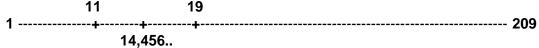
MÒDUL M03

Exercicis de comprensió dels nombres primers

Aquesta vegada va de primers. Es demana fer els exercicis que es demanen, tots ells relacionats amb els nombres primers. Els primers son aquells que no tenen altres divisors que ells mateixos i la unitat. Es a dir, si un nombre n és primer, tindrà únicament dos divisors (1 i n), i si no ho es, en tindrà més. Els divisors d'un nombre n, normalment van en parelles, per cada divisor que hi ha per sota de l'arrel quadrada de n, hi ha d'haver un altre per sobre. Si n1 és un divisor de n tal que n1 < Math.sqrt(n), ha d'existir un altre n2 > Math.sqrt(n) tal que n1*n2 = n. En el límit, si n és un quadrat perfecte, tindrà un divisor just en la seva arrel quadrada. Si n no es un quadrat perfecte, el nombre de divisors de n ha de ser parell, i si ho es, serà senar. Fixeu-vos a l'exemple numèric que s'indica. Posem el cas del 209, té com a divisors trivials el 1 i el 209. La seva arrel quadrada és Math.sqrt(209) = 14,45683229...



El 209 no és primer, apart del 1 i el 209, té com a divisors el 11 i el 19 (els dos primers per cert). I llavors 11 * 19 = 209. Per saber si un número n és primer, no cal buscar divisors per sobre de la seva arrel quadrada, ja que si hem arribat a la seva arrel quadrada i no hem trobat divisors, per sobre de la seva arrel quadrada només hi ha el divisor trivial n (1 * n = n), ja que si hi hagués un altre divisor n1, hauria d'existir també un altre n2, i estant els dos per sobre de l'arrel quadrada de n, llavors n1 * n2 > n (el producte seria major a n). A l'exemple, per saber si 209 és primer, només caldria que busquéssim divisors fins a 14.

1º) – Definir una funció esPrimer(n), a la que passarem un valor n (enter positiu, de tipus long) i retornarà true si n és primer, i false en cas contrari. Recordeu optimitzar la funció perquè el temps d'execució sigui petit. Un exemple de funció que fa això (en python), i que no està gens optimitzada, la podem per exemple trobar a (llegiu els comentaris també):

https://www.lawebdelprogramador.com/codigo/Python/2413-Determinar-si-un-numero-es-primo-o-no.html

```
El codi per provar la funció esPrimer(n) podria ser:
```

2°) – A partir de la funció anterior, feu una funció **vectorPrimers(n)** a la que passem un valor **int n**, i retorna un vector amb els **n** primers nombres primers. Calculeu el temps que inverteix la vostra funció en retornar (sense escriure'ls a la consola) una llista dels primers **100000** primers i compareu-la amb la de l'exemple.

```
public static void main(String[] args) {
   long t = System.currentTimeMillis();
   long [] v = vectorPrimers(100000);
   System.out.println("Temps invertit al càlcul: " + (System.currentTimeMillis() - t) + " milisegons");
   System.out.println("Últim primer del vector v[99999] = " + v[99999]);
}

run:
   Temps invertit al càlcul: 672 milisegons
   Últim primer del vector v[99999] = 1299709
   BUILD SUCCESSFUL (total time: 0 seconds)
```

MÒDUL M03 Realitzat per : José Javier Faro Rodríguez FULL 1 - 3

Si el que volem és un **vector** de primers, podem optimitzar encara més la nostra funció, ja que per saber si un número **n** és primer, només cal dividir entre els primers anteriors o iguals a la seva arrel quadrada, es a dir, no cal dividir entre tots els números, dividirem només entre els que son primers (els tenim al vector que anem creant).

```
public static void main(String[] args) {
    long t = System.currentTimeMillis();
    long [] v = vectorPrimersOptim(100000);
    System.out.println("Temps invertit al calcul: " + (System.currentTimeMillis() - t) + " milisegons");
    System.out.println("Últim primer del vector v[99999] = " + v[99999]);

**Galida ×*

Debugger Console × Primers (run) ×

run:

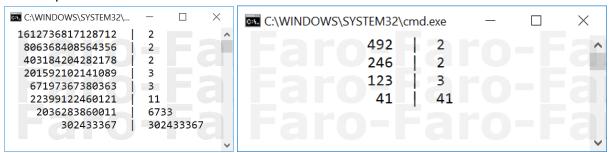
Temps invertit al calcul: 203 milisegons

Últim primer del vector v[99999] = 1299709

BUILD SUCCESSFUL (total time: 0 seconds)
```

Fixeu-vos que la funció optimitzada s'executa així tres vegades més ràpid. Calculeu el temps d'execució del vostre codi i mireu d'optimitzar-ho de la mateixa forma.

3°) – Es demana programar la funció **factoritza(n)**, que retorna un vector amb la descomposició de **n** en factors primers. La factorització de **492** per exemple és **[2,2,3,41]**. Feu també un programa per escriure el resultat a la consola formatant la sortida com s'indica als exemples:



El cost (en temps d'execució) de calcular els factors primers de **n** és considerable per a **n** gran. Clar, amb el que sabem dels números primers, veiem que aquesta funció també es pot optimitzar. Per cada factor trobat abans de l'arrel quadrada de **n**, hi haurà un altre per sobre d'aquesta, que podem extreure directament, i afegir-lo a la llista de factors si és primer.

