. Però com que el bucle no faria cap iteració, no canvia la funcionalitat.

add() #1

INTRODUCCIÓ

- La funció ‘add()’ rep dos ‘arrays’ que representen un polinomi de forma estesa i retorna el resultat de sumar els dos polinomis (retorna un altre polinomi codificat de forma estesa). És una funció amb sobrecàrrega, i en aquest cas resol la suma de dos polinomis codificats de forma estesa.

DISSENY

- Es comprova que les dos ‘arrays’ ‘expandedX’ tenen una mida superior a zero, i poden passar tres casos diferents.
- En el cas que la mida de les dos ‘arrays’ sigui igual a zero, es retorna una nova ‘array’ de mida zero. Es fa una còpia per a que la ‘array’ resultant no entri amb conflicte amb la ‘array’ entrada com a paràmetre.
- En cas que només la mida d’una ‘array’ sigui superior a zero, es retorna una còpia de la ‘array’ de mida superior que zero. Es fa una còpia perquè la ‘array’ resultant no entri amb conflicte amb la ‘array’ entrada com a paràmetre.
- En cas que la mida de les dos ‘arrays’ siguin superior a zero, se segueixen els passos següents:
  - Es determina la ‘array’ amb la mida més gran, i la ‘array’ amb la mida més petita.

    ex. 1:

    expanded1 ➡ | 4 | 3 | 6 |

    expanded2 ➡ | 2 | 5 | 1 | 3 |

    Mida més gran: expanded2; mida més petita: expanded1.

    ex. 2:

    expanded1 ➡ | 2 | 5 | 1 | 3 |

    expanded2 ➡ | 5 | 2 | -1 | -3 |

    Les dos iguals.

  - Es determina la mida de la ‘array’ ‘expandedResultIsh’, que serà igual a la mida de la ‘array’ ‘expandedX’ amb la mida més gran.

    ex. 1:

    expanded1 ➡ | 4 | 3 | 6 |

    - expanded2 ➡ | 2 | 5 | 1 | 3 |

    expandedResultIsh ➡ | | | | |

ex. 2:

expanded1 ➡ | 2 | 5 | 1 | 3 |

expanded2 ➡ | 5 | 2 | -1 | -3 |

expandedResultIsh ➡ | | | | |

- S'itera per tots els indexes en comú entre les dos 'arrays'. En la 'array' 'expandedResultIsh', es guarda, amb el mateix ordre, la suma dels valors amb el mateix índex de les 'arrays' 'expandedX'.

  ex. 1:

  expanded1 ➡ | 4 | 3 | 6 |

  expanded2 ➡ | 2 | 5 | 1 | 3 |

  4+2 = 6; 3+5 = 8; 6+1 = 7

  expandedResultIsh ➡ | 6 | 8 | 7 | |

  ex. 2:

  expanded1 ➡ | 2 | 5 | 1 | 3 |

  expanded2 ➡ | 5 | 2 | -1 | -3 |

  2+5 = 7; 5+2 = 7; 1+(-1) = 0; 3+(-3) = 0

  expandedResultIsh ➡ | 7 | 7 | 0 | 0 |

- Si hi ha una 'array' que, al tenir una mida més gran que l'altra, no s'ha pogut passar els valors a la 'array' 'expandedResultIsh', s'itera per la resta de valors i es passen a la 'array' 'expandedResultIsh'.

  ex. 1:

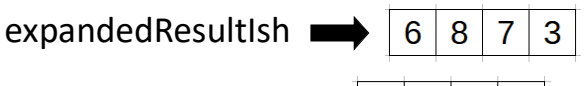
  expandedResultIsh (abans) ➡ | 6 | 8 | 7 | |

  expanded2 ➡ | 2 | 5 | 1 | 3 |

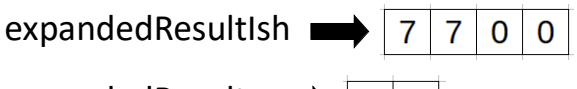
  expandedResultIsh ➡ | 6 | 8 | 7 | 3 |

- Es determina la mida de la 'array' 'expandedResult'. La mida serà igual a la mida de la 'array' 'expandedX' amb la mida més gran, menys tots els espais que han quedat amb valors iguals a zero i no tenen cap altre valor diferent de zero més endavant.

ex. 1:

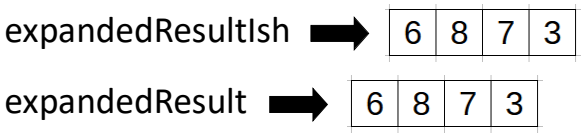


ex. 2:

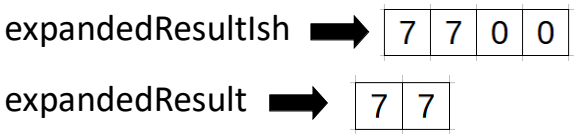


- S'itera per la 'array' 'expandedResultIsh' fins a l'arribar a l'últim valor diferent de zero, i per cada pas de la iteració, es copia l'element amb el mateix ordre a la 'array' 'expandedResult'.

ex. 1:



ex. 2:



- Finalment es retorna la 'array' 'expandedResult' (independentment de per quina secció del programa sigui modificada).

