Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 8. Mètode p-1 de Pollard

8.0. Introducció

En aquest capítol es tracta d'estudiar una versió bàsica i simplificada del mètode p-1 de Pollard, de la qual en farem una implementació, que afegirem també a l'algoritme general de factorització.

El mètode es pot aplicar a un nombre natural compost, n, i permetrà (probablement) trobar-ne un factor (probablement primer) no trivial, p, si els divisors primers de p-1 són tots nombres prou "petits". Aquí, "petits" depèn de la capacitat de càlcul i del temps que estiguem disposats a invertir en el procés (aleatori) de prova. Cal també que per a algun nombre primer q que divideixi n no tots els divisors de q-1 siguin "petits".

8.0.0. Funcions que aprofitarem de capítols anteriors

8.0.0.0. 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'

```

8.0.0.1. 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
# 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 c
    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]))-
                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']

```

8.0.0.2. 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

```

8.0.0.3. Funcions per a factoritzar

In [9]:
```python
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]:
```python
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

```

8.0.0.4. La funció PollardRho(nn,tt,ff)

In [11]:

```
def PollardRho(nn,tt,ff):
    f=ff
    while f>0:
        a=ZZ.random_element(nn)
        x=ZZ.random_element(nn)
        y=x
        t=tt
        while t>0:
            x=(x^2+a)%nn
            y=((y^2+a)^2+a)%nn
            d=gcd(x-y,nn)
            if d>1 and d<nn:
                # Cal mantenir enters els paràmetres!!!
                return [d,nn//d]
            if d==1:
                t=t-1
            if d==nn:
                t=0
        f=f-1
    return nn

```

8.0.0.5. Tercera versió de la funció Factoritza(nn)

In [12]:

```
def Factoritza(nn):
# Control del paràmetre d'entrada i factoritzacions trivials.
    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]]
# Creació de les llistes pendents, primers i compostos.
    if nn<0:
        primers=[[-1,1]]
        pendents=[[-nn,1]]
    else:
        primers=[]
        pendents=[[nn,1]]
    compostos=[]
# Repartició dels pendents. Si no en queda cap, retorn.
    [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
# Calculem la llista de primers petits per a la divisió.
    FitaEratostenes=10^5
    n=pendents[0][0]
    e=pendents[0][1]  # Notem que aquí ha de ser e=1.
    pendents=[]
    ff=min(FitaEratostenes,max(10,ceil(sqrt(n))))
    pr=LlistaDePrimers(ff)
    l=len(pr)
 # Comencem la divisió.
    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:
        fact=primers
        return fact
# Si som aquí, és que queda un factor. Mirem si és primer.
    FitaSolovayStrassen=1024
    fSS=min(FitaSolovayStrassen,max(20,1+log(n,2)))
    if SolovayStrassenTest(n,fSS)=='Indeterminat':
        primers=primers+[[n,1,'?']]
    else:
        pendents=pendents+[[n,1,'**']]
    compostos=[]
# Si som aquí, queda un factor compost, en pendents.
# Comencem amb el mètode Rho de Pollard.
    FitaRho=10^6    # Es pot augmentar, probablement fins a 10^8
    IteracionsRho=8 # Es pot augmentar, però no millora gaire.
    while len(pendents)>0:
        n=pendents[0][0]
        e=pendents[0][1]
        pendents=pendents[1:len(pendents)]
        rho=PollardRho(n,min(FitaRho,floor(10*sqrt(n))),Iteracio
        if rho==n:
            compostos=compostos+[[n,e,'**']]
        else:
            [prm,cp]=Reparteix(Refina([[rho[0],e],[rho[1],e]]))
            primers=sorted(primers+prm)
            prm=[]
            cp=[cp[i]+[' ** '] for i in range(len(cp))]
            pendents=pendents+cp
            cp=[]
    pendents=compostos
    compostos=[]
# Hem acabat el mètode rho.
    fact=primers+pendents
    return fact

```

8.1. Descripció del mètode p-1

Com que per a qualssevol nombres naturals b, n, es té que $\gcd(b, n)$ és un divisor de n, podem intentar calcular-ne a fi d'obtenir un divisor no trivial d'un nombre natural compost $n > 1$. Cal veure, però, com es poden triar els nombres b a fi de poder tenir èxit.

