Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 7. Mètode rho de Pollard

Artur Travesa

(versió 2024-04)

7.0. Introducció

En aquest capítol es tracta d'estudiar una versió bàsica i simplificada del mètode rho 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 no trivial si el divisor primer més petit de n és un nombre prou "petit". Aquí, "petit" depèn de la capacitat de càlcul i del temps que estiguem disposats a invertir en el procés (aleatori) de prova.

7.0.0. Funcions que aprofitarem de capítols anteriors

7.0.0.0. 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'

```

7.0.0.1. 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']

```

7.0.0.2. 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

```

7.0.0.3. 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

```

7.0.0.4. Segona 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=[]
    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:
```

```
        fact=primers
        return fact
    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=[]
    fact=primers+pendents
    return fact

```

7.1. El fonament teòric

7.1.0. Proposició

Siguin C un conjunt finit i no buit, $f : C \longrightarrow C$ una aplicació qualsevol, i $x_0 \in C$ un element qualsevol. Considerem la successió $\{x_k\}_{k\geq 0}$ d'elements $x_k \in C$ construïda recursivament a partir de x_0 per aplicació reiterada de f; és a dir, $x_{k+1} := f(x_k)$, per a $k \geq 0$; o sigui, $x_k = f^k(x_0)$. Llavors, la successió és periòdica d'un lloc endavant; és a dir, existeixen nombres naturals $K \geq 0, T \geq 1$, tals que per a tot $k \geq K$ és $x_{T+k} = x_k$.

7.1.1. Definició

Els menors nombres $K \geq 0, T \geq 1$ per als quals se satisfà la propietat s'anomenen, respectivament, el *preperíode* i el *període* de la successió.

És un exercici senzill provar que el preperíode i el període són nombres menors o iguals que $\#C$; i, encara més, que per a la seva suma és $1 \leq K + T \leq \#C$.

Notem que el preperíode i el període no només depenen de l'aplicaió f, sinó també de l'element x_0 que triem com a primer terme de la successió.

7.1.2. Observació.

Fixem un nombre natural n (que si es vol es pot suposar compost), un divisor d de n (que també es pot suposar propi i no trivial), i una aplicació polinòmica $f : \mathbb{Z}/n\mathbb{Z} \longrightarrow \mathbb{Z}/n\mathbb{Z}$; és a dir, tal que existeix un polinomi $a_0 + a_1 X + \cdots + a_k X^k \in (\mathbb{Z}/n\mathbb{Z})[X]$, $a_0, \ldots, a_k \in \mathbb{Z}/n\mathbb{Z}$, $a_k \neq 0$, $k \geq 1$, tal que per a tot $x \in \mathbb{Z}/n\mathbb{Z}$ és $f(x) = a_0 + a_1 x + \cdots + a_k x^k$.

Notem que, en particular, si $x \equiv y \pmod{d}$, llavors $f(x) \equiv f(y) \pmod{d}$.

Per a un element $x_0 \in \mathbb{Z}/n\mathbb{Z}$, considerem la successió $\{x_k\}_{k\geq 0}$ definida per $x_k := f^k(x_0)$, i els seus preperíode i període, K, T. Llavors, la successió $\{x_k \pmod{d}\}_{k\geq 0}$ és periòdica i per als seus preperíode i període, K_d, T_d, se satisfan les desigualtats $K_d \leq K$ i $T_d \leq T$; de fet, encara més, es té que T_d és un divisor de T.

7.2. L'algoritme

7.2.0. La base

Suposem, doncs, que n és un nombre que volem factoritzar i que d n'és un divisor propi i no trivial.

Considerem una aplicació polinòmica com abans, f, un element $x_0 \in \mathbb{Z}/n\mathbb{Z}$, i la successió $x_k := f^k(x_0)$, per a $k \geq 0$.

Si tenim que $x_u \equiv x_v \pmod{d}$, llavors resulta que d és un divisor de $\gcd(x_u - x_v, n)$, que és diferent de n si $x_u \not\equiv x_v \pmod{n}$.

Per tant, si el preperíode i el període de la successió mòdul d són prou petits, potser podrem trobar parelles (u, v), amb $v = u + r \cdot T_d$ per a algun r petit, de manera que aquest càlcul de màxims comuns divisors ens proporcioni un divisor no trivial de n.

7.2.1. Observació

Encara que no ho demostrarem, un estudi acurat de distribucions de probabilitat permet provar que, si p és un nombre primer, la mitjana dels períodes de les successions periòdiques otingudes a partir d'una aplicació **aleatòria** $f : (\mathbb{Z}/p\mathbb{Z}) \longrightarrow (\mathbb{Z}/p\mathbb{Z})$ és de l'ordre de $\sqrt{\dfrac{\pi p}{8}}$ (cf. **[Co 93**, punt 8.5.4**]**).

Ara, notem que encara que una funció polinòmica no sigui aleatòria, és un exercici senzill provar que tota aplicació $f : (\mathbb{Z}/p\mathbb{Z}) \longrightarrow (\mathbb{Z}/p\mathbb{Z})$ és polinòmica. I per a polinomis no lineals, "sembla" que les funcions polinòmiques es comporten aleatòriament.

Doncs, si ens creiem aquest resultat, i que alguna successió que poguem obtenir tingui un període i un preperíode d'aquest ordre, si el menor nombre primer p que divideix n és de l'ordre de 10^r, el període i el preperíode serien de l'ordre de $10^{r/2}\sqrt{\dfrac{\pi}{8}}$; o sigui, de l'ordre de $0.62 \cdot 10^{r/2}$.

Per exemple, si p és de l'ordre de 10^{14}, podríem obtenir períodes i preperíodes de l'ordre de $6.2 \cdot 10^6$, una quantitat de càlculs que sembla accessible.

7.2.2. La tria de la funció

Els càlculs que haurem de fer són, òbviament, mòdul n, on n és el nombre que volem factoritzar, i calcular valors de la forma $\gcd(x_u - x_v, n)$, per a elements $x_u = f^u(x_0)$, $x_v = f^v(x_0)$, on f és l'aplicació polinomica de $\mathbb{Z}/n\mathbb{Z}$ en si mateix. Per tant, és convenient que el polinomi f sigui senzill, però no de grau 1. La tria més òbvia sembla ser un polinomi de la forma $X^2 + a$, per a un element $a \in \mathbb{Z}/n\mathbb{Z}$, que triarem a l'atzar.

7.2.3. Els valors per a la comparació

Ja només resta decidir quins valors podem comparar a fi de fer més probable obtenir el període i el preperíode de la successió mòdul un divisor no trivial de n.

Una manera de fer-ho és calcular simultàniament les successions x_k i $y_k := x_{2k}$. Així, si k és un múltiple del període (mòdul d) que sigui més gran que el preperíode (mòdul d), tindrem que $x_k \equiv y_k \pmod{d}$, i el càlcul de $\gcd(x_k - y_k, n)$ proporcionarà un múltiple de d que divideix n; probablement, el divisor propi d.

Observació. Aquesta tria és l'original del mètode rho. Tot i que hi ha diferents millores que fan el mètode més eficient, i algunes prou senzilles d'aplicar, ens limitarem a aquest algoritme original.

Ara, notem que $y_k = x_{2k} = f^k(f^k(x_0)) = f^k(x_k)$, i que $y_{k+1} = f(f(y_k))$, de manera que les dues successions $\{x_k\}_{k\geq 0}$ i $\{y_k\}_{k\geq 0}$ es poden calcular simultàniament.

Finalment, només cal afegir límits per al nombre de comparacions que farem, a fi que el procés acabi. Això permet escriure la funció següent.

7.3. La funció PollardRho(nn,tt,ff)

Anomenarem nn el nombre que volem factoritzar, tt el límit de comparacions que volem fer, i ff el nombre màxim de funcions que usarem, en cas que no aconseguim factoritzar nn abans.

En efecte, si després d'una tria a l'atzar de la funció $f:\ x \mapsto x^2 + a$, o sigui, del valor de $a \pmod{n}$, i de tt comparacions no obtenim un factor no trivial de nn, canviarem el valor de a un màxim de ff vegades. I si no ens en sortim, aturarem el procés sense factoritzar. I si en alguna comparació obtenim un divisor no trivial d de nn, retornarem la parella de factors d i nn/d.

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, cosa que suposarem.

In [12]:

