Universitat de Lleida

Escola Politècnica Superior

Grau en Enginyeria Informàtica

Programació II

Juan Manuel Gimeno Illa

Pràctica 1
Polinomis

Laura Haro Escoi

49260219S

GPraLab2

Diumenge 28 de març de 2021

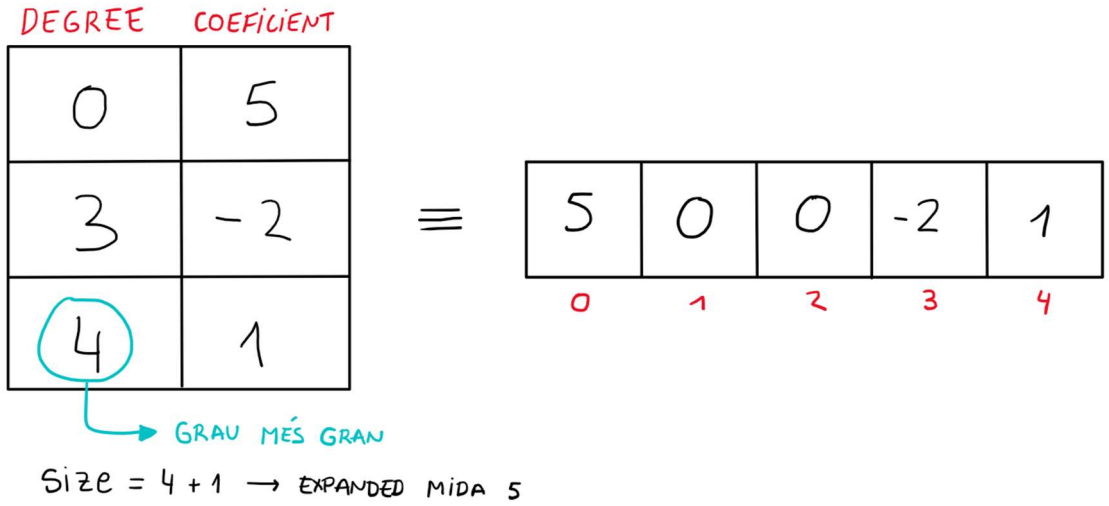
Índex

Funcions de càlcul de les mides

1. public int expandedSize (int[][] compressed)

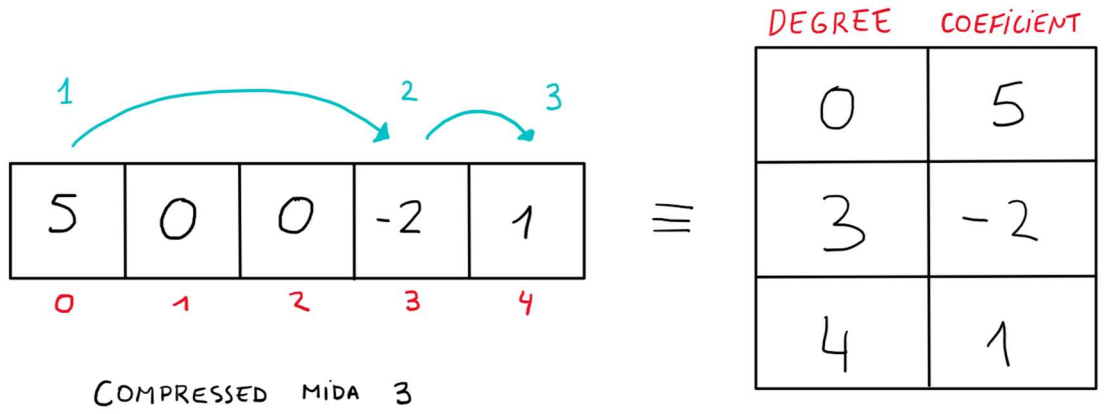
Per a resoldre aquesta funció, he agafat directament el grau més gran de la taula i li he sumat 1 per a que em doni la mida que hauria de tenir el *expanded*.

En aquest cas, està situat a l'últim element de la primera columna del *compressed*.



2. public int compressSize (int[] expanded)

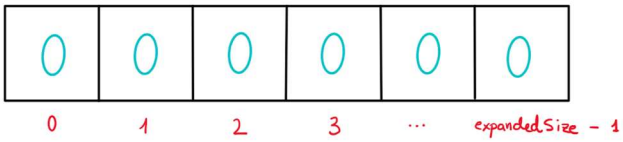
En aquest cas, he utilitzat un comptador per a que s'incrementi cada vegada que el nombre llegit sigui diferent de 0. El valor final d'aquest serà la mida que hauria de tenir el *compressed*.



