Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 4. Un algoritme general de factorització

4.0. Introducció

Continuem amb un altre fragment de l'article 329 de les Disquisicions Aritmètiques, de C.F. Gauss, en la versió de la Dra. Griselda Pascual Xufré, editada per l'IEC, el 1996 (cf. [Ga-DA]).

"[...] es presenta no rarament l'ocasió al calculador expert de treure grans profits de la descomposició en factors de nombres grans que compensen llargament una despesa de temps regular; i, a part d'això, la dignitat de la ciència sembla requerir que es perfeccionin curosament tots els recursos per a la resolució d'un problema tan elegant i tan cèlebre. [...]"

(continuarà)

4.0.0. Funcions que aprofitarem de capítols anteriors

4.0.0.0. Tests de primeritat i certificats de composició

In [1]:
```
def SolovayStrassenTest(nn,ff):
    if nn==1:
        return false
    if nn in [2,3,5,7]:
        return true
    if is_even(nn):
        return false
    if ff<1:
        return 'Cal fer alguna prova.'
    f=0
    n2=(nn-1)//2
    while f<ff:
        g=ZZ.random_element(2,nn-1)
        x=Mod(g,nn)^n2
        if x==1 or x==nn-1:
            y=Mod(kronecker(g,nn),nn)
            if y!=x:
                return false
        else:
            return false
        f=f+1
    return 'Indeterminat'

```

4.1. Els passos inicials

Anomenarem Factoritza(nn) la funció general de factorització que volem programar. L'anirem programant pod a poc, de manera que la funció canviarà sovint; i no només en aquest capítol, sinó en els posteriors, quan hi anirem afegint mètodes específics de factorització.

4.1.0. El paràmetre ha de ser un nombre enter

Escriurem, momentàniament, nn, per al nombre que volem factoritzar, i que, en primer lloc, hem de comrpovar que és enter (en cas contrari, el programa retornarà error i acabarà).

In [2]:
```
def Factoritza(nn):
    if not nn in ZZ:
        return 'El paràmetre ha de ser un nombre enter.'
    return [[nn,1,'**']]
```

In [3]:
```
Factoritza('-33')
```

Out[3]:
```
'El paràmetre ha de ser un nombre enter.'
```

In [4]:
```
Factoritza(-33)
```

Out[4]:
```
[[-33, 1, '**']]
```

4.1.1. El format general de la factorització

Comencem per definir el format de presentació de les factoritzacions a fi que els diferents mètodes els puguin donar unificadament i que el programa general de factorització pugui cridar els diferents mètodes segons els necessiti.

Presentarem el resultat de la factorització final, i sovint els resultats de les factoritzacions parcials a mida que les anem obtenint, en una llista de llistes; per exemple, la factorització del nombre $40=2^3\cdot 5$ serà [[2,3],[5,1]]. Cadascuna de les llistes serà de la forma [n1,e1] o bé [n1,e1,'obs'], on n1 és un factor de nn, amb multiplicitat e1, i 'obs' és un paràmetre o una observació sobre el factor corresponent. Per exemple, per a la factorització $-120=-2^3\cdot 15$, podríem presentar [[-1,1],[2,3],[15,1,' ** ']], on ' ** ' voldria dir que el factor 15 apareix amb exponent 1 però encara no s'ha intentat factoritzar, o bé [[-1,1],[2,3],[15,1,' * ']], per a dir que el factor 15 apareix amb exponent 1, és compost, però no s'ha aconseguit factoritzar.

Així, per exemple, un resultat final com [[-1,1],[2,3],[7,2,?],[15,1,' * ']] voldria dir que el nombre és -5880 ($= -2^3 \cdot 7^2 \cdot 15$), però que no s'ha provat que 7 és primer ni s'ha aconseguit factoritzar 15, tot i que se sap que és compost.

Observació: notem que el paràmetre **' ** '** ja s'ha utilitzat en la sortida de les proves de més amunt, perquè no hem intentat factoritzar el factor corresponent.