8.1.0. Proposició

Siguin $n > 1$ un nombre natural compost, p un divisor primer de n, k un múltiple de $p - 1$, i a un nombre natural coprimer amb n. Llavors, $\gcd(a^k - 1, n) > 1$.

Per tant, potser podem provar de triar valors de a i de k adequats.

8.1.1. Tria d'un valor a

Siguin $n > 2$ un nombre natural senar i $k \geq 1$ un nombre natural qualsevol. Per a a tal que $a \equiv -1, 0, 1 \pmod{n}$,, se satisfà que $\gcd(a^k - 1, n) = 1$, o $\gcd(a^k - 1, n) = n$.

En particular, doncs, si volem factoritzar n via el càlcul d'aquest màxim comú divisor, no convé triar aquests valors de a. Però podrem triar aleatòriament a tal que $2 \leq a \leq n - 2$. O bé, per a assegurar-nos que a és coprimer amb n, i si suposem que n no és divisible per nombres primers "petits", podem prendre a un nombre primer "petit".

Notem que si hem provat la divisió fins a una fita fE, els primers menors que fE satisfan aquesta propietat. A la nostra implementació, triarem $a = 2$ (notem que suposem que n és senar), però podríem refer fàcilment el programa i prendre a a l'atzar. O, fins i tot, canviar unes quantes vegaces, si convé, de valor a, en cas que l'algoritme no aconsegueixi factoritzar n. A la pràctica, aquest canvi de valor no suposa una millora apreciable de l'algoritme, i la descartem.

8.1.2. Tria de l'exponent k

Suposem que per a uns certs valors a i k se satisfà que $\gcd(a^k - 1, n) = n$. Llavors, per a qualsevol múltiple de k es tindrà la mateixa propietat, $\gcd(a^{rk} - 1, n) = n$, perquè $a^k \equiv 1 \pmod{n}$ implica que $a^{rk} \equiv 1 \pmod{n}$.

Així, si tenim un valor de k per al qual el màxim comú divisor és n, no factoritzarem amb cap múltiple de k. En aquest cas, caldria provar divisors de k (sense cap seguretat que això funcioni), o bé canviar el valor de a (també sense cap seguretat que això funcioni), o bé aturar el procés, que no ha aconseguit la factorització de n.

Ara bé, suposem que existeix un divisor primer p de n tal que tots els divisors primers de $p - 1$ són menors que una fita "petita", f. Per exemple, que $p - 1$ sigui un divisor de $k = f!$, o un divisor (d'una potència "petita") de $k = lcm(2, 3, 4, \dots, f)$, o un divisor d'una potència "petita" del producte de tots els nombres primers menors que f, o..., i que això no succeeixi (amb la mateixa fita) per a algun altre divisor primer q de n.

Com que a és invertible mòdul n, també ho és mòdul p i mòdul q. Ara, l'ordre multiplicatiu de a mòdul p, que és un divisor de $p - 1$, és un divisor del valor triat de k i, per tant, p divideix $\gcd(a^k - 1, n)$. En canvi, és probable que això no succeeixi per a q, de manera que q no divideix $\gcd(a^k - 1, n)$ i, en conseqüència, $1 < \gcd(a^k - 1, n) < n$ i factoritzem n.