Funcions de creació

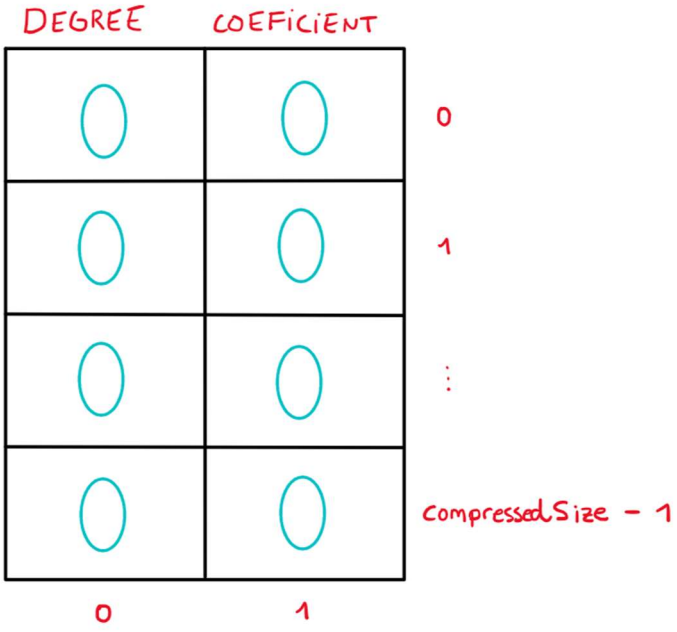
3. public int[] createExpanded (int[] expanded)

He creat una taula d'una dimensió amb la mida del *expanded* que introduïm plena de zeros.



4. public int[][] createCompressed (int[][] compressed)

He creat una taula de dues dimensions plena de zeros. El nombre de files seria la mida del *compressed* que introduïm i el nombre de columnes seria 2: una per als graus del polinomi i l'altra per als coeficients.

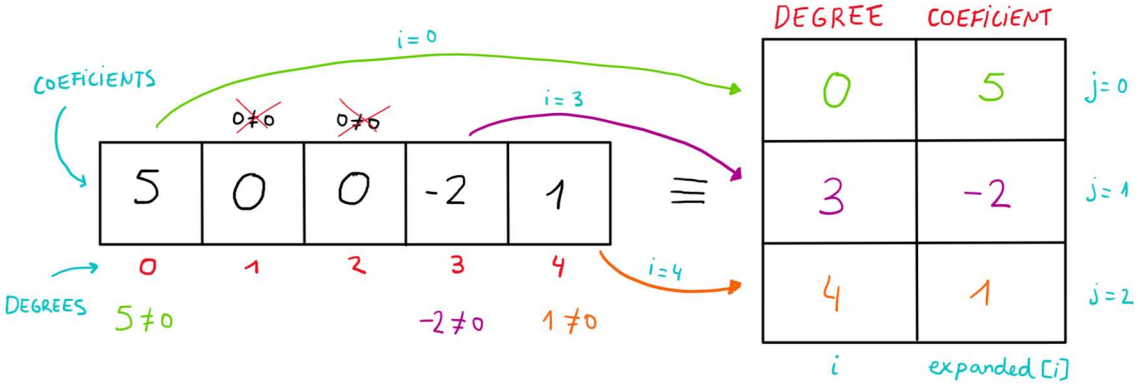


Funcions de còpia

5. public void copyTo (int[] fromExpanded, int[][] toCompressed)

Per a poder passar un polinomi de *expanded* a *compressed* he utilitzat un bucle que vagi emplenant la taula de dues dimensions amb els nombres de les caselles del *expanded* que no siguin iguals a 0.

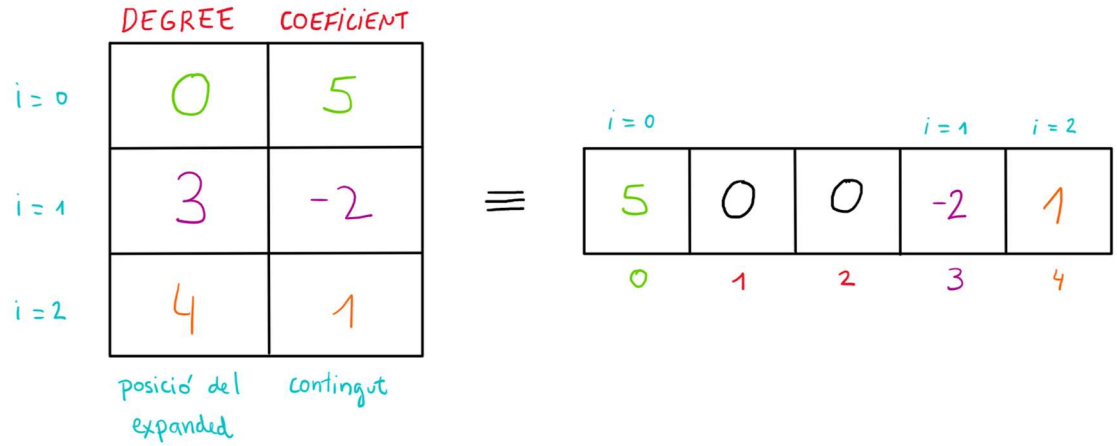
Diguem que la variable que utilitzem per a recórrer la taula de *expanded* és **i**; si el nombre que es troba en la posició **i** compleix la condició, copiarem aquest nombre a la columna dels coeficients i el valor de **i** a la dels graus utilitzant una variable auxiliar **j** per a poder emplenar la taula del *compressed.*



6. public void copyTo (int[][] fromCompressed, int[] toExpanded)

Per a poder passar un polinomi de *compressed* a *expanded* he utilitzat un bucle que vagi recorrent la taula del *compressed* i que copiï el coeficient de cada fila a la posició corresponent de la taula del *expanded.* Aquesta posició ha de ser la que marqui el grau de la fila que estem llegint en el moment.