A l'exemple, es veuen dues crides a dues funcions que fan servir algorismes diferents, i constatem la diferència considerable en el temps d'execució per a un valor de **n** donat. temps factoriza(1612736817128712L) = [2, 2, 2, 3, 3, 11, 6733, 302433367] 109 ms temps factoritza(1612736817128712L) = [2, 2, 2, 3, 3, 11, 6733, 302433367] 984 ms BUILD SUCCESSFUL (total time: 1 second)

La primera funció, **factoriza(n)**, s'ha basat en buscar només divisors primers fins a l'arrel quadrada de **n**, sabent-hi que, per a cada divisor trobat abans de l'arrel quadrada de **n**, hi haurà un altre per sobre d'aquesta. Caldrà abans d'afegir-lo a la llista que comprovar que aquest altre divisor també és primer, i després haurem d'ordenar el vector resultant, ja que els primers poden aparèixer desordenats donat el mètode d'obtenció.

La segona funció, **factoritza(n)**, ha buscat divisors fins a **n/2**, els primers apareixen ordenats, però triga **9** vegades més en la seva execució per al número de l'exemple.

4°) – Es diu que un número **M** és de **Marsenne** si és una unitat menor que una potència de **2**. Es a dir, **M** = **2**ⁿ – **1** per a **n Natural** (enter positiu i **n>=1**) **n** = **1**, **2**, **3**, **4**, **5**, **6**, **7** ... Es demana fer dues funcions, una que anomenarem **marsenne**(**n**) que retornarà un vector amb els **n** primers números de **Marsenne**, i una altra funció **primosMarsenne**(**m**) que retorni un vector amb tots els primers de **Marsenne** anteriors o iguals a **m**. Actualment, només és coneixen **51** números primers de **Marsenne**. Per exemple, si executem:

Si cridem en aquestes funcions com s'indica a continuació:

```
public static void main(String[] args) {
    System.out.println(java.util.Arrays.toString(marsenne(10)));
    System.out.println(java.util.Arrays.toString(primosMarsenne(1000000000L)));
}

El resultat podria ser:
```

```
C:\WINDOWS\SYSTEM32\cmd.exe
[1, 3, 7, 15, 31, 63, 127, 255, 511, 1023]
[1, 3, 7, 31, 127, 8191, 131071, 524287, 2147483647]
```

Més informació sobre els primers de **Marsenne** a la pàgina: https://ca.wikipedia.org/wiki/Nombre primer de Mersenne

5°) – **Fermat** va conjecturar que tots els números naturals (enters majors que zero) de la forma **2**° + **1** serien primers, però **Leonhard Euler** va demostrar que no era així al **1732**. A dia d'avui, només es coneixen **cinc** números de **Fermat** que no siguin primers. Consultar:

https://es.wikipedia.org/wiki/N%C3%BAmero de Fermat

Es demana fer una funció **fermat(n)** que retorni un vector amb els primers **n** números de **Fermat**. Es demana també fer una funció **primerFermatNoPrimo()** que retorni el primer número de **Fermat** que no és primer (el segon és **18446744044073709551617**, i excedeix la capacitat de representació dels valors de tipus **long** en java). Per saber si un número **n** es primer, podem utilitzar la funció **esPrimer(n)** del primer exercici. A continuació s'ha indicat un programa d'exemple per provar la primera funció demanada.

```
public static void main(String[] args) {
    System.out.println(java.util.Arrays.toString(fermat(5)));

    }

    ida ×
ebugger Console × Primers(run) ×

run:
[5, 17, 257, 65537, 4294967297]
BUILD SUCCESSFUL (total time: 0 seconds)
```

6º) – Un número és **Smith** si la suma dels seus dígits és igual a la suma dels dígits de cadascú dels nombres de la seva descomposició en factors primers. Per exemple tenim:

```
Exemple 1: 22 D.F.: 2 \times 11 2 + 2 = 2 + 1 + 1 = 4 Exemple 2: 378 D.F.: 2 \times 3 \times 3 \times 3 \times 7 3 + 7 + 8 = 2 + 3 + 3 + 3 + 7 = 18
```

Adonar-se que tots els primers no son **Smith** per definició, ja que no complirien amb el supòsit indicat (només son divisibles per si mateixos i la unitat). Més informació a:

https://es.wikipedia.org/wiki/N%C3%BAmero de Smith

Es demana fer una funció **isSmith(n)** que **retorni True** si **n** es **Smith**, i una altra **smiths(n)** que retorni un vector amb els primers **n** números **Smith**. Aquestes funcions farien us de **esPrimer(n)** i **factoriza(n)**. S'indiquen a continuació els primers números **Smith** retornats per la segona funció proposada.

```
☐ C:\WINDOWS\SYSTEM32\cmd.exe — □ ×

[4, 22, 27, 58, 85, 94, 121, 166, 202, 265, 274, 319, 346, 355, 378, 382, 391, 438, 454, ^
483, 517, 526, 535, 562, 576, 588, 627, 634, 636, 645, 648, 654, 663, 666, 690, 706, 72

8, 729, 762, 778, 825, 852, 861, 895, 913, 915, 922, 958, 985, 1086, 1111, 1165, 1219, 1
255, 1282, 1284, 1376, 1449, 1507, 1581, 1626, 1633, 1642, 1678, 1736, 1755, 1776, 1795, ✓
```

MÒDUL M03 Realitzat per : José Javier Faro Rodríguez FULL 3 - 3