Per a la implementació, cal un mètode que faci senzill el càlcul dels elements $a^k - 1$. Notem que no cal calcular el valor exacte $a^k - 1$, sinó que és suficient calcular-lo mòdul n, en l'interval $0 \leq a^k - 1 \leq n - 1$; o sigui, calcular $a^k \pmod{n}$ en l'interval $1 \leq k \leq n$. En particular, si la tria que fem per a l'exponent és $f!$, a partir de $a^{r!} \pmod{n}$ és senzill calcular $a^{(r+1)!} = (a^{r!})^{r+1} \pmod{n}$, per a $1 \leq r \leq f$ *i mentre no haguem aconseguit factoritzar* n. És a dir, si per a un valor $r < f$ ja factoritzem n, no cal continuar i aturem els càlculs.

En resum, anem calculant $a^{r!} \pmod{n}$ incrementant el valor de r com a màxim fins a la fita, amb una exponenciació mòdul n per a cada nou valor de r.

8.2. Lafunció PollardPmU(nn,ff)

Anomenarem nn el nombre que volem factoritzar, i ff la fita màxima per a l'exponent $r!$, $1 \leq r \leq ff$, per al qual calcularem.

Observació. Com que la funció està pensada per a ser inclosa en l'algoritme general de factorització, on els paràmetres que es proporcionin ja seran els adequats, no farem cap control d'aquests paràmetres a l'entrada; per exemple, no comprovarem que nn sigui un nombre natural compost i no divisible per primers menors que ff, cosa que suposarem.

In [13]:
```
def PollardPmU(nn,ff):
    a=2
    x=Mod(a,nn)
    r=2
    while r<ff:
        x=x^r
        d=gcd(x-1,nn)
        if d==nn:
            return nn
        if d>1:
# Cal que d i nn//d siguin enters, però d no ho és.
            d=Integer(d)
            return [d,nn//d]
        r=r+1
    return nn

```

8.2.0. Exemples

In [14]:
```
PollardPmU((factorial(27)+1)*(factorial(37)+1),50)
```

Out[14]:
```
[10888869450418352160768000001, 1376375309122634504631597958158090240000000
400000001]
```

In [15]:
```
PollardPmU((factorial(27)+1)*(factorial(37)+1)*(factorial(80)+1)
```

Out[15]:
```
[10888869450418352160768000001,
 985064335657904530906571232094964195854994970463220635338590103922
6258945090868747762977271781876119903243190008334078951210026131970
0432713005958158090240000001]
```

In [16]:
```
PollardPmU((factorial(37)+1)*(factorial(80)+1),50)
```

Out[16]:
```
[13763753091226345046315979581580902400000001,
 715694570462638022948115337231865321655846573423657525771094450582
2703925548014884266894486728081408000000000000000001]
```

In [17]:
```
PollardPmU(factorial(80)+1,10000)
```

Out[17]:
```
[672937,
 106353874205555352573586433385571802658472720837709551677362732407
6801234818120401206486563634943747780252831988873]
```

In [18]:
```
PollardPmU((factorial(80)+1)//672937,1000000)
```

Out[18]:
```
1063538742055553525735864333855718026584727208377095516773627324076
80123481812040120648656363494374778025283198873
```

Notem que altres mètodes poden funcionar en aquest nombre, mentre que els altres no funcionen en anteriors exemples.

In [19]:
```
PollardRho((factorial(80)+1)//672937,1000000,15)
```

Out[19]:
```
[351900745811,
 3022269076482044119312468500438400813073539099709650534866303679749
89354040997196506077104204669539043]
```

8.3. Quarta versió de la funció Factoritza(nn)

Hem posat aquest mètode PollardPmU al final de l'algoritme general. Però potser seria igual de bo?, o més?, o menys?, posar-lo abans que el mètode PollardRho?

Probablement no hi hagi una resposta única a aquesta qüestió, i l'ordre dels diferents algoritmes es pugui modificar de manera general, o bé només es pot triar l'ordre de manera més eficient si coneixem algunes propietats del nombre que cal factoritzar.