D'altra banda, cal presentar les llistes ordenades de la manera següent. En primer lloc, si n'hi ha, el signe. Després, les llistes que corresponen a factors primers, ordenades per ordre creixent dels factors; si el nombre primer s'ha pogut certificar, apareix sense tercer paràmetre; si no, apareix amb el tercer paràmetre '?'. A continuació, les llistes que corresponen a factors compostos de nn, ordenades per ordre creixent del factor, amb un asterisc, ' * ', com a tercer element de la llista; això vol dir que aquest nombre és compost, però no s'ha pogut factoritzar. I, finalment, si n'hi ha, els altres amb explicació del seu paràmetre corresponent (per exemple, ' ** ' si encara no s'ha intentat factoritzar).

4.1.2. Els casos trivials i el signe

El primer pas de l'algoritme serà estudiar els casos trivials i el signe del nombre que es vol factoritzar.

Com a factorització dels nombres 0, 1, i -1, escriurem exactament [0], [1], i [[-1,1]], i acabarem. En cas contrari, com que els nombres enters són productes d'un signe i un nombre natural, si el nombre nn és negatiu, començarem la factorització amb la parella [-1,1] i canviarem nn per -nn, de manera que, en tots els casos, continuarem amb la factorització d'un nombre natural no nul n=nn o n=-nn, que és un nombre natural més gran que 1.

Notem que, com que no hem intentar factoritzar nn, el nombre apareix amb el paràmetre ' ** '.

In [5]:
```
def Factoritza(nn):
    if not nn in ZZ:
        return 'El paràmetre ha de ser un nombre enter.'
    if nn==0:
        return [0]
    if nn==1:
        return [1]
    if nn==-1:
        return [[-1,1]]
    if nn<0:
        n=-nn
        fact=[[-1,1],[n,1,'**']]
        return fact
    n=nn
    fact=[[nn,1,'**']]
    return fact
```

In [6]:
```
Factoritza(0)
```

Out[6]:
```
[0]
```

In [7]:
```
Factoritza(-1)
```

Out[7]:
```
[[-1, 1]]
```

In [8]:
```
Factoritza(1)
```

Out[8]:
```
[1]
```

In [9]:
```
Factoritza(-33)
```

Out[9]:
```
[[-1, 1], [33, 1, '**']]
```

In [10]:
```
Factoritza(33)
```

Out[10]:
```
[[33, 1, '**']]
```

4.2. L'algoritme bàsic

4.2.0. Control dels diferents factors

Es tracta que l'algoritme general apliqui successivament alguns mètodes de factorització als nombres que encara no s'ha provat de factoritzar.

A fi de poder portar un control dels diferents factors (primers, compostos, o encara per factoritzar) de nn, crearem tres llistes on posarem els factors corresponents: primers, compostos, o pendents.

Inicialment, a la llista pendents hi posatem els factors que encara resta per intentar fatoritzar, i, de fet, treballarem sobre els elements d'aquesta llista.

A la llista primers hi afegirem els factors primers que anem descobrint, amb la seva multiplicitat.

A la llista compostos hi anirem afegint els nombres que ho siguin però que el mètode que fem servir en aquell moment no aconsegueixi factoritzar, també amb la seva multiplicitat.

Al final d'aquest mètode, la llista dels pendents serà buida, perquè haurem repartit els factors que en resulten en les altres dues llistes, i intercanviarem les dues llistes pendents i compostos a fi d'aplicar el mètode següent als nombres (compostos) que ara són a la llista dels pendents.

I així fins que acabem tots els mètodes, o fins que la llista de pendents (i també la de compostos) sigui buida.

Inicialment, doncs, la llista compostos serà buida; la llista primers sera buida si nn és positiu, i contindrà el signe, [-1,1], si nn és negatiu; i la llista pendents contindrà la parella [n,1], on n=nn o bé n=-nn de manera que n és un nombre natural no nul.

Refem la funció amb la creació i la inicialització d'aquestes llistes.

