Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 5. Algoritme de divisió

5.0. 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]).

"[...] Abans que els mètodes següents siguin invocats, sempre és utilíssim intentar la divisió de qualsevol nombre proposat per alguns nombres primers molt petits, posem per 2, 3, 5, 7, etc., fins a 19 o encara més enllà, no només perquè no disgusti haver descobert, per mètodes subtils i artificiosos, un nombre, quan és divisor, tal que s'hauria pogut trobar molt més fàcilment per simple divisió, [...]"

5.0.0. Funcions que aprofitarem de capítols anteriors

5.0.0.0. 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'

```

In [2]:

```
def MillerRabinTest(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.'
    v=0
    m=nn-1
    while is_even(m):
        v=v+1
        m=m//2
    f=0
    while f<ff:
        g=ZZ.random_element(2,nn-1)
        x=Mod(g,nn)^m
        if x!=1 and x!=nn-1:
            k=1
            x=x^2
            while (x!=nn-1 and k<v-1):
                x=x^2
                k=k+1
            if k>=v-1 and x!=nn-1:
                return false
        f=f+1
    return 'Indeterminat'

```

In [3]:
```
def SolovayStrassenCert(nn,ff):
    if nn==1:
        return false, "n=1"
    if nn in [2,3,5,7]:
        return true
    if is_even(nn):
        return false, "g=",2
    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,"g=",g
        else:
            return false,"g=",g
        f=f+1
    return 'Indeterminat'

```

In [4]:
```
def MillerRabinCert(nn,ff):
    if nn==1:
        return false,'n=1'
    if nn in [2,3,5,7]:
        return true
    if is_even(nn):
        return false, "g=",2
    if ff<1:
        return 'Cal fer alguna prova.'
    v=0
    m=nn-1
    while is_even(m):
        v=v+1
        m=m//2
    f=0
    while f<ff:
        g=ZZ.random_element(2,nn-1)
        x=Mod(g,nn)^m
        if x!=1 and x!=nn-1:
            k=1
            x=x^2
            while (x!=nn-1 and k<v-1):
                x=x^2
                k=k+1
            if k>=v-1 and x!=nn-1:
                return false,"g=",g
        f=f+1
    return 'Indeterminat'

```

5.0.0.1. Certificats de primeritat

In [5]:

```
def Certifica(pp,lta,ff):
    if pp==1:
        return [pp,false,"p=1"]
    if pp==2 or pp==3:
        return [pp,true,pp-1,[pp-1]]
    if is_even(pp):
        return [pp,false,"g=2"]
    if ff<1:
        return ["Cal fer alguna prova."]
    if len(lta)==0:
        lta1=factor(pp-1)
        fppmu=[lta1[i][0] for i in range(len(lta1))]
    else:
        fppmu=sorted(lta)
    l=len(fppmu)
    f=0
    while f<ff:
        g=ZZ.random_element(2,pp-2)
        if (s:=Mod(g,pp)^((pp-1)//2))==pp-1:
            i=1
            while i<= l-1 and Mod(g,pp)^((pp-1)//fppmu[i])!=1:
                i=i+1
            if i==l:
                return [pp,true,g,fppmu]
        else:
            if s!=1:
                return [pp,false,g]
        f=f+1
    return [pp,'Indeterminat']
```

In [6]:

```
def Pocklington(pp,tt,ff):
    if not pp in ZZ or pp<1:
        return ['Cal que el nombre P sigui enter positiiu.']
    if pp==1:
        return [pp,false,1]
    if pp==2 or pp==3:
        return [pp,true,pp-1,[pp-1]]
    if is_even(pp):
        return [pp,false,2]
    if ff<1:
        return 'Cal fer alguna prova.'
# Comprovació que la llista tt és de divisors de pp-1, i càlcul d
# però no que són primers.
    if false in [(r in ZZ and r>1) for r in tt]:
        return 'La llista T no és de nombres enters >1.'
# Si 2 no pertany a la llista tt, li afegim (per a millora del ca
    t=tt
    if not (2 in t):
        t=[2]+t
    x=prod(t)
    q,r=divmod(pp-1,x)
    if r:
        return 'La llista T no és correcta.'
    d=gcd(q,x)
    while d>1:
        q=q//d
        d=gcd(q,x)
    uu=q
    q=uu^2
    if q==pp:
        return [pp,false,uu]
    if q>pp:
        return 'U és massa gran.'
    t=sorted(t)
# Si hem arribat aquí, és que P, T, F i U són correctes (excepte
# potser, que alguns elements de T no siguin primers).
    l=len(t)
    f=0
    while f<ff:
        g=ZZ.random_element(2,pp-2)
        if (s:=Mod(g,pp)^((pp-1)//2))==pp-1:
            i=1
            while i<= l-1 and gcd((s:=Mod(g,pp)^((pp-1)//t[i]))-1
                i=i+1
            if i==l:
                return [pp,true,g,t]
        else:
            if s!=1:
                return [pp,false,g]
        f=f+1
    return [pp,'Indeterminat']

```

5.0.0.2. Un garbell d'Eratòstenes

In [7]:
```
def Eratostenes(ff):
    f=floor((ff+1)/2)
    pr=[1 for i in range(f)]
    i=1
    k=floor((sqrt(ff)+1)/2)
    while i<k:
        if pr[i]==1:
            for j in range(2*i*(i+1),f,2*i+1):
                pr[j]=0
        i=i+1
    return pr

```

In [8]:
```
def LlistaDePrimers(ff):
    f=floor((ff+1)/2)
    pr=[1 for i in range(f)]
    i=1
    k=floor((sqrt(ff)+1)/2)
    while i<k:
        if pr[i]==1:
            for j in range(2*i*(i+1),f,2*i+1):
                pr[j]=0
        i=i+1
    lta=[pr[n]*(2*n+1) for n in range(f) if pr[n]>0]
    lta[0]=2
    return lta

```

5.0.0.3. Funcions per a factoritzar

In [9]:
```
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)
```

In [10]:
```
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

```

5.0.0.4. Primera versió de la funció Factoritza(nn)

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=[]
    [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
```

5.1. El primer pas, el garbell

Evidentment, de la definició de nombre primer es dedueix que si un nombre natural n>1 no és primer, llavors és divisible per algun nombre primer menor o igual que l'arrel quadrada de n.

Però en general aquest nombre és massa gran per a poder intentar fer aquestes divisions. Així, caldrà decidir fins a on estem disposats a dur a terme aquesta tasca. I un cop decidit això, caldrà calcular la llista dels nombres primers pels quals provarem de dividir.

Notem que la funció LlistaDePrimers(ff) proporciona la llista de tots els nombres primers menors que qualsevol fita ff>1. Però si la fita és molt gran, podríem trigar molt a calcular-la i hauríem de fer moltes divisions. Per tant, retallarem les possibilitats amb una fita màxima (que incorporarem al programa, però de manera que es pugui canviar fàcilment). Inicialment, posarem FitaEratostenes=10^5, i la fita ff per al garbell serà el mínim entre aquesta fita i el màxim entre 10 i l'arrel quadrada de n+1, on n>1 és el nombre (natural) que cal factoritzar. D'aquesta manera, ens assegurarem de dividir per 2 i per 3, fet que serà útil en alguns algoritmes (i en alguns casos, necessari), i també per 5 i per 7.

Incorporem aquesta tria i el càlcul de la llista de primers al nostre programa de factorització.

In [12]:
```
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=[]
    if len(pendents)==0:
        return primers
    FitaEratostenes=10^5
    n=pendents[0][0]
    e=pendents[0][1]
    ff=min(FitaEratostenes,max(10,ceil(sqrt(n))))
    pr=LlistaDePrimers(ff)
    fact=primers+pendents
    pendents=[]
    return fact, n, ff, len(pr)

```

In [13]:
```
Factoritza(450000)
```

Out[13]:
```
([[450000, 1, ' ** ']], 450000, 671, 121)
```

In [14]:
```
Factoritza(450000^2)
```

Out[14]:
```
([[202500000000, 1, ' ** ']], 202500000000, 100000, 9592)
```

5.2. Dividim

Notem que un cop trobem un divisor primer p de n, caldrà calcular-ne l'exponent, v, i el cofactor, s, no divisible per p, tals que $n=p^v \cdot s$. A continuació, ja només caldrà dividir fins a l'arrel quadrada de s. Per tant, només intentarem la divisió per p si $n<p^2$.

I així successivament; cada factor que trobem escurça (potser) la llista de factors per als quals hem de provar la divisió.