In [20]:

```
def Factoritza(nn):
# Control del paràmetre d'entrada i factoritzacions trivials.
    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]]
# Creació de les llistes pendents, primers i compostos.
    if nn<0:
        primers=[[-1,1]]
        pendents=[[-nn,1]]
    else:
        primers=[]
        pendents=[[nn,1]]
    compostos=[]
# Repartició dels pendents. Si no en queda cap, retorn.
    [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
# Calculem la llista de primers petits per a la divisió.
    FitaEratostenes=10^5
    n=pendents[0][0]
    e=pendents[0][1]  # Notem que aquí ha de ser e=1.
    pendents=[]
    ff=min(FitaEratostenes,max(10,ceil(sqrt(n))))
    pr=LlistaDePrimers(ff)
    l=len(pr)
 # Comencem la divisió.
    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:
        fact=primers
        return fact
# Si som aquí, és que queda un factor. Mirem si és primer.
    FitaSolovayStrassen=1024
    fSS=min(FitaSolovayStrassen,max(20,1+log(n,2)))
    if SolovayStrassenTest(n,fSS)=='Indeterminat':
        primers=primers+[[n,1,'?']]
    else:
        pendents=pendents+[[n,1,'**']]
    compostos=[]
# Si som aquí, queda un factor compost, en pendents.
# Comencem amb el mètode Rho de Pollard.
    FitaRho=10^6    # Es pot augmentar, probablement fins a 10^8
    IteracionsRho=8 # Es pot augmentar, però no millora gaire.
    while len(pendents)>0:
        n=pendents[0][0]
        e=pendents[0][1]
        pendents=pendents[1:len(pendents)]
        rho=PollardRho(n,min(FitaRho,floor(10*sqrt(n))),Iteracio
        if rho==n:
            compostos=compostos+[[n,e,'**']]
        else:
            [prm,cp]=Reparteix(Refina([[rho[0],e],[rho[1],e]]))
            primers=sorted(primers+prm)
            prm=[]
            cp=[cp[i]+[' ** '] for i in range(len(cp))]
            pendents=pendents+cp
            cp=[]
    pendents=compostos
    compostos=[]
# Hem acabat el mètode rho.
# Comencem el mètode p-1.
    FitaPmU=FitaEratostenes  # Es pot augmentar, probablement fi
    while len(pendents)>0:
        n=pendents[0][0]
        e=pendents[0][1]
        pendents=pendents[1:len(pendents)]
        PmU=PollardPmU(n,FitaPmU)
        if PmU==n:
            compostos=compostos+[[n,e,'**']]
        else:
            [prm,cp]=Reparteix(Refina([[PmU[0],e],[PmU[1],e]]))
            primers=sorted(primers+prm)
            prm=[]
            cp=[cp[i]+[' ** '] for i in range(len(cp))]
            pendents=pendents+cp
            cp=[]
    pendents=compostos
    compostos=[]
# Hem acabat el mètode p-1.
    fact=primers+pendents
    return fact

```

8.4. Exemples

In [21]:
```
Factoritza(factorial(37)*538736922377^2*304821096639811)
```

Out[21]:
```
[[2, 34],
 [3, 17],
 [5, 8],
 [7, 5],
 [11, 3],
 [13, 2],
 [17, 2],
 [19, 1],
 [23, 1],
 [29, 1],
 [31, 1],
 [37, 1],
 [538736922377, 2, '?'],
 [304821096639811, 1, '?']]
```

In [22]:
```
Factoritza(factorial(37)*538736922377^2*337991527361*30482109663
```

Out[22]:
```
[[2, 34],
 [3, 17],
 [5, 8],
 [7, 5],
 [11, 3],
 [13, 2],
 [17, 2],
 [19, 1],
 [23, 1],
 [29, 1],
 [31, 1],
 [37, 1],
 [337991527361, 1, '?'],
 [538736922377, 2, '?'],
 [304821096639811, 1, '?']]
```