CODI

- Paràmetres:
  - expanded1 → 'array' d'enters, representa un polinomi de forma estesa.
  - expanded2 → 'array' d'enters, representa un polinomi de forma estesa.

- Variables:
  - len1 → enter, allargada de la 'array' 'expanded1'.
  - len2 → enter, allargada de la 'array' 'expanded2'.
  - largest → enter, l'allargada de la 'array' més gran.

- shortest → enter, l'allargada de la ‘array’ més petita.
- largestArray → enter, si té un zero, les dos ‘arrays’ són igual de llargues; si té un u, la ‘array’ ‘expanded1’ és més llarga que la ‘array’ ‘expanded2’; si té un dos, la ‘array’ ‘expanded2’ és més llarga que la ‘array’ ‘expanded1’.
- lastIndex → enter, l’últim índex on a la ‘array’ ‘expandedResultIsh’ hi ha el valor diferent de zero.
- expandedResultIsh → ‘array’ d’enters, resultat de sumar els dos polinomis. Aquesta ‘array’ no és la definitiva, és possible que tingui espais que s’hagin de suprimir. L’extensió ‘Ish’ prové de l'anglès ‘-ish’ (traduït com: quasi, més o menys, per l’estil...).
- expandedResult → ‘array’ d’enters, ‘array’ resultant de la suma dels dos polinomis. Array que serà retornada.

- Es determina si hi ha alguna ‘array’ ‘expandedX’ de mida zero (comparant les variables ‘len1’ i ‘len2’ amb el zero). Els casos poden ser: les dos són de mida igual a zero, una és de mida igual a zero i l’altra és de mida més gran que zero, i les dos són de mida més gran que zero.
- Comparació de les ‘arrays’ per determinar de quin cas es tracta:

```
if (len1 > 0 && len2 > 0) {
...
}
else if (len1 > 0 && len2 == 0) {
...
}
else if (len1 == 0 && len2 > 0) {
...
}
else {
...
}
```

- En cas que les dos ‘arrays’ són de mida igual a zero, es dóna l’espai de memòria igual a zero a la ‘array’ ‘expandedResult’, utilitzant la funció ‘createExpanded()’.
- Donar l’espai de memòria igual a zero a la ‘array’ ‘expandedResult’:

```
expandedResult = createExpanded(0);
```

- En cas que solament una ‘array’ és de mida igual a zero, es copia els valors de la ‘array’ de mida més gran que zero a la ‘array’ ‘expandedResult’. La còpia és efectuada per la funció ‘makeCopy()’.
- Còpia de valors de la ‘array’ de mida superior a zero a la ‘array’ ‘expandedResult’ (l’exemple mostrat és en cas que la ‘array’ ‘expanded2’ és de mida igual a zero):

```
expandedResult = createExpanded(len1);
makeCopy(expanded1, expandedResult);
```

- En cas que la mida de les dos ‘arrays’ són superior a zero, es determina quina és la de la mida més gran, més petita, o si són iguals. En cas que la mida de les ‘arrays’ són diferents: en la variable ‘largest’ es guarda la mida de la ‘array’ de mida més gran, en la variable ‘shortest’ es guarda la mida de la ‘array’ de mida més petita, i en la variable ‘largestArray’ es guarda un u si la mida de la ‘array’ ‘expanded2’ és la més gran, i un dos si la mida de la ‘array’ ‘expanded1’ és la més gran (es compara els valors de les variables ‘len1’ i ‘len2’). En cas que la mida de les dos ‘arrays’ són iguals: en les variables ‘shortest’ i ‘largest’ es guarda el valor de ‘len1’, i en la variable ‘largestArray’ es guarda un zero. Es dóna l’espai de memòria de la ‘array’ ‘expandedResultIsh’ amb la mida determinada per la variable ‘largest’.
- Determinar per mides, la ‘array’ més gran i petita:

```
if (len1 > len2) {
      largest = len1;
      shortest = len2;
      largestArray = 1;
}
else if (len1 < len2) {
      largest = len2;
      shortest = len1;
      largestArray = 2;
}
else {
      largest = shortest = len1;
```

```
	largestArray = 0;
}

expandedResultIsh = createExpanded(largest);
```

- El bucle itera tantes vegades com marca la variable 'shortest'. En cada pas de la iteració suma els dos elements de les 'arrays' 'expandedX' amb el mateix índex que marca el bucle, i el resultat es guarda, en el mateix índex, a la 'array' 'expandedResultIsh'. Al mateix moment de fer la suma, si el valor de la suma és diferent de zero i la variable 'largestArray' és igual a zero, es guarda en la variable 'lastIndex' l'índex actual del pas de la iteració. L'últim valor guardat en la variable 'lastIndex' ens servirà per saber quina és la última posició d'un valor diferent de zero en la 'array' 'expandedResultIsh'.
- Suma parcial/total de les dos 'arrays' 'expandedX':

```
for (int i = 0; i < shortest; i++) {
	expandedResultIsh[i] = expanded1[i] + expanded2[i];
	if (expanded1[i] + expanded2[i] != 0 && largestArray == 0) {
		lastIndex = i + 1;
	}
}
```