In [11]:
```
def Factoritza(nn):
    if not nn in ZZ:
        return 'El paràmetre ha de ser un nombre enter.'
    if nn==0:
        return [0]
    if nn==1:
        return [1]
    if nn==-1:
        return [[-1,1]]
    if nn<0:
        primers=[[-1,1]]
        pendents=[[-nn,1,'**']]
    else:
        primers=[]
        pendents=[[nn,1,'**']]
    compostos=[]
    fact=primers+pendents
    return fact
```

In [12]:
```
Factoritza(-1)
```

Out[12]:
```
[[-1, 1]]
```

In [13]:
```
Factoritza(0)
```

Out[13]:
```
[0]
```

In [14]:
```
Factoritza(1)
```

Out[14]:
```
[1]
```

In [15]:
```
Factoritza(33)
```

Out[15]:
```
[[33, 1, '**']]
```

In [16]:
```
Factoritza(-33)
```

Out[16]:
```
[[-1, 1], [33, 1, '**']]
```

4.2.1. Distribució dels factors en les llistes

4.2.1.0. En començar un mètode de factorització

Notem que, en començar, la llista primers no conté cap nombre primer (potser només conté la parella [-1,1]), que la llista compostos és buida, i que la llista pendents només conté una parella [n,1,' ** '], amb dos asteriscs perquè encara no hem intentat factoritzar n (i, de fet, tampoc no sabem si n és primer o compost). En particular, els elements d'aquesta llista són coprimers dos a dos, coprimers dos a dos amb els elements de la llista primers i coprimers dos a dos amb els elements de la llista compostos. Aquesta propietat s'ha de mantenir durant tot l'algoritme: *tots els elements de les tres llistes han de ser primers entre si dos a dos, independentment de la llista en què estiguin*.

En un determinat estadi de la factorització, tindrem que la llista primers contindrà els primers que haurem trobat que divideixen nn, amb la seva multiplicitat respectiva; que la llista compostos conté tots els altres factors compostos que coneixem, amb la seva multiplicitat, i que la llista pendents és buida, perquè hem acabat l'aplicació d'un mètode de factorització i ja tenim els factors ben repartits.

Si hem acabat tots els mètodes de factorització implementats, ja podrem retornar la factorització: fem la reunió de les dues llistes primers i compostos (amb els tercers paràmetres que corresponguin, si cal) i acabem. En cas contrari, canviem els elements de la llista compostos a la llista pendents i continuem amb el mètode següent de factorització.

4.2.1.1. Mentre apliquem un nou mètode

Fixem un nou mètode de factorització, Metode(x).

No hauríem d'intentar factoritzar un nombre que no és compost (si més no, perquè és impossible "factoritzar-lo"). Per tant, convé assegurar-se d'aquest fet abans d'intentar factoritzar els nombres de la llista pendents; és a dir, hauríem de comprovar que tots els elements d'aquesta llista són compostos.

Això podria venir donat pel fet que el mètode anterior ja ho hagi deixat així, o no. En aquest segon cas, intentarem certificar que els nombres són compostos (sense escriure cap certificat) a partir de tests de primeritat. Els nombres que no certifiquem com a compostos, els afegirem a la llista de primers, amb la multiplicitat corresponent i que depèn de la multiplicictat de x abans d'aquesta factorització, i amb el tercer paràmetre '?' (o sense aquest paràmetre si podem certificar-lo com a primer). Els altres, compostos, els afegirem a la llista compostos, amb la mateixa multiplicitat que acompanya x en la llista pendents abans d'aquesta factorització.

Cal, doncs, programar una funció Reparteix(lta)) que, aplicada a una llista lta (de llistes [x,e] o [x,e,'obs']) reparteixi els seus elements en dues llistes prm i altres. La llista prm hauria de contenir les llistes [x,e] o bé [x,e,'?'], si x és primer, amb el tercer paràmetre '?' si no és certificat. L'altra llista, altres, hauria de contenir només les parelles [x,e] tals que x és compost. Com que aquesta repartició pot ser al final o bé entre diferents mètodes de factorització, el seu contingut s'afegirà després, si cal amb tercers paràmetres, a la llista pendents o a la llista compostos.