En particular, pot ser que el nombre n sigui molt gran, però els seus divisors primers siguin petits; això farà senzilla la factorització per divisió d'aquest nombre. Per exemple, el factorial d'un nombre m només és divisible per nombres primers menors que m, de manera que aquests nombres es factoritzen fàcilment per divisió.

D'altra banda, els nombres primers per als quals intentem la divisió, obtinguts a partir d'un garbell d'Eratòstenes, són primers amb tota seguretat, de manera que podem afegir aquests factors a la llista dels primers sense cap més indicació.

Finalment, si el cofactor, diem-ne n, que queda en fer les divisons és menor que el quadrat del darrer nombre primer per al qual hem intentat la divisió, segur que és primer, perquè no és divisible per cap primer menor que la seva arrel quadrada; per tant, també el podem afegir a la llista de primers, i hauríem acabat la factorització. En cas contrari, caldria mirar si és o no primer amb algun test de primeritat i actuar en conseqüència: si és primer, afegir-lo a a llista de primers amb un '?' i acabar la factorització, i si és compost, afegir-lo a la llista de pendents. Utilitzarem el test de Solovay-Strassen amb una fita min(1024,max(20,1+log(n,2))), que té en compte el nombre de xifres binàries del nombre n al qual s'aplica.

Observació. Notem que el fet de provar la divisió com a mínim fins a 7 obliga que el retorn del test de Solovay-Strassen només pugui ser false o 'Indeterminat', fet que farà més senzilla la implementació.

Implementem aquests fets a la funció Factoritza(nn). Els incorporarem com a primer pas (i obligatori) de la factorització, en el nucli del programa.

In [15]:

```
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=[]
    if len(pendents)==0:
        return primers
    FitaEratostenes=10^5
    n=pendents[0][0]
    e=pendents[0][1]
    pendents=[]
    ff=min(FitaEratostenes,max(10,ceil(sqrt(n))))
    pr=LlistaDePrimers(ff)
    l=len(pr)
    i=0
    p=pr[0]
    while n>=p^2 and i<l-1:
        a,b=divmod(n,p)
        if b==0:
            v=0
            while b==0:
                n=a
                v=v+1
                a,b=divmod(n,p)
            primers=primers+[[p,v]]
        i=i+1
        p=pr[i]
    if n>=p^2 and i==l-1:
        a,b=divmod(n,p)
        if b==0:
            v=0
            while b==0:
                n=a
                v=v+1
                a,b=divmod(n,p)
            primers=primers+[[p,v]]
    if n<p^2 and n>1:
        primers=primers+[[n,1]]
        n=1
        fact=primers
        return fact
    if n==1:
```

```
60          fact=primers
61          return fact
62      FitaSolovayStrassen=1024
63      fSS=min(FitaSolovayStrassen,max(20,1+log(n,2)))
64      if SolovayStrassenTest(n,fSS)=='Indeterminat':
65          primers=primers+[[n,1,'?']]
66      else:
67          pendents=pendents+[[n,1,'**']]
68      compostos=[]
69      fact=primers+pendents
70      return fact
71
```

5.3. Exemples

In [16]:
```
1 Factoritza(factorial(25)*631*next_prime(10^10))
```

Out[16]:
```
[[2, 22],
 [3, 10],
 [5, 6],
 [7, 3],
 [11, 2],
 [13, 1],
 [17, 1],
 [19, 1],
 [23, 1],
 [631, 1],
 [10000000019, 1, '?']]
```

In [17]:
```
1 p1=32875210195602465200111111089
2 p2=32875210195602465200111111207
```

In [18]:
```
1 Factoritza(factorial(25)*631*p1)
```

Out[18]:
```
[[2, 22],
 [3, 10],
 [5, 6],
 [7, 3],
 [11, 2],
 [13, 1],
 [17, 1],
 [19, 1],
 [23, 1],
 [631, 1],
 [32875210195602465200111111089, 1, '?']]
```

In [19]:
```
1 Factoritza(factorial(25)*631*p1*p2)
```

Out[19]:
```
[[2, 22],
 [3, 10],
 [5, 6],
 [7, 3],
 [11, 2],
 [13, 1],
 [17, 1],
 [19, 1],
 [23, 1],
 [631, 1],
 [10807794454050442782030139970083604289200164448872098744
23, 1, '**']]
```