- En cas que les mides de les 'arrays' 'expandedX' siguin diferents, s'iterarà per la resta no iterada en el bucle anterior, i es passen els valors a la 'array' 'expandedResultIsh' a continuació dels valors ja passats en el bucle anterior. En la variable 'lastIndex' es guarda el valor de la variable 'largest'. L'últim valor diferent de zero es donat directament per la 'array' amb la mida més gran, i és en la posició que marca la variable 'largest' (index 'largest' menys u).
- Còpia de la resta de valors no passats:

```
if (largestArray != 0) {
	if (largestArray == 1) {
		for (int i = shortest; i < largest; i++) {
			expandedResultIsh[i] = expanded1[i];
		}
	} else if (largestArray == 2) {
		for (int i = shortest; i < largest; i++) {
			expandedResultIsh[i] = expanded2[i];
		}
	}

	lastIndex = largest;
}
```

- Es dóna l'espai de memòria de la 'array' 'expandedResult' amb la mida determinada per la variable 'lastIndex'. El bucle itera tantes vegades com marca la variable 'lastIndex'. Per cada pas de la iteració es copia els valors de la 'array' 'expandedResultIsh' a la 'array' 'expandedResult' en el mateix índex.
- Passar valors a la 'array' 'expandedResult':

```
expandedResult = createExpanded(lastIndex);

for (int i = 0; i < lastIndex; i++) {
      expandedResult[i] = expandedResultIsh[i];
}
```

- Finalment es retorna la 'array' 'expandedResult'.
- Retorn d'una 'array' de mida igual a zero:

```
return expandedResult;
```

ALTRES ASPECTES IMPORTANTS

- La variable 'lastIndex' ens indica quin és l'últim índex on hi ha l'últim valor de la 'array' 'expandedResultIsh' diferent de zero. Si hi ha una 'array' 'expandedX' de mida més gran que l'altra, sempre hi haurà un valor diferent de zero en l'última posició de la 'array' 'expandedResultIsh'.

ALTRES POSSIBILITATS

- S'havia pensat en retornar una 'array' per cada cas diferent (si les dos 'arrays' tenen mida més gran que zero, si només una o si cap de les dos), però per deixar el codi més net i entenedor s'utilitza la mateixa 'array' ('expandedResultIsh').

add() #2

INTRODUCCIÓ

- La funció 'add()' rep dos 'arrays' que representen un polinomi de forma comprimida i retorna el resultat de sumar els dos polinomis (retorna un altre polinomi codificat de forma comprimida). És una funció amb sobrecàrrega, i en aquest cas resol la suma de dos polinomis codificats de forma comprimida.

DISSENY

- Es comprova que les dos ‘arrays’ ‘compressedX’ (‘arrays’ entrades com a paràmetre) tenen una mida superior a zero, i poden passar tres casos diferents.
- En el cas que la mida de les dos ‘arrays’ sigui igual a zero, es retorna una nova ‘array’ de mida zero. Es fa una còpia perquè la ‘array’ resultant no entri amb conflicte amb la ‘array’ entrada com a paràmetre.
- En cas que només la mida d’una ‘array’ sigui superior a zero, es retorna una còpia de la ‘array’ de mida superior que zero. Es fa una còpia perquè la ‘array’ resultant no entri amb conflicte amb la ‘array’ entrada com a paràmetre.
- En cas que la mida de les dos ‘arrays’ siguin superior a zero, es segueixen els passos següents:
    - S’entra en un bucle que, mentre no s’hagin sumat/passats tots els elements de les ‘arrays’ ‘compressedX’ seguirà iterant. Es pot diferenciar el bucle en dos parts: la primera, que tracta de comparar dos elements de les ‘arrays’ ‘compressedX’ per afegir-los/sumar-los en la ‘array’ ‘compressedResultIsh’ (‘array’ dús temporal); i la segona, que passa tots els valors de la ‘array’ no iterada (si es dóna el cas) a la ‘array’ ‘compressedResultIsh’. La mida de la ‘array’ ‘compressedResultIsh’ és igual a la suma de les mides de les ‘arrays’ ‘compressedX’.
    - La primera secció del bucle compara els valors de la posició zero de dos elements marcats per l’índex de cada ‘array’. Es poden donar tres casos: que el valor de la ‘array’ ‘compressed1’ sigui més gran que el de la ‘array’ ‘compressed2’; que el valor de la ‘array’ ‘compressed1’ sigui més petit que el de la ‘array’ ‘compressed2’; i que els dos valors siguin iguals:
        - Si els valors de la posició u són diferents, s’afegeix l’element amb el valor més baix en la ‘array’ ‘compressedResultIsh’ i s’incrementa l’índex de la mateixa ‘array’ ‘compressedX’.