4.3. La funció Reparteix(lta)

La funció següent, Reparteix(lta) reparteix els nombres de la llista lta en dues llistes, que proporciona com a sortida, prm (amb el tercer paràmetre, si cal) i altres (sense tercer paràmetre), que haurem d'incorporar a la llista que correspongui (i amb el paràmetre que correspongui, si cal).

Per a repartir els elements de la llista, farem servir el test de primeritat de Solovay-Strassen. Per a això, caclularem un valor de la fita en funció del nombre de xifres binàries del nombre n que volem repartir. De fet, a partir del valor FitaSolovayStrassen=1024 (incorporat a la funció, però que es pot canviar fàciment), calcularem el valor real de la fita que utilitzarem com el mínim entre aquest valor 1024 i max(20,1+log(n,2))). Això assegura una fita com a mínim 20 i com a màxim 1024 per al nombre de proves a l'atzar del test aplicat a n.

In [17]:
```
def Reparteix(lta):
    aux=lta
    prm=[]
    altres=[]
    FitaSolovayStrassen=1024
    while len(aux)>0:
        n=aux[0][0]
        e=aux[0][1]
        aux=aux[1:len(aux)]
        fSS=min(FitaSolovayStrassen,max(20,1+log(n,2)))
        res=SolovayStrassenTest(n,fSS)
        if res=='Indeterminat':
            prm=prm+[[n,e,'?']]
        if res==true:
            prm=prm+[[n,e]]
        if res==false:
            altres=altres+[[n,e]]
    return prm,altres

```

Fem alguna prova per a veure'n el funcionament.

Notem que podem afegir el resultat de les llistes de sortida a la llista que més ens convingui.

In [18]:
```
primers=[[-1,1]]
compostos=[[143,2,'*']]
pendents=[[77,2,'**']]
llista=[[6,3],[13,4],[5,8]]
```

In [19]:
```
[prm,alt]=Reparteix(llista)
```

In [20]:
```
[prm,alt]
```

Out[20]:
```
[[[13, 4, '?'], [5, 8]], [[6, 3]]]
```

In [21]:
```
1 [sorted(primers+prm),sorted(pendents+alt),sorted(compostos+alt)]
```

Out[21]:
```
[[[-1, 1], [5, 8], [13, 4, '?']],
 [[6, 3], [77, 2, '**']],
 [[6, 3], [143, 2, '*']]]
```

I que el resultat de la funció és una parella de llistes, de manera que podem cridar-la directament, tot i que no és recomanable, perquè faríem dues vegades la feina.

In [22]:
```
1 Reparteix(llista)[0]
```

Out[22]:
```
[[13, 4, '?'], [5, 8]]
```

In [23]:
```
1 Reparteix(llista)[1]
```

Out[23]:
```
[[6, 3]]
```

4.4 Refinament de factoritzacions

Recordem que hem imposat que s'ha de satisfer la propietat de coprimeritat dos a dos dels elements de les tres llistes primers, pendents i compostos. I, a més a més, només aplicarem mètodes de factorització a nombres compostos. I en iniciar l'aplicació d'un nou Metode(x), tenim que tots els elements x de la llista pendents són compostos.

El procés serà el següent. Aplicarem el Metode(x) successivament a tots els nombres de la llista pendents, un a un, mentre la llista sigui no buida. Per a això, fixem-nos en el primer element (el 0-èsim), [x,e], de la llista pendents, que traiem de la llista. Poden succeir dues coses.

1. El Metode(x) no aconsegueix factoritzar x; és a dir, retorna [[x,1,'*']]. En aquest cas, afegirem la parella [x,e] a la llista compostos, de manera que podem continuar amb el següent element de la llista pendents (que torna a ser el primer, però la llista és més curta). I els nombres de les tres llistes encara són primers entre si dos a dos.