En aquest diagrama podem veure com es col·loca el coeficient a la posició on el nombre del grau indica. Si el grau es 3, anirà a la posició 3 de la taula *expanded.*



Funcions de conversió

7. public int[][] compress (int[] expanded)

Per a fer aquest exercici he utilitzat les funcions *createCompressed(), compressedSize()* i la *copyTo()* creades anteriorment.

Amb la funció CreateCompressed() creo un *compressed* de la mida que determini la funció compressedSize() (el qual estarà ple de zeros), passant-li el *expanded* com a paràmetre. Finalment aquest el passo com a paràmetre junt amb el *expanded* a la funció copyTo().

8. public int[] expanded (int[][] compressed)

Per a fer aquest exercici he utilitzat les funcions *createExpanded(), ExpandedSize()* i la *copyTo()* creades anteriorment.

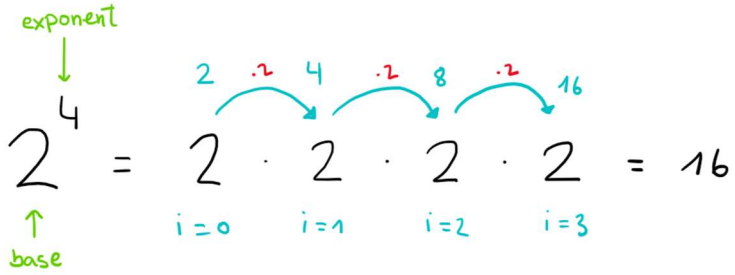
Amb la funció CreateExpanded() creo un *expanded* de la mida que determini la funció expandedSize() (el qual estarà ple de zeros), passant-li el *compressed* com a paràmetre. Finalment aquest el passo com a paràmetre junt amb el *compressed* a la funció copyTo().

Funcions d'evaluació

9. public int pow (int base, int exponent)

Aquest exercici es pot resoldre fàcilment utilitzant un bucle que multipliqui el nombre *base* per si mateix tantes vegades com l'*exponent* indiqui.

Per exemple, si de base tenim 2 i d'exponent tenim un 4, el programa multiplicarà el 2 per si mateix 4 vegades.

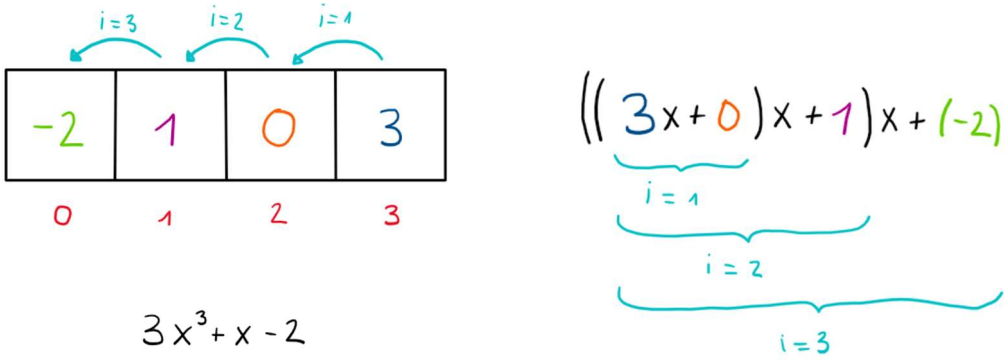


10. public int evaluate (int[] expanded, int x)

Per a resoldre aquest problema m'he ajudat de l'*algoritme de Horner.* Aquest es el següent:

$$p(x) = a_0 + x(a_1 + x(a_2 + \cdots x(a_{n-1} + a_n x)\cdots))$$

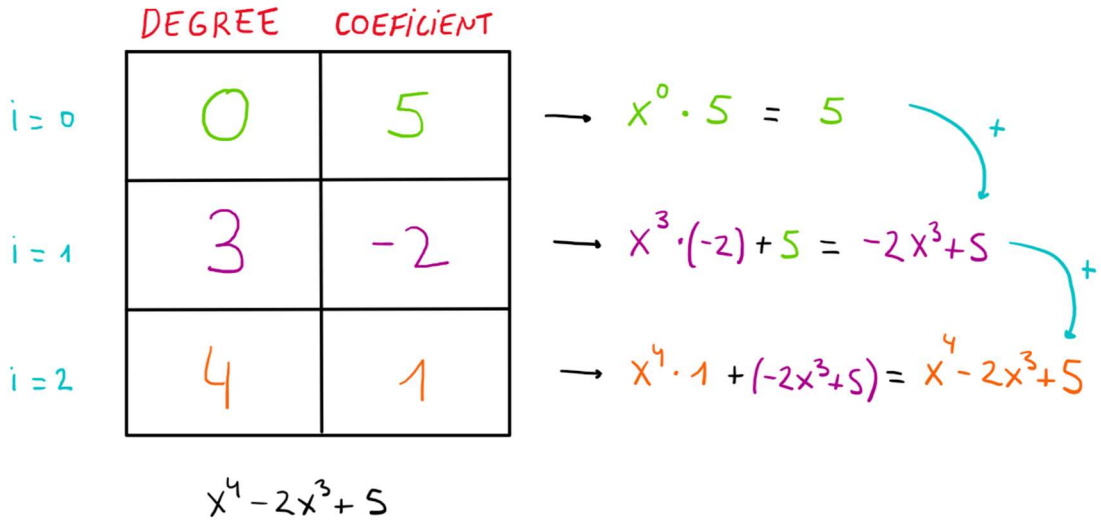
Per a poder aplicar-lo en programació la millor manera es començar pel final d'aquest. Fent això podem fer aquest algoritme amb un bucle for. Aquí un exemple:



11. public int evaluate (int[][] compressed, int x)

En aquest exercici, a diferencia de l'anterior he utilitzat la funció pow() creada anteriorment.

Per a poder implementar-ho he utilitzat un bucle on he recorregut la taula i he aplicat la funció pow() passant com a paràmetre la x per a la base i el grau de la casella en la que hem trobo com a exponent. Després aquest resultat el multiplico amb el coeficient corresponent.

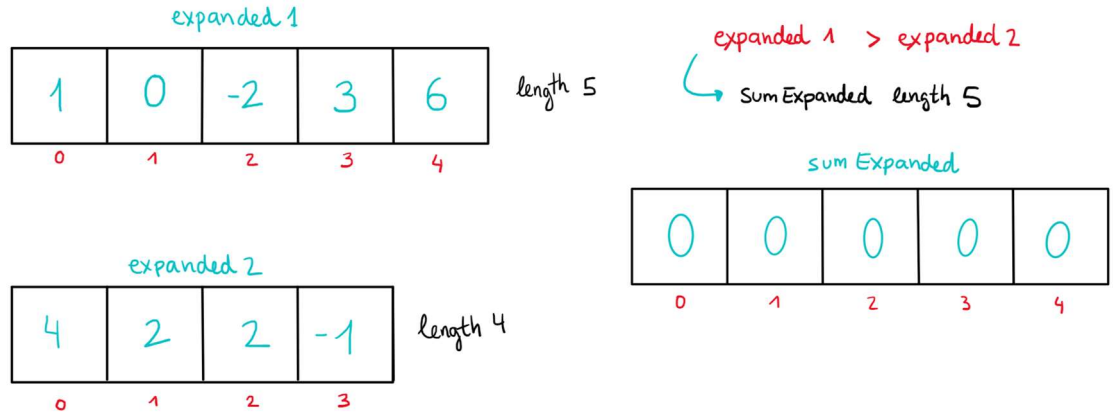


Funcions de suma de dos polinomis

12. public int[] add (int[] expanded1, int[] expanded2)

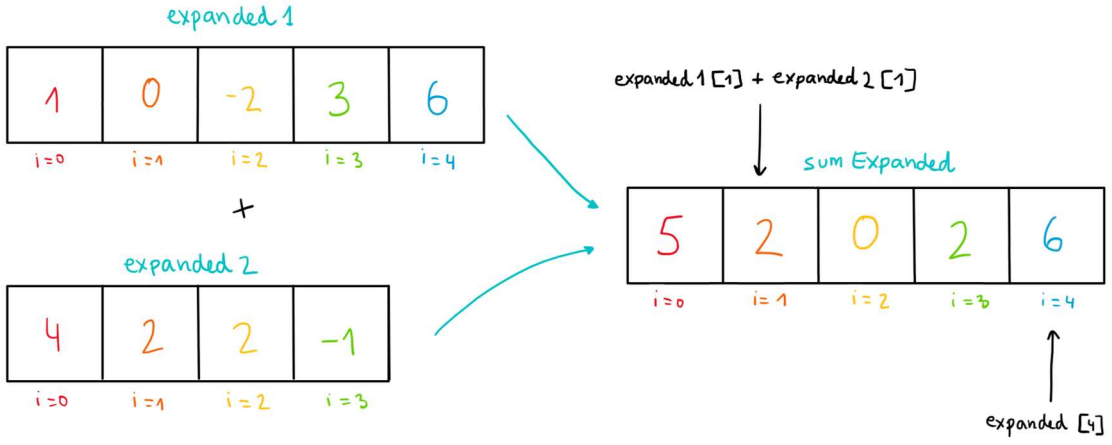
La resolució d'aquest exercici la he dividit en tres parts diferents mitjançant una sèrie de funcions auxiliars.