```
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

```

7.3.0. Exemples

Els factors són nombres primers que s'han obtingut alguna vegada per un tria a l'atzar.

In [13]:

```
PollardRho(538736922377*337991527361*304821096639811,10^6,15)
```

Out[13]:

```
[337991527361, 164218379479313874234950747]
```

In [14]:

```
PollardRho(538736922377*304821096639811,10^6,15)
```

Out[14]:

```
[538736922377, 304821096639811]
```

In [15]:

```
PollardRho(79059099415544842823*304821096639811,10^8,15)
```

Out[15]:

```
[304821096639811, 790590994155448428 23]
```

In [16]:

```
PollardRho(538736922377*337991527361^2*304821096639811,10^6,15)
```

Out[16]:

```
[538736922377, 34822235522361021397702380630784443331]
```

In [17]:

```
PollardRho(538736922377^2*337991527361^2*304821096639811,10^6,15
```

Out[17]:

```
[337991527361, 29902280894501683887359641205254047816909247001459]
```

In [18]:

```
PollardRho(538736922377^2*337991527361^2*304821096639811,10^6,15
```

Out[18]:

```
[337991527361, 29902280894501683887359641205254047816909247001459]
```

Naturalment, com que el procés té una bona part d'aleatorietat, no sempre s'obté el mateix resultat. Per exemple, en algunes proves inicials s'han obtingut aquests resultats per a PollardRho(538736922377^2·337991527361^2·304821096639811,10^6,15):

[538736922377, 187600239956038216325044234800350269434321023177877]

[337991527361, 29902280894501683887359641205254047816909247001459]

7.4. Tercera versió de la funció Factoritza(nn)

Com que la funció Factoritza(nn) comença a ser una mica complicada, convé comentar què fa cada secció del codi. Juntament amb la incorporació de la funció PollardRho(nn,tt,ff), hi afegirem alguns comentaris.

In [19]:

```
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

```

7.4.0. Exemples

In [20]:

```
Factoritza(factorial(37)*538736922377^2*304821096639811)
```

Out[20]:

```
[[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 [21]:
```
1 Factoritza(factorial(37)*538736922377^2*337991527361*30482109663
```
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],
 [337991527361, 1, '?'],
 [538736922377, 2, '?'],
 [304821096639811, 1, '?']]
```
In [22]:
```
1 Factoritza(factorial(37)*538736922377^2*337991527361^2*30482109
```
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, 2, '?'],
 [538736922377, 2, '?'],
 [304821096639811, 1, '?']]
```
In [23]:
```
1 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, '?'],
 [929159009566969969150701115721, 1, '**']]
```

```
In [24]:  1  Factoritza(factorial(37)*538736922377^2*304821096639811*(factoria
```

```
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],
          [538736922377, 2, '?'],
          [36141868641810568643239911209838214694276122747928795795691090792
         639811,
           1,
           '**']]
```

Fi del capítol 7