2. El Metode(x) aconsegueix factoritzar x i retorna una llista, lta, de longitud estrictament més gran que 1, formada per parelles de factors propis (i més grans que 1) de x, amb multiplicitats. Aquest és el cas que ens interessa més, perquè obtenim factors propis no trivials del nombre que volíem factoritzar, però podria ser que per a aquesta llista no se satisfés la propietat de coprimeritat dos a dos dels seus elements. Per exemple, en una factorització de 45, podríem obtenir els factors 3 i 15. Caldria, doncs, refer aquesta factorització en una altra, "més fina" per a la qual se satisfaci la propietat de coprimeritat dos a dos dels factors.

Observació. Que sigui "més fina" vol dir que tots els seus factors siguin divisors dels factors inicials, fet que comportarà que la suma de les multiplicitats sigui més gran que la suma de les multiplicitats dels factors inicials.

Anomenarem *refinament* de la llista a aquest procés, i caldrà fer una funció Refina(lta) per a obtenir-la. Un cop tinguem la llista aux=Refina(lta), caldrà repartir els seus elements en les llistes primers o compostos o pendents, segons els correspongui, amb els exponents multiplicats per e, i amb el paràmetre '?', si és el cas. Aquí és on es podrà aplicar la funció Reparteix(lta) per a repartir els factors en les llistes. I notem que si Metode(x) aconsegueix

trencar x i d és un factor propi i no trivial de x, encara podria ser que el mateix Metode(d) pugui trencar d, de manera que convé posar-lo a la llista pendents; o que segur que el Metode(d) no trencaria d, i convé posar-lo a la llista compostos. (És per això que la funció Reparteix(lta) no assigna els factors no primers a cap de les llistes pendents o compostos.)

I a continuació ja podrem aplicar el mètode Metode(x) a l'element següent x de la llista pendents, si és que encara en queden, llista per a la qual encara se satisfà la propietat de coprimeritat dos a dos dels seus elements entre si i amb els elements de les altres llistes.

4.4.0. La funció Refina(lta)

Cal, doncs, implementar una funció Refina(lta) que s'apliqui a una llista lta_1 de parelles $[x_1, e_{x_1}]$ amb $x_1 > 1$, $e_{x_1} > 0$, i en proporcioni una altra, lta_2, de parelles $[x_2, e_{x_2}]$ amb $x_2 > 1$, $e_{x_2} > 0$, però tal que $\prod_{x_1 \in lta_1} x_1^{e_{x_1}} = \prod_{x_2 \in lta_2} x_2^{e_{x_2}}$, que cadascun dels divisors x_2 sigui divisor d'algun dels x_1, i que els x_2 siguin primers entre si dos a dos. L'algoritme és es següent.

Si la llista lta és buida o conté un sol element, ja és refinada i la retornem. En cas contrari, tenim que la longitud és, com a mínim, 2.

Considerem una llista buida, ref=[ ], on anirem afegint els factors que siguin coprimers amb tots els posteriors, i una llista aux que, inicialment, és la llista lta.

Considerem el primer element, test=aux[0], i el traiem de la llista aux, de manera que fem la llista aux=aux[1:len(aux)] més curta, i considerem una segona llista buida aux2=[ ]. Volem comparar el nombre test[0] amb tots els de la llista aux.

Per a això, repetim el procés anterior: considerem el (nou) primer element, test2=aux[0], i el traiem de la llista aux, de manera que fem la llista aux=aux[1:len(aux)] encara més curta. Ara, per a l'element test2[0], calculem el màxim comú divisor d:=gcd(test[0],aux[0][0]). Poden succeir dues coses.

1. Si d>1, llavors calculem els nombres a=test[0]//d, v=test[1], b=test2[0]//d, w=test2[1]. Notem que a i b són primers entre si, i que el producte $test[0]^{test[1]} \cdot test2[0]^{test2[1]}$ coincideix amb el producte $d^{v+w} \cdot a^v \cdot b^w$. Per tant, si canviem el test per test=[d,v+w], i afegim a la llista aux les parelles [a,v], [b,w] per a les quals el primer element sigui diferent de 1, hem refinat la llista i podem continuar amb el test test i la llista aux, mentre aquesta sigui no buida.