La primera es tracta de trobar la mida per al resultat de la suma de polinomis en forma *expanded*. Es possible que ens arribin *expandeds* de mides diferents, per tant, he fet una funció anomenada *checkExpandedSize.* A aquesta funció li passo per paràmetre dos *expandeds* i em retorna un de nou amb la mida del *expanded* més gran. D'aquesta manera m'asseguro que el polinomi suma (*sumExpanded)* tindrà la mida adequada per a poder dur a terme l'operació correctament.



En la segona part es on es fa la suma dels polinomis com a tal. Aquesta consta d'un bucle for que va recorrent el *sumExpanded* mitjançant una variable auxiliar que li direm **i.** Aquesta marcarà la posició en la que ens trobem a l'hora de fer la suma tant al *sumExpanded* com als *expandeds* que sumem.

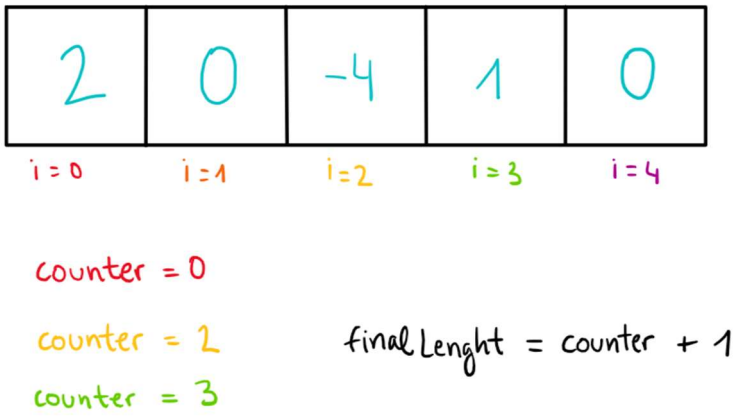
Cada vegada que iteri el for, mitjançant una sèrie de condicions, es sumaran els termes que tinguin la mateixa posició i es col·locaran a la posició del *sumExpanded* en la que ens trobem.



Com es pot apreciar a l'exemple, una vegada el programa no te per sumar dos nombres que tinguin la mateixa posició, col·loca el que correspongui directament al *sumExpanded.* Si no fiquem aquestes condicions, quan els *expandeds* tinguin mides diferents, el for es sortiria de rang i ens donaria *index out of bounds.*

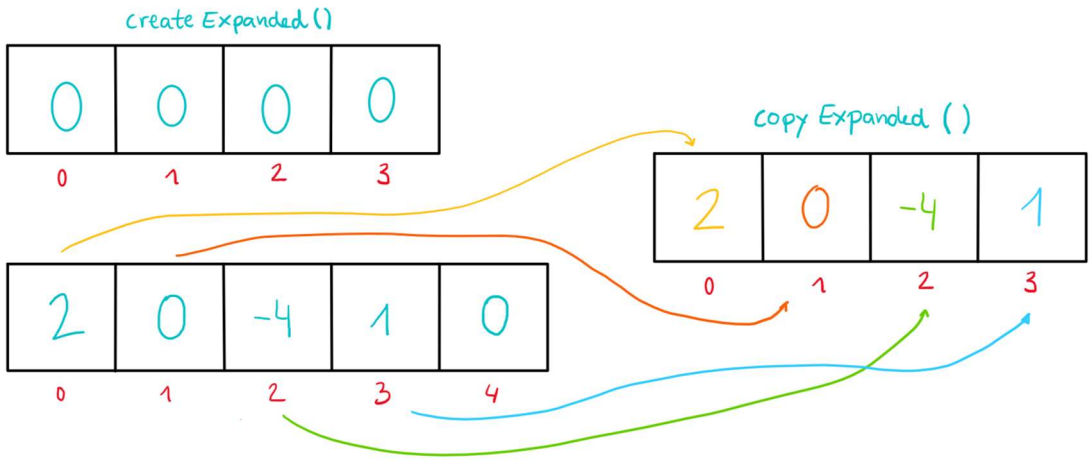
Finalment, la ultima part de la funció "corregeix" el polinomi final per a que no quedin zeros innecessaris. Per a dur això a terme he creat una funció anomenada *correctExpanded()* en la que li passo un *expanded* (que no te perquè ser correcte) i em retorna un de nou rectificat.

Aquesta funció utilitza un for principalment per a poder comprovar quina es l'última posició en la que em trobat un nombre diferent de 0. Per a fer això, he utilitzat una variable com a comptador que emmagatzema el valor de la **i** que utilitza el for quan el numero de la posició es diferent a 0. D'aquesta manera, una vegada el for acabi, el comptador tindrà el valor de última posició on ha trobat un valor vàlid.