ex.:

compressed1 ➡

	Posició 0	Posició 1
Índex 0	0	4
Índex 1	2	6
Índex 2	3	3

compressed2 ➡

	Posició 0	Posició 1
Índex 0	1	2
Índex 1	4	5
Índex 2	6	3

Pas 1 de la iteració:

compressed1 amb índex 0 ➡ | 0 | 4 |

compressed2 amb índex 0 ➡ | 1 | 2 |

Com que 0 < 1, es passa l'element de la 'array' 'compressed1' amb l'índex 0 a la 'array' 'compressedResultIsh'.

compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	0	4
Índex 1		
Índex 2		
Índex 3		
Índex 4		
Índex 5		

L'índex de la 'array' 'compressed1' augmenta u per poder accedir en una posició superior.

Pas 2 de la iteració:

compressed1 amb índex 1 ➡ | 2 | 6 |

compressed2 amb índex 0 ➡ | 1 | 2 |

com que 2 > 1, es passa l'element de la 'array' 'compressed1' amb l'índex 0 a la 'array' 'compressedResultIsh'.

compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	0	4
Índex 1	1	2
Índex 2		
Índex 3		
Índex 4		
Índex 5		

L'índex de la 'array' 'compressed2' augmenta u per poder accedir en una posició superior.

- Si els valors de la posició u són iguals i els valors de la posició zero, sumen diferent de zero, s'afegeix el valor de la posició zero (no importa de quina 'array') i es fan la suma dels valors de la posició zero. S'incrementa l'índex de les dos 'arrays' 'compressedX'.

  ex.:

  compressed1 ➡

	Posició 0	Posició 1
Índex 0	0	2
Índex 1	1	3
Índex 2	2	4

  compressed2 ➡

	Posició 0	Posició 1
Índex 0	0	2
Índex 1	1	-3
Índex 2	2	3

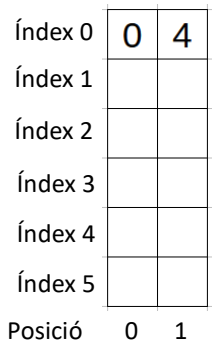
  Pas 1 de la iteració:

  compressed1 amb índex 0 ➡ | 0 | 2 |

  compressed2 amb índex 0 ➡ | 0 | 2 |

  com que 0 = 0 i la suma dels valors de la posició u és diferent de zero, es passa la suma dels valors de la posició u dels elements de les 'arrays' 'compressedX' i el valor de la posició zero, amb l'índex corresponent per cada un a la 'array' 'compressedResultIsh'.

- compressedResultIsh ➡



Els indexos de les ‘arrays’ ‘compressedX’ augmenten u per poder accedir en una posició superior.

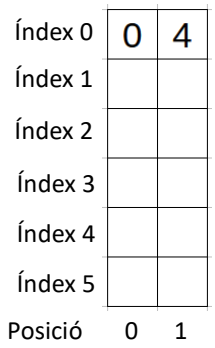
Pas 2 de la iteració:

compressed1 amb índex 1 ➡ | 1 | 3 |

compressed2 amb índex 1 ➡ | 1 | -3 |

com que 1 = 1 i la suma dels valors de la posició u no és diferent de zero, no es passa la suma dels elements de les ‘arrays’ ‘compressedX’ a la ‘array’ ‘compressedResultIsh’.

compressedResultIsh ➡

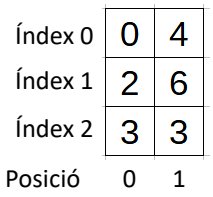


Els indexos de les ‘arrays’ ‘compressedX’ augmenten u per poder accedir en una posició superior.

- La segona secció del bucle itera (si és que ha quedat alguna part d’una array sense iterar) per la ‘array’ no iterada i es passen a la ‘array’ ‘compressedResultIsh’.

  ex.:

  compressed1 ➡



compressed2 ➡

	Posició 0	Posició 1
Índex 0	1	2
Índex 1	4	5
Índex 2	6	3

compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	0	4
Índex 1	1	2
Índex 2		
Índex 3		
Índex 4		
Índex 5		

Passar elements.

compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	0	4
Índex 1	1	2
Índex 2	2	6
Índex 3	3	3
Índex 4	4	5
Índex 5	6	3

- Es determina la mida de la ‘array’ ‘compressedResult’ (‘array’ que es retornada). La mida serà igual a la quantitat d’elements entrats a la ‘array’ ‘compressedResultIsh’ (és possible que no tingui tots els espais amb valors), no hi haurà elements que s'hagin de suprimir.

