Èric Bitrià Ribes Grup 1

DNI: 49755704B

Abstract

Documentació i el perquè de les funcions implementades

**Pràctica I: BIGNATURALS**

Operar amb nombres de llargada “Il·limitada”

Índex

[1. Funció zero() i one() 1](#_Toc129198732)

[2. Funció equals() 1](#_Toc129198733)

[3. Funció add() 2](#_Toc129198734)

[4. Funció shiftLeft() 3](#_Toc129198735)

[5. Funció multiplyByDigit() 4](#_Toc129198736)

[6. Funció multiply() 5](#_Toc129198737)

[7. Funció arrayToInt() 6](#_Toc129198738)

[8. Funció copy() 7](#_Toc129198739)

[9. Funció factorial() 7](#_Toc129198740)

[10. Funció fibonacci() 8](#_Toc129198741)

# 1. Funció zero() i one()

Simplement retornen un array d’una posició que conté un 1 i un 0, respectivament.

# 2. Funció equals()

Aquesta funció compara dos vectors d'enters (d’ara endavant tots els nombres que utilitzem tindran aquest format) i torna un booleà indicant si són iguals o no.

**Explicació codi:**

if(number1.lenght != number2.lenght){

equal=false;

}

Primer comprovem que la llargada dels arrays sigui la mateixa, evidentment si no ho són no seran iguals i atribuïm a equal false.

}else{

while(i<number1.length && equal){

if(number1[i]==number2[i]){

i++;

}else{

equal=false;

}

}

}

return equal;

En cas que tinguin la mateixa mida comprovem dígit per dígit que siguin iguals entre ells, en cas contrari equal serà false sortirà del while i retornarà false.

# 3. Funció add()

La funció add obtindrà dos nombres i retornarà la seva suma. Per realitzar la suma s’ha plantejat la situació on el nombre amb més dígits estarà a la posició top i el que menys a la posició bot de la forma:

**Explicació codi:**

Per això crearem dos nous arrays on copiarem cada nombre d’entrada en funció de la seva llargada amb la funció copy():

int[] top = (num1.length >= num2.length) ? copy(num1) : copy(num2);

int[] bot = (num1.length >= num2.length) ? copy(num2) : copy(num1);

Una vegada tenim situats els números procedirem a crear un nou array que emmagatzemarà el resultat final, aquest tindrà de llargada màxima la llargada del nombre “top” +1 en cas que la suma amb el nombre “bot” tingui algun carry.

int[] sum = new int[top.length+1];

A continuació es realitza un bucle “for” per recórrer totes les posicions tant del top com el bot, en cas que la suma d’aquests dos més el possible carry que hi pugui haver excedeixi 9 li restarem 10 a aquest nombre i sumarem un 1 a la posició següent.

for(int i=0;i<top.length;i++){

if(top[i]+bot[j]+sum[i]>9){

sum[i] += (top[i]+bot[j])-10;

sum[i+1]+=1;

}else{

sum[i] += top[i]+bot[j];

}

Les posicions del bot[ ] no tenen que coincidir amb el top per això crearem una variable addicional “j” que recorrerà el “bot”. Si s’arriba al punt que ja no existeixen més posicions que sumar amb el “top” atribuirem a la ultima posició, que ja no utilitzarem, un 0 i continuarem fent la suma.

if(j<(bot.length-1)){

j++;

}else if(j==(bot.length-1)){

bot[j]=0;

}

Finalment tenim dos casos: on el nombre de posicions del nombre final “sum” hagi estat completament ocupat, en aquest cas retornem el nombre “sum”, o ens hagi sobrat una posició, en aquest cas crearem un nou array “sum\_final” amb la llargada correcta i copiarem “sum” a “sum\_final” per retornar-lo.

if(sum[top.length]==0){

int[] sum\_final = new int[top.length];

for(int i=0;i<top.length;i++){

sum\_final[i]=sum[i];

}

return sum\_final;

}else{

return sum;

}

**Altres possibilitats:**

Per no dependre de quina posició “top” o “bot” ocupa cada nombre es podria realitzar de manera que el tots els nombres es complissin amb 0 fins la posició del nombre més gran +1, juntament amb un array nou que l’anomenarem carry. Seguint el mateix procediment que el anterior sumariem tots tres nombres i finalment si ens sobres una posició la eliminaríem. Però a l’hora d’implementar-la em va parèixer més llarga i amb més arrays així que es va descartar.