In [20]:
```
1 Factoritza(factorial(25)*631*next_prime(10^10)*p1)
```

Out[20]:
```
[[2, 22],
 [3, 10],
 [5, 6],
 [7, 3],
 [11, 2],
 [13, 1],
 [17, 1],
 [19, 1],
 [23, 1],
 [631, 1],
 [32875210258065364571755794969211111069 1, '**']]
```

In [21]:
```
1 Factoritza(factorial(25)*631*next_prime(10^10)*p1*p2)
```

Out[21]:
```
[[2, 22],
 [3, 10],
 [5, 6],
 [7, 3],
 [11, 2],
 [13, 1],
 [17, 1],
 [19, 1],
 [23, 1],
 [631, 1],
 [10807794474585252244725981255940870232359012598352411197086987614
037,
  1,
  '**']]
```

Observem que *SageMath* és capaç de factoritzar prou ràpidament aquest nombre.

In [22]:
```
1 %time factor(factorial(25)*631*next_prime(10^10)*p1*p2)
```

```
CPU times: user 7.18 s, sys: 24.5 ms, total: 7.2 s
Wall time: 7.16 s
```

Out[22]:
```
2^22 * 3^10 * 5^6 * 7^3 * 11^2 * 13 * 17 * 19 * 23 * 631 * 10000000
019 * 32875210195602465200111111089 * 3287521019560246520011111120 7
```

5.4. Segona versió de la funció Factoritza(nn)

Notem que, en aplicar la funció com és ara, al final o bé hem obtingut tota la factorització, fet que es determina si el darrer factor no porta el paràmetre ' * ' ni el paràmetre ' ** ', o bé només resta un factor compost per a factoritzar, el darrer, que apareix amb el paràmetre ' ** ', perquè encara no s'ha intentat factoritzar.

Això es tradueix en què la llista pendents és buida, en el primer cas, i consisteix en [[n,1,' ** ']], en el segon; a més a més, la llista compostos és buida i la llista primers conté la factorització actual, excepte la part que correspon al factor (compost) n.

A partir d'aquí, doncs, hem d'utilitzar altres mètodes més sofisticats de factorització sobre la llista pendents.

Notem que ara no hem de refinar ni de repartir, perquè, com a màxim, resta un sol valor, i ja sabem que és compost.

A partir d'aquí, l'algoritme general aplicarà successivament mètodes de factorització de la manera següent.

Fixem un primer mètode de factorització, que passarem a tots els nombres de la llista pendents, un a un. Per a això, eliminarem el primer element de la llista pendents, diem-ne [x,e], o bé [x,e,' * '], i calcularem Metode(x). Si el mètode el trenca, i amb l'ús de la funció Refina( ), convertirem els factors de x en factors primers entre si dos a dos (encara que modifiquem la quantitat de factors), i amb l'ús de la funció Reparteix( ) afegirem aquests factors, amb els exponents corresponents, a la llista primers o a la llista pendents, segons el cas. Si el Metode(x) no trenca x, enviarem [x,e] a la llista compostos i passarem al proper element de la llista pendents.

In [23]:

```
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=[]
    if len(pendents)==0:
        return primers
    FitaEratostenes=10^5
    n=pendents[0][0]
    e=pendents[0][1]
    pendents=[]
    ff=min(FitaEratostenes,max(10,ceil(sqrt(n))))
    pr=LlistaDePrimers(ff)
    l=len(pr)
    i=0
    p=pr[0]
    while n>=p^2 and i<l-1:
        a,b=divmod(n,p)
        if b==0:
            v=0
            while b==0:
                n=a
                v=v+1
                a,b=divmod(n,p)
            primers=primers+[[p,v]]
        i=i+1
        p=pr[i]
    if n>=p^2 and i==l-1:
        a,b=divmod(n,p)
        if b==0:
            v=0
            while b==0:
                n=a
                v=v+1
                a,b=divmod(n,p)
            primers=primers+[[p,v]]
    if n<p^2 and n>1:
        primers=primers+[[n,1]]
        n=1
        fact=primers
        return fact
    if n==1:
```

```
60          fact=primers
61          return fact
62      FitaSolovayStrassen=1024
63      fSS=min(FitaSolovayStrassen,max(20,1+log(n,2)))
64      if SolovayStrassenTest(n,fSS)=='Indeterminat':
65          primers=primers+[[n,1,'?']]
66      else:
67          pendents=pendents+[[n,1,'**']]
68      compostos=[]
69      fact=primers+pendents
70      return fact
71
```

Fi del capítol 5