  ex. 1:

  compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	0	3
Índex 1	2	7
Índex 2	3	8
Índex 3	5	2

  compressedResult ➡

	Posició 0	Posició 1
Índex 0		
Índex 1		
Índex 2		
Índex 3		

ex. 2:

compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	4	3
Índex 1	6	2
Índex 2		
Índex 3		

compressedResult ➡

	Posició 0	Posició 1
Índex 0		
Índex 1		

- Es passaran tots els elements de la 'array' 'compressedResultIsh' que en la posició u tinguin elements diferents de zero, a la 'array' 'compressedResult'.

ex. 1:

compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	0	3
Índex 1	2	7
Índex 2	3	8
Índex 3	5	2

compressedResult ➡

	Posició 0	Posició 1
Índex 0	0	3
Índex 1	2	7
Índex 2	3	8
Índex 3	5	2

ex. 2:

compressedResultIsh ➡

	Posició 0	Posició 1
Índex 0	4	3
Índex 1	6	2
Índex 2		
Índex 3		

compressedResult ➡

	Posició 0	Posició 1
Índex 0	4	3
Índex 1	6	2

- Finalment es retorna la ‘array’ ‘compressedResult’ (independentment de per quina secció del programa sigui modificada).

CODI

- Paràmetres:
  - compressed1 → ‘array’ de ‘arrays’ d’enters, representa un polinomi de forma comprimida.
  - compressed2 → ‘array’ de ‘arrays’ d’enters, representa un polinomi de forma comprimida.

- Variables:
  - len1 → enter, allargada de la ‘array’ ‘compressed1’.
  - len2 → enter, allargada de la ‘array’ ‘compressed2’.
  - index1 → enter, índex de la ‘array’ ‘compressed1’.
  - index2 → enter, índex de la ‘array’ ‘compressed2’.
  - sumIndex1 → booleà, si es necessita incrementar la variable ‘index1’.
  - sumIndex2 → booleà, si es necessita incrementar la variable ‘index2’.
  - ended1 → booleà, si s’ha arribat al final de la ‘array’ ‘compressed1’.
  - ended2 → booleà, si s’ha arribat al final de la ‘array’ ‘compressed2’.
  - indexRIsh → enter, índex de la ‘array’ ‘compressedResultIsh’.
  - compressedResultIsh → ‘array’ de ‘arrays’ d’enters, resultat de sumar els dos polinomis. Aquesta ‘array’ no és la definitiva, és possible que tingui espais que s’hagin de suprimir. L’extensió ‘Ish’ prové de l'anglès ‘-ish’ (traduït com: quasi, més o menys, per l’estil...).
  - indexR → enter, índex de la ‘array’ ‘compressedResult’.
  - compressedResult → ‘array’ de ‘arrays’ d’enters,‘array’ resultant de la suma dels dos polinomis. ‘array’ que serà retornada.

- Es determina si hi ha alguna ‘array’ ‘compressedX’ de mida zero (comparant les variables ‘len1’ i ‘len2’ amb el zero). Els casos poden ser: les dos són de mida igual a zero, una és de mida igual a zero i l’altra és de mida més gran que zero, i les dos són de mida més gran que zero.
- Comparació de les ‘arrays’ per determinar de quin cas es tracta:

```
if (len1 > 0 && len2 > 0) {
...
}
else if (len1 > 0 && len2 == 0) {
...
}
else if (len1 == 0 && len2 > 0) {
...
}
else {
...
}
```

- En cas que les dos ‘arrays’ són de mida igual a zero, es dóna l’espai de memòria igual a zero a la ‘array’ ‘compressedResult’, utilitzant la funció ‘createCompressed()’.
- Donar l’espai de memòria igual a zero a la ‘array’ ‘compressedResult’:

```
compressedResult = createCompressed(0);
```

- En cas que solament una ‘array’ és de mida igual a zero, es copia els valors de la ‘array’ de mida més gran que zero a la ‘array’ ‘compressedResult’. La còpia és efectuada per la funció ‘makeCopy()’.
- Còpia de valors de la ‘array’ de mida superior a zero a la ‘array’ ‘compressedResult’ (l’exemple mostrat és en cas que la ‘array’ ‘compressed2’ és de mida igual a zero):

```
compressedResult = createCompressed(len1);
makeCopy(expanded1, compressedResult);
```

- En cas que la mida de les dos ‘arrays’ són superior a zero, s’entra en un bucle que itera mentre almenys una de les variables ‘ended1’ i ‘ended2’ siguin igual a ‘false’ (mentre no s'hagin introduït tots els elements de les dos ‘arrays’ ‘compressedX’). El bucle es divideix en dos parts: la primera, tracta les dos ‘arrays’ ‘compressedX’ i compara els seus elements; i la segona, que, en cas de ser necessari (haver introduït tots els valors de només un ‘array’ ‘compressedX’), s’afegeix els elements restants de la ‘array’ ‘compressedX’ la qual no ha pogut introduir tots els seus elements.
- Bucle amb les dos seccions:

```
while (!ended1 || !ended2) {
      if (!ended1 && !ended2) {
      ...
      }
      else {
      ...
      }
}
```