Aquest comptador ens servirà per a determinar la llargada que haurà de tenir el *expanded* correcte. A continuació crearem un *expanded* amb la funció *createExpanded()* passant-li com a paràmetre el comptador + 1.

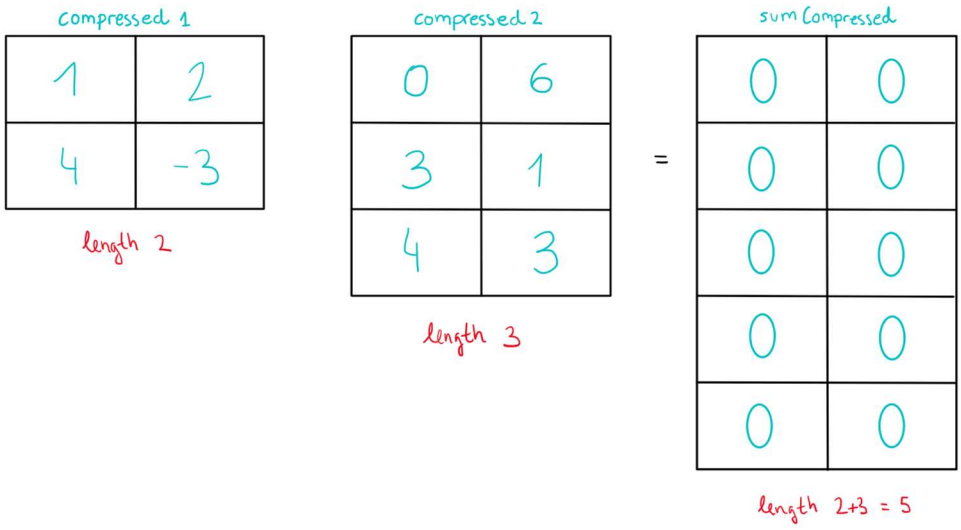
Finalment copiarem el *expanded* que teníem com a paràmetre de la funció al nou que hem creat, amb la funció *copyExpanded()* que he creat, i el retornarem.



13. public int[][] add(int[][] compressed1, int[][] compressed2)

Finalment, l'últim exercici de la pràctica. Per a poder fer un codi nítid i entenedor he fet us de vàries funcions, ja que la suma de *compresseds* es més complexa que els altres exercicis.

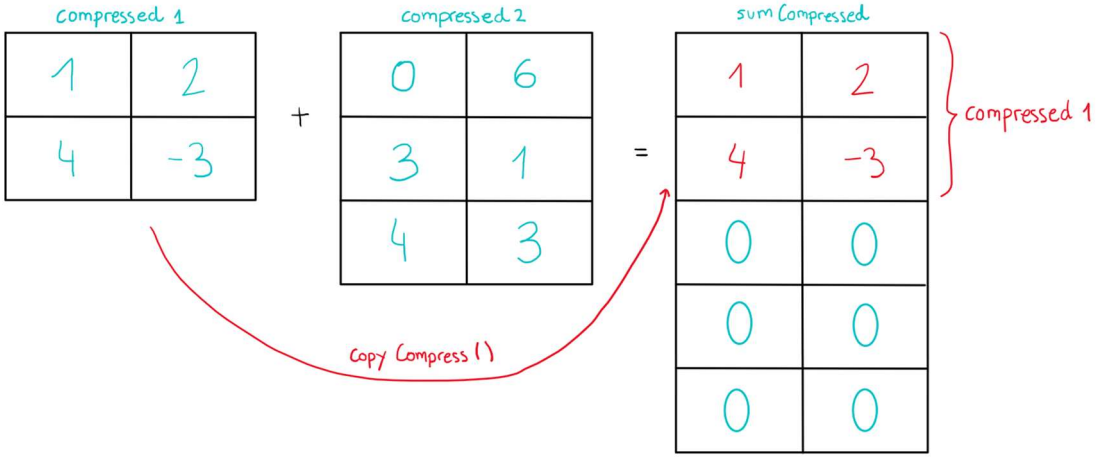
Primerament, per a poder fer la suma guardaré tots els resultats a una nova matriu que he anomenat *sumCompressed* la qual, per a evitar problemes, tindrà de mida la suma de les dues mides de les matrius que sumem. Per exemple, si tenim un *compressed1* de mida 2 i un *compressed2* de mida 3, la mida de *sumCompressed* serà 5.



A partir d'aquí, fem la primera comprovació per a saber si alguna de les dues matrius que estem sumant, està buida. Si aquesta condició es compleix, la matriu "no buida" es copia directament amb una funció anomenada *copyCompressed*(), la qual copia tot el contingut d'una matriu, a una altra.