# 4. Funció shiftLeft()

Aquesta funció te l’objectiu de retornar un nombre mogut en un determinat nombre de posicions, el que es pot traduir en una multiplicació per 10 un cert nombre de vegades.

**Explicació codi:**

Per la creació de shfitLeft requerirem crear un nou array que contingui tantes posicions com les que volem moure, així com les del nombre original:

int[] shifted = new int[number.length + positions];

Una vegada tenim definit l’array final amb dos bucles for, copiarem el array original i emplenarem amb el nombre de 0 que ens doni el paràmetre d’entrada:

for (int i = 0; i < positions; i++) {

shifted[i] = 0;

}

for (int j = 0; j < number.length; j++) {

shifted[j + positions] = number[j];

}

Si el nombre que rebem es simplement el zero, retornem el zero un altre cop sense requerir moure les posicions.

If(equals(number,zero())){

Return(zero());

}

# 5. Funció multiplyByDigit()

Com el seu nom indica aquesta funció multiplicarà un nombre per un dígit, on tots dos seran els valors d’entrada.

**Raonament:**

Com que sabem que la multiplicació per un sol dígit és la suma del nombre les vegades que ens digui el dígit, per exemple:

**Explicació Codi:**

Tenint en compte que estem utilitzant la mateixa operació “suma” un nombre de vegades simplement implementem la multiplicació per un dígit com un bucle de la funció add():

public int[] multiplyByDigit(int[] number, int digit) {

int[] multiplied = new int[]{0};

for(int i=0;i<digit;i++){

multiplied = add(multiplied,number);

}

return multiplied;

}

On el resultat guardat en el nombre multiplied s’anirà constantment actualitzant en funció del nombre de vegades que li digui el paràmetre dígit.

# 6. Funció multiply()

La funció multiply multiplicarà dos nombres qualsevols i retornarà el resultat.

**Raonament:**

Aquesta funció presenta moltes similituds amb la funció multiplyByDigit(), però aquesta vegada utilitzarem més dígits, per això podem reutilitzar-la i aplicar la multiplicació com es faria en una multiplicació manual:

Si ens fixem la multiplicació es composa de dues parts: La multiplicació per cada dígit on cada dígit nou que multipliquem tindrà el seu resultat desplaçat una posició marcat per la fletxa . I la segona part és la suma on es simplement sumar cada nombre multiplicat pel dígit i desplaçat.

Amb aquest raonament també ens hem de fixar que el nombre resultant de nombres multiplicats per cada dígit és el mateix que la llargada del segon nombre que multipliquem, és a dir per al 45 hem utilitzat 2 files, si haguéssim utilitzat el 234 hauríem requerit 3 files.

**Explicació del codi:**

Tenint en compte la última explicació situem el nombre amb més llargada a la posició top i l’altre a la posició bot, d’aquesta manera no necessitarem crear més arrays dels necessaris.

Una vegada tenim situats els nombres crearem una matriu que emmagatzemarà tots els resultats de la multiplicació per dígits on el nombre de files vindrà determinat per la llargada del nombre “bot”.

int[] top = (number1.length >= number2.length) ? copy(number1) : copy(number2);

int[] bot = (number1.length >= number2.length) ? copy(number2) : copy(number1);

int[][] multiplied = new int[bot.length][];

Ja tenim on guardar cada dada ara solament hem d’operar cada dígit del nombre “bot” i moure una posició cada vegada que canviem de dígit utilitzant la funció shiftLeft();

for(int i=0;i<bot.length;i++){

multiplied[i] = multiplyByDigit(top, bot[i]);

multiplied[i] = shiftLeft(multiplied[i],i);

}

Ara que ja tenim una matriu amb els nombres ordenats simplement sumem cada fila de la matriu resultant en una nova per obtenir el resultat final de la multiplicació amb la funció add():

int[] result = new int[]{0};

for (int[] ints : multiplied) {

result = add(ints, result);

}

return result;

# 7. Funció arrayToInt()

**Raonament:**

Aquesta funció té l’únic objectiu de transformar un nombre representat amb un array en un enter, NO s’utilitza per operar en cap funció anterior simplement ens serveix com a índex per realitzar bucles en les funcions fibonacci i factorial de forma més eficient sense tenir que recorre a treballar amb decrements del nombre original en format array.