- Es compara la posició zero dels elements marcats per l’índex de les ‘arrays’ ‘expandedX’. Poden passar tres casos: que el valor de la posició zero de la ‘array’ ‘compressed1’ sigui més petit que el valor de la posició zero de la ‘array’ ‘compressed2’; l'inrevés; i que els valors siguin iguals. En cas que un és superior a l’altre, es passa l’element amb el valor més petit en la posició zero a la ‘array’ ‘compressedResultIsh’ en l’índex marcat per la variable ‘indexRIsh’. En cas que els dos valors són iguals i els valors sumats de la posició u sumen diferent de zero, es passa el valor de la posició zero en la ‘array’ ‘compressedResultIsh’ en l’índex marcat, i en la posició u se sumen els dos valors de la posició zero de les ‘arrays’ ‘compressedX’. En els tres casos s’incrementa la variable ‘indexRIsh’ per accedir en la posició següent de la ‘array’ ‘compressedResultIsh’. Les variables ‘sumIndexX’ es posen a ‘true’ sempre que es necessita actualitzar les variables ‘indexX’. Si una variable ‘sumIndexX’ està a ‘true’, es mira si es pot seguir incrementant l’índex corresponent (‘sumIndex1’ per ‘index1’, ‘sumIndex2’ per ‘index2’), i en cas d'haver arribat al final d’una ‘array’ ‘compressedX’ la variable ‘endedX’ es posa a ‘true’.

- Comparacions de valors:

```
if (compressed1[index1][DEGREE] < compressed2[index2][DEGREE]) {
	compressedResultIsh[indexRIsh][DEGREE] = compressed1[index1]
[DEGREE];
	compressedResultIsh[indexRIsh][COEFFICIENT] = compressed1[index1]
[COEFFICIENT];
	indexRIsh++;
	sumIndex1 = true;
}
else if (compressed2[index2][DEGREE] < compressed1[index1][DEGREE]) {
	compressedResultIsh[indexRIsh][DEGREE] = compressed2[index2]
[DEGREE];
	compressedResultIsh[indexRIsh][COEFFICIENT] = compressed2[index2]
[COEFFICIENT];
	indexRIsh++;
	sumIndex2 = true;
}
else if (compressed1[index1][DEGREE] == compressed2[index2][DEGREE]) {
	if (compressed1[index1][COEFFICIENT] + compressed2[index2]
[COEFFICIENT] != 0) {
		compressedResultIsh[indexRIsh][DEGREE] = compressed1[index1]
[DEGREE];
		compressedResultIsh[indexRIsh][COEFFICIENT] =
compressed1[index1][COEFFICIENT] + compressed2[index2][COEFFICIENT];
		indexRIsh++;
	}
	sumIndex1 = true;
	sumIndex2 = true;
}
```

- Actualització d'indexs:

```
if (sumIndex1) {
	sumIndex1 = false;
	if (index1 + 1 < len1) {
		index1++;
	}
	else if (!ended1 && index1 + 1 == len1) {
		ended1 = true;
	}
}
```

```
if (sumIndex2) {
	sumIndex2 = false;
	if (index2 + 1 < len2) {
		index2++;
	}
	else if (!ended2 && index2 + 1 == len2) {
		ended2 = true;
	}
}
```

- En cas només es tingui una variable ‘endedX’ igual a ‘true’ i una altra igual a ‘false’, s’iterarà per tots els elements restants de la ‘array’ ‘compressedX’ que tingui la variable ‘endedX’ igual a ‘false’ (no s’han passat tots els seus elements a la ‘array’ ‘compressedResultIsh’) , i cada element serà passat a la ‘array’ ‘compressedResultIsh’ (se seguirà incrementant la variable ‘indexX’ corresponent de la ‘array’ ‘compressedX’). Una vegada acabat de passar tots els elements restants de la ‘array’ ‘compressedX’, es posarà la variable ‘endedX’ igual a ‘true’ per així acabar el bucle.
- Passar, si és necessari, els elements restants:

```
if (ended2 && !ended1) {
	compressedResultIsh[indexRIsh][DEGREE] = compressed1[index1]
[DEGREE];
	compressedResultIsh[indexRIsh][COEFFICIENT] = compressed1[index1]
[COEFFICIENT];
	indexRIsh++;
	if (!ended1 && index1 + 1 == len1) {
		ended1 = true;
	}
	index1++;
}
else if (ended1 && !ended2) {
	compressedResultIsh[indexRIsh][DEGREE] = compressed2[index2]
[DEGREE];
	compressedResultIsh[indexRIsh][COEFFICIENT] = compressed2[index2]
[COEFFICIENT];
	indexRIsh++;
	if (!ended2 && index2 + 1 == len2) {
		ended2 = true;
```

```
        }
        index2++;
    }
```