In [23]:
```
Factoritza(factorial(37)*538736922377^2*337991527361^2*30482109
```

Out[23]:
```
[[2, 34],
 [3, 17],
 [5, 8],
 [7, 5],
 [11, 3],
 [13, 2],
 [17, 2],
 [19, 1],
 [23, 1],
 [29, 1],
 [31, 1],
 [37, 1],
 [337991527361, 2, '?'],
 [538736922377, 2, '?'],
 [304821096639811, 1, '?']]
```

In [24]:
```
1 Factoritza(factorial(37)*538736922377^2*337991527361^2*304821096(
```
Out[24]:
```
[[2, 34],
 [3, 17],
 [5, 8],
 [7, 5],
 [11, 3],
 [13, 2],
 [17, 2],
 [19, 1],
 [23, 1],
 [29, 1],
 [31, 1],
 [37, 1],
 [337991527361, 2, '?'],
 [538736922377, 2, '?'],
 [929159009566969969 15070115721, 1, '**']]
```

In [25]:
```
1 Factoritza(factorial(37)*538736922377^2*304821096639811*(factori
```
Out[25]:
```
[[2, 34],
 [3, 17],
 [5, 8],
 [7, 5],
 [11, 3],
 [13, 2],
 [17, 2],
 [19, 1],
 [23, 1],
 [29, 1],
 [31, 1],
 [37, 1],
 [538736922377, 2, '?'],
 [304821096639811, 1, '?'],
 [10888869450418352160768000001, 2, '?']]
```

In [26]:
```
1 Factoritza((factorial(27)+1)*(factorial(37)+1)*(factorial(80)+1)
```
Out[26]:
```
[[672937, 1, '?'],
 [351900745811, 1, '?'],
 [10888869450418352160768000001, 1, '?'],
 [13763753091226345046315979581580902400000001, 1, '?'],
 [302226907648204411931246850043840081307353909970965053486630367974
98935404099719650607710420466953904 3,
  1,
  '**']]
```

In [27]:
```
1 Factoritza((factorial(27)+1)^2)
```
Out[27]:
```
[[10888869450418352160768000001, 2, '?']]
```

In [28]:
```
1 factor(302226907648204411931246850043840081307353909970965053486(
```
Out[28]:
```
1374851388985363 * 2198251462445313008276184348343804395996611503335
254613064671967626983387366042164213 61
```

```
In [29]:  1  factor(1374851388985363-1)
Out[29]: 2 * 3^2 * 17 * 179 * 25100437963

In [30]:  1  factor(21982514624453130082761843483438043959966115033352546130б
Out[30]: 2^4 * 5 * 311 * 313 * 1896019 * 2679283 * 26117587 * 21275919259916
         44286029498079727935474305201081989639160770 31

In [31]:  1  Factoritza(2198251462445313008276184348343804395996611503335254б
Out[31]: [[2, 4],
          [5, 1],
          [311, 1],
          [313, 1],
          [1896019, 1, '?'],
          [2679283, 1, '?'],
          [26117587, 1, '?'],
          [2127591925991644286029498079727935474305201081989639160770 31, 1,
         '?']]

In [32]:  1  factor(21982514624453130082761843483438043959966115033352546130б
Out[32]: 2 * 3 * 53408669 * 1600675343246101 * 4285594837523036162520641873 1
         679984926593244004751631200012 4083

In [33]:  1  factor(1374851388985363+1)
Out[33]: 2^2 * 401 * 1013 * 16987 * 49811

In [34]:  1  Factoritza(1374851388985363+1)
Out[34]: [[2, 2], [401, 1], [1013, 1], [16987, 1], [49811, 1]]

In [35]:  1  Factoritza(2198251462445313008276184348343804395996611503335254б
Out[35]: [[2, 1],
          [3, 1],
          [53408669, 1, '?'],
          [68598459875659043549533888975458409694894646639208034843702744449
         8128005950383,
           1,
           '**']]
```

Fi del capítol 8