**Explicació del codi:**

Tenint en compte que cada posició del array representa les unitats, desenes, centenars (múltiples de 10), creem una variable anomenada índex que representarà aquesta posició.

Seguidament amb un índex “i” que recorrerà la llargada del nombre podrem anar multiplicant cada dígit per la seva posició en funció del múltiple de deu que ocupa i sumar el resultat final per a retornar així el nombre en format d’enter (int).

public int arrayToInt(int[] num){

int[] decrement = copy(num);

int sum=0;

int index=1;

for(int i=0;i<num.length;i++){

sum += decrement[i]\*index;

index \*= 10;

}

return sum;

}

**Altres maneres de fer-ho:**

Operar directament sobre el nombre en format array i anar restant una posició i sumant 1 cada vegada fins arribar a zero, gran inconvenient operar amb carrys més llargada més ineficient etc.

# 8. Funció copy()

No te gran misteri simplement donat un array d’entrada retorna la seva copia.

public int[] copy(int[] num){

int[] copied = new int[num.length];

for(int i=0;i<num.length;i++){

copied[i]=num[i];

}

return copied;

}

# 9. Funció factorial()

**Raonament:**

La funció factorial calcula el factorial del nombre d’entrada, definim el factorial d’un nombre com la multiplicació d’aquest per el mateix nombre menys 1 fins arribar a 1:

Seguint aquesta lògica s’ha plantejat realitzar un bucle “for” començant des de l’1 fins al nombre d’entrada n, però com que el nombre d’entrada està representat en arrays cridarem a la funció arrayToInt per convertir el nombre inicial en el final del nostre índex.

**Explicació del codi:**

D’aquesta manera dins del bucle “for” podem implementar la funció multiplyByDigit on el dígit serà el índex d’1 fins a n del nombre d’entrada:

public int[] factorial(int[] number) {

int[] factorial = new int[]{1};

for(int i=1;i<=arrayToInt(number);i++){

factorial = multiplyByDigit(factorial,i);

}

return factorial;

}

# 10. Funció fibonacci()

La funció Fibonacci calcula la successió de Fibonacci que és la suma dels dos elements anteriors un a l’altre de forma continua:

D’on els valors inicials són 0,1 i queda de la forma: 0,1,1,2,3,5,8,13...

**Raonament:**

Com que els valors inicials sempre seran 0 i 1, podem definir dos arrays amb les funcions zero i one. I un altre array per emmagatzemar el resultat.

Utilitzant la funció arrayToInt definirem el índex final del bucle que realitzarà la suma dels arrays fins al nombre desitjat.

**Explicació del codi:**

Primer definim els arrays sobre els que operarem:

int[] prev = new int[]{0};

int[] post = new int[]{1};

int[] fibonacci = new int[]{0};

Cada vegada que es realitza una suma els valors previs incrementaran de posició en un, és a dir si fem Fibonacci = add(prev,post), la següent suma serà post + Fibonacci = Fibonacci i així successivament. De manera que es pot implementar com un bucle on atribuïm els valors anteriors i posteriors entre les variables amb la funció copy():

for(int i=0;i<arrayToInt(number);i++){

fibonacci = add(prev,post);

post = copy(prev);

prev = copy(fibonacci);

}

Finalment quan s’acabi el bucle retornarem la successió de Fibonacci resultant.

**Altres maneres d’implementació:**

Mitjançant la creació d’una funció sub() on donats dos arrays obtindrà la resta i així operar com si fos un índex, en aquest cas seria tant simple com anar restant el array original 1 cop fins que s’arribi a 0. Aquesta implementació de la funció sub() també serviria per la funció factorial, però com que al convertir l’array en un int no estem operant sobre ell i per tant no estem infringint l’objectiu de la pràctica he decidit implementar-ho d’aquesta manera.