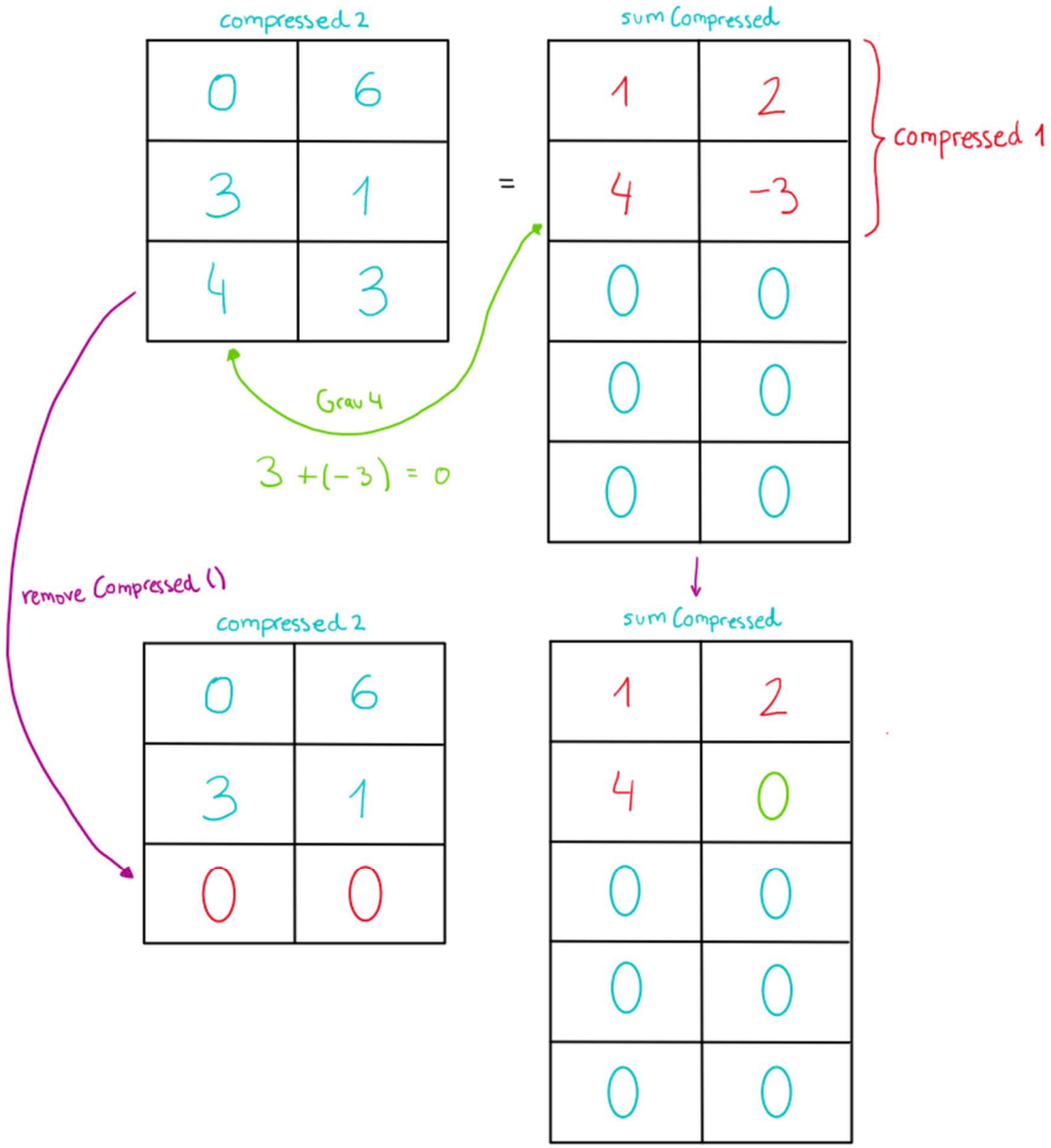
En el cas de que cap de les dues sigui nul·la, el programa farà el següent:

Primer copiarà el *compressed1* a la matriu *sumCompressed* i a partir d'aquí anirà copiant element a element el *compressed2* mitjançant una sèrie de condicions per a que el resultat quedi perfectament ordenat i sense elements repetits. De moment, *sumCompressed* encarà no serà un polinomi vàlid, ja que es possible que hi hagi camps nuls que s'hagin d'eliminar.



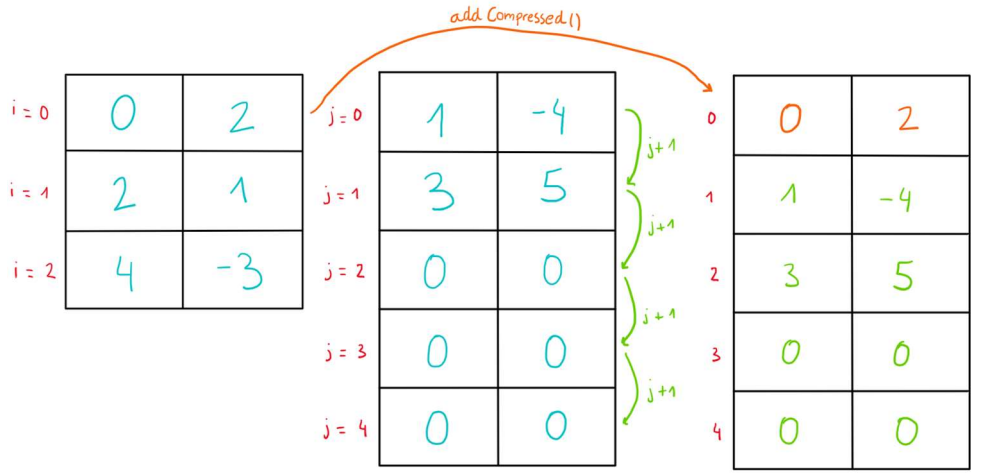
Per a poder copiar els elements del *compressed2* el primer que faré serà crear una copia per a poder modificar-la sense que afecti als paràmetres de la funció principal. A partir d'aquesta podré operar sense problemes amb les funcions següents.