- Es dóna l'espai de memòria de la 'array' 'compressedResult' igual al valor de la variable 'indexRIsh'. Es passen tots els valors de la 'array' 'compressedResultIsh' a la 'array' 'compressedResult' que no tenen al mateix moment un zero en la posició zero i un zero en la posició u (la 'array' 'compressedResultIsh' per defecte té valor zero en cada posició).
- Donar espai de memòria i passar valors:

```
compressedResult = createCompressed(indexRIsh);

if (compressedResult.length > 0) {
        for (int[] element : compressedResultIsh) {
                if (!(element[DEGREE] == 0 && element[COEFFICIENT] == 0)) {
                        compressedResult[indexR][DEGREE] = element[DEGREE];
                        compressedResult[indexR][COEFFICIENT] =
element[COEFFICIENT];
                        indexR++;
                }
        }
}
```

- Finalment es retorna la 'array' 'compressedResult'.
- Retorn d'una 'array' de mida igual a zero:

```
return compressedResult;
```

ALTRES ASPECTES IMPORTANTS

- El joc de les variables 'sumIndexX' està fet per no fer repetitiu el procés d'incrementar les variables 'indexX', ja que s'ha de comprovar si es pot incrementar i també es mira si s'ha arribat al final de la 'array' 'compressedX'.

ALTRES POSSIBILITATS

- S’ha provat d’una altra forma de resoldre la suma. El mètode anterior era, crear un còpia de les ‘arrays’ entrades com a paràmetres i cada vegada que s’afegia un element en la ‘array’ ‘resultat’ s’eliminava l’element posat de la ‘array’ origen i es feia córrer tots els elements una posició menys. L’avantatge era que sempre havies de mirar en la posició zero de la ‘array’ per comparar valors, no feia falta les variables d’indexs. L’inconvenient era que hi havia molts bucles, cada vegada que es volia fer córrer un valor s’havia de córrer tota la ‘array’. Algoritme molt ineficient i lent.

extraAddTest()

INTRODUCCIÓ

- La funció ‘extraAddTest()’ executa les funcions ja creades per provar el codi. Les funcions que executa proven les funcions ‘add()’ (tant per les ‘arrays’ de tipus estès com de tipus comprimit). En aquest cas les ‘arrays’ que se'ls hi passa no són les que venien amb l’enunciat. S’han creat perquè s’ha vist que amb les ‘arrays’ de prova no estaven contemplades altres possibilitats.

DISSENY

- S’executen les funcions ‘checkAddExpanded()’ i ‘checkAddCompressed()’ amb les ‘arrays’ de nova creació.

CODI

- Es posen a prova les funcions ‘add()’ amb les ‘arrays’ de nova creació Totes les funcions ‘print...()’ només són per donar per pantalla informació extra, per fer una lectura més senzilla del resultat de les funcions ‘checkAddExpanded()’ i ‘checkAddCompressed()’.
- Les funcions ‘checkAddExpanded()’ i ‘checkAddCompressed()’ són per posar a prova les funcions ‘add()’. Aquestes funcions ja són donades amb l’enunciat.
- El codi consta de múltiples línies com següents, però canviant les ‘arrays’:

```
checkAddExpanded(EXPANDED_TEST_A_ONE, EXPANDED_TEST_A_TWO,
EXPANDED_TEST_A_RESULT);
```

- `checkAddCompressed(COMPRESSED_TEST_A_ONE, COMPRESSED_TEST_A_TWO, COMPRESSED_TEST_A_RESULT)`

ALTRES ASPECTES IMPORTANTS

- Més endavant en l'apartat 'ARRAYS' s'explica com estan estructurades les 'arrays' de nova creació.

ALTRES POSSIBILITATS

- S'havia pensat de fer una creació aleatòria de polinomis, però en ser una funció no supervisada podria portar a errors. Aquests polinomis nous serien per posar a prova les funcions 'add()'.

bestType() #1

INTRODUCCIÓ

- La funció 'bestType()' rep d'entrada una 'array' que representa un polinomi en forma estesa, i en determina si la forma de representar el polinomi de forma estesa és la millor en termes d'optimització en memòria. Retorna 'true' si ja és correcta la forma de representar el polinomi i 'false' si s'hauria de passar a forma comprimida. És una funció amb sobrecàrrega, i en aquest cas avalua un polinomi codificat de forma estesa.

DISSENY

- A partir de l'entrada es compara amb la mida que tindria en cas que el polinomi fos representat de forma comprimida. Si ocupa igual o menys espai de forma estesa que de forma comprimida, es retorna 'true', i si ocupa més espai de forma estesa que de forma comprimida, es retorna 'false'.
- Quan ocupen el mateix espai de memòria es retorna 'true' perquè es dóna prioritat a la forma estesa. I es dóna prioritat a la forma estesa perquè decreix la complexitat l'operar amb ella.