2. Si d=1, el primer factor de test2 és coprimer amb el primer factor de test, de manera que podem passar test2 a la llista aux2 i continuar amb el test dels següents de la llista aux, si en queden, perquè no ha canviat cap parella (ni el test ni la parella test2, que ha passat a aux2).

Un cop haurem acabat la llista aux, tindrem que el primer element de test és coprimer amb tots els de la llista aux, de manera que el podem passar a la llista ref, posar aux=aux2, i continuar mentre la llista aux tingui més d'un element.

Al final, la llista aux contindrà un sol element (o potser, cap), de manera que la llista ref+aux serà la llista refinada que cerquem.

Notem que si els factors de la llista pendents són coprimers dos a dos, els possibles factors que apareguin en una factorització d'un dels seus elements també seran coprimers amb els

altres factors; per tant, la funció Refina(lta) només s'haurà d'aplicar a la sortida de cadascuna de les aplicacions del mètode a un factor de la llista pendents i no a tota aquesta llista ni a l'altra, de compostos.

Implementem aquesta funció.

In [24]:
```
def Refina(lta):
    if len(lta)<2:
        return lta
    aux=lta
    ref=[]
    while len(aux)>1:
        test=aux[0]
        aux=aux[1:len(aux)]
        aux2=[]
        while len(aux)>0:
            test2=aux[0]
            aux=aux[1:len(aux)]
            d=gcd(test[0],test2[0])
            if d>1:
                a=test[0]//d
                v=test[1]
                b=test2[0]//d
                w=test2[1]
                if b>1:
                    aux=[[b,w]]+aux
                if a>1:
                    aux=[[a,v]]+aux
                test=[d,v+w]
            else:
                aux2=aux2+[test2]
        ref=ref+[test]
        aux=aux2
    ref=ref+aux
    return sorted(ref)
```

Fem algunes proves de la funció. En particular, notem que pot ajudar en el procés de factorització, perquè pot detectar alguns factors primers!

In [25]:
```
Refina([[6,4],[2, 2],[6,4],[77,2],[154,2],[3, 2],[10,3]])
```

Out[25]:
```
[[2, 15], [3, 10], [5, 3], [77, 4]]
```

In [26]:
```
Refina([[6,1],[10,1],[15,1]])
```

Out[26]:
```
[[2, 2], [3, 2], [5, 2]]
```

4.5. Primera versió de la funció Factoritza(nn)

Refem la funció Factoritza(nn) amb la incorporació de la funció Reparteix(lta) a fi de deixar només nombres compostos per a factoritzar.

In [27]:
```
def Factoritza(nn):
    if not nn in ZZ:
        return 'El paràmetre ha de ser un nombre enter.'
    if nn==0:
        return [0]
    if nn==1:
        return [1]
    if nn==-1:
        return [[-1,1]]
    if nn<0:
        primers=[[-1,1]]
        pendents=[[-nn,1]]
    else:
        primers=[]
        pendents=[[nn,1]]
    compostos=[]
    [pr,cp]=Reparteix(pendents)
    cp=[cp[i]+[' ** '] for i in range(len(cp))]
    primers=primers+pr
    pr=[]
    pendents=cp
    cp=[]
    fact=primers+pendents
    return fact
```

In [28]:
```
Factoritza(0)
```

Out[28]: [0]

In [29]:
```
Factoritza(1)
```

Out[29]: [1]

In [30]:
```
Factoritza(-1)
```

Out[30]: [[-1, 1]]

Comprovem que la funció Factoritza(nn) fa allò que ha de fer.

In [31]:
```
Factoritza(33)
```

Out[31]: [[33, 1, ' ** ']]

In [32]:
```
Factoritza(-33)
```

Out[32]: [[-1, 1], [33, 1, ' ** ']]

In [33]:
```
Factoritza(-330)
```

Out[33]: [[-1, 1], [330, 1, ' ** ']]

In [34]:
```
Factoritza(-331)
```

Out[34]: [[-1, 1], [331, 1, '?']]

Fi del capítol 4