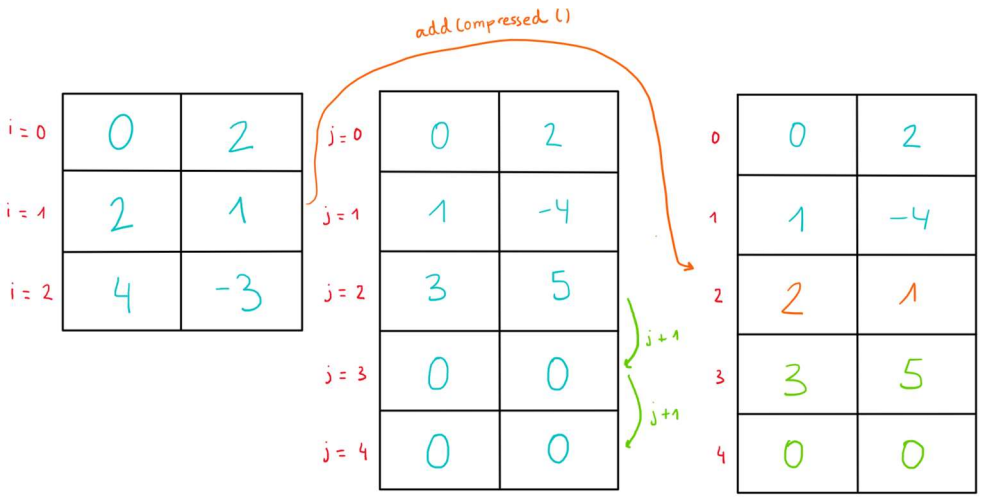
Primer sumaré els elements de *compressed2* que tinguin el mateix grau que algun dels elements de *compressed1.* La funció encarregada d'això es la addCompressedSameDegree(). El que fa es sumar l'element a la matriu *sumCompressed* que tingui el mateix grau i despres eliminar aquest de la matriu *compressed2* amb ajuda de la funció removeCompressed().

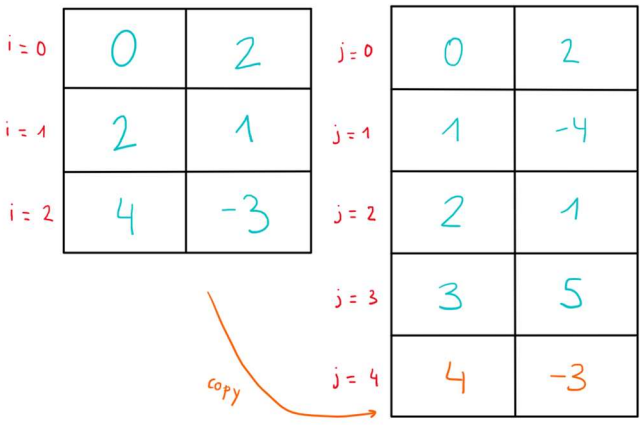


Després de sumar els que tenen el mateix grau, aplico una altra funció anomenada *addCompressedDifferentDegree(),* la qual s'encarrega de col·locar els elements que queden de *compressed2* a *sumCompressed* de manera que quedi ordenat de menor a major els graus.

Per a fer això utilitzo un for per a recórrer *compressed2* i després un while per a determinar quan l'element en el que ens trobem amb el for ha estat sumat a *auxCompressed.* Dins d'aquest es troben tres condicions principals: que el numero a sumar vagi a la primera posició de la taula, que vagi entre mig de dos números, o que vagi al final de la taula. A part d'aquestes tres condicions principals, per a que els zeros del final no afectin, hi ha una última condició dins del else.







Finalment tenim la funció *correctCompressed()* a la qual se li passa un polinomi que no te perquè ser vàlid com a paràmetre i retorna el polinomi corregit. Per a fer-ho, primer recorre la matriu i compta quants resultats vàlids hi ha (que no siguin [0, 0] o [num, 0]). A part d'això, si troba una posició [num, 0], la borra amb *removeCompressed().*

Amb el numero del comptador anterior, creo una nova matriu amb mida num – 1 i copio la matriu anterior a aquesta.