- En comparar la mida de cada una de les dos possibilitats s'utilitza la funció 'compressedSize()', que a partir d'una 'array' de tipus estès, retorna la mida que hauria de tenir una 'array' de tipus comprimit per al mateix polinomi.
- Com que en la 'array' de tipus comprimit té dos valors dins de cada element, es multiplica la mida de la 'array' comprimida per dos.

CODI

- Paràmetres:
  - expanded → 'array' d'enters, representa un polinomi de forma estesa.
- El codi dins la funció només consta de la línia següent que al mateix moment que avalua la comparació, retorna el valor booleà de la funció mare.
- `return (expanded.length <= compressedSize(expanded) * 2);`

ALTRES ASPECTES IMPORTANTS

- Al comparar la mida de les dues 'arrays' només mira si és més gran la 'array' de tipus estès, no fa falta posar condicionals.

ALTRES POSSIBILITATS

- S'havia pensat en posar un condicional per efectuar la funció principal de la funció, però per netedat del codi s'ha suprimit el condicional, i s'ha posat tot en una sola línia, al mateix moment que retorna el valor.

bestType() #2

INTRODUCCIÓ

- La 'funció bestType()' rep d'entrada una 'array' que representa un polinomi en forma comprimida, i en determina si la forma de representar el polinomi de forma comprimida és la millor en termes d'optimització en memòria. Retorna 'true' si ja és correcta la forma de representar el polinomi i 'false' si s'hauria

de passar a forma estesa. És una funció amb sobrecàrrega, i en aquest cas avalua un polinomi codificat de forma comprimida.

DISSENY

- A partir de l'entrada es compara amb la mida que tindria en cas que el polinomi fos representat de forma estesa. Si ocupa menys espai de forma comprimida que de forma estesa, es retorna 'true', i si ocupa igual o més espai de forma comprimida que de forma estesa, es retorna 'false'.
- Quan ocupen el mateix espai de memòria es retorna 'false' perquè es dóna prioritat a la forma estesa. I es dóna prioritat a la forma estesa perquè decreix la complexitat l'operar amb ella.
- Al comparar la mida de cada una de les dos possibilitats, s'utilitza la funció 'expandedSize()', que a partir d'una 'array' de tipus comprimit, retorna la mida que hauria de tenir una 'array' de tipus estès per al mateix polinomi.
- Com que en la 'array' de tipus comprimit té dos valors dins de cada element, es multiplica la mida de la 'array' comprimida per dos.

CODI

- Paràmetres:
  - compressed → 'array' de 'arrays' d'enters, representa un polinomi de forma comprimida.
- El codi dins la funció només consta de la línia següent que al mateix moment que avalua la comparació, retorna el valor booleà de la funció mare.
- `return (compressed.length * 2 < expandedSize(compressed));`

ALTRES ASPECTES IMPORTANTS

- Al comparar la mida de les dues 'arrays' només mira si és més petita la 'array' de tipus comprimit, no fa falta posar condicionals.

ALTRES POSSIBILITATS

- S’havia pensat en posar un condicional per efectuar la funció principal de la funció, però per netedat del codi s’ha suprimit el condicional, i s’ha posat tot en una sola línia, al mateix moment que retorna el valor.

ARRAYS

Les ‘arrays’ creades per provar amb més profunditat les funcions ‘add()’ es separen en dos conjunts: les de tipus estès i les de tipus comprimit. Cada conjunt està diferenciat per una lletra de l’abecedari: ‘A’, ‘B’, ‘C’. S’agrupen en conjunts formats per tres ‘arrays’: ‘ONE’, ‘TWO’ i ‘RESULT’. Cada ‘array’ representa un polinomi de la seva forma corresponent. Si es suma una ‘array’ ‘ONE’ amb la ‘array’ ‘TWO’ del mateix conjunt, el resultat serà la ‘array’ ‘RESULT’ del conjunt.

ex.:

- `public int[] EXPANDED_TEST_A_ONE = new int[]{ ... }`
- `public int[] EXPANDED_TEST_C_RESULT = new int[]{ ... }`
- `public int[][] COMPRESSED_TEST_B_TWO = new int[][]{ ... }`
- `public int[][] COMPRESSED_TEST_D_RESULT = new int[][]{ ... }`

CONCLUSIÓ

El codi de la pràctica ha estat estructurat de la forma més clara possible. El nivell no és molt alt, però, s'ha intentat fer el codi eficient i net. Algunes funcions implementades en el codi podrien ser reemplaçades per funcions pròpies del sistema ja optimitzades, com la funció 'System.arraycopy()' (copia els elements d'una 'array' a una altra.).

El codi ha sigut canviat vàries vegades per buscar la netedat d'ell mateix i ser menys complex (sobretot les funcions 'add()').

Les funcions 'bestType()' són una aportació de l'alumne. En cas que es volgués optimitzar l'espai de memòria de les 'arrays' dels polinomis en un futur, és una eina més.

Les funcions 'createExpanded()' i 'createCompressed()' retornen solament l'espai de memòria de les 'arrays', no retornen una 'array' ja declarada (és innecessari).