Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 1. Tests de primeritat

1.0. Introducció

En aquest capítol presentarem els tests de primeritat de Solovay-Strassen i de Miller-Rabin. Són algoritmes que produeixen (i poden proporcionar-lo), amb probabilitat tan gran com es vulgui, un nombre enter que permet assegurar (**cert-**ifica), amb un càlcul senzill, que el nombre és compost.

Per a la sortida dels algoritmes tenim tres possibilitats. Que proporcionin "Fals", i llavors el nombre test no és primer amb total seguretat; que proporcionin "True" (això només succeeix en pocs casos), i llavors el nombre test és primer amb total seguretat; o que proporcionin "Indeterminat", i això vol dir que hi ha una certa probabilitat que el nombre objecte de la prova no sigui primer, però el test no ho hagi detectat.

1.1. Tests o certificats?

1.1.0. Sobre la terminologia

Comencem aquesta secció amb un exemple sobre un ús inadequat de la terminologia que se sent sovint.

La frase: "**donat** un nombre entre 1 i 100, quina probabilitat té de ser primer?" no té sentit. En canvi, sí que en té la frase "quina probabilitat hi ha que un nombre **triat a l'atzar** (i amb probabilitat equirepartida o, si es vol, amb distribució uniforme) entre 1 i 100 sigui primer?"

En aquest exemple, com que hi ha 25 nombres primers entre 1 i 100, la probabilitat és 25/100=1/4. Això diu que de cada quatre nombres que triem, un serà primer (en mitjana). Però un cop en tenim un, aquest o bé és primer, o bé és compost, però no "probablement" res.

Efectivament, no té sentit dir que un nombre és "probablement primer". De fet, donat un nombre natural n>1, o bé n és primer, o bé és n és compost, i això no és subjecte a cap probabilitat; per tant, la frase "n és probablement primer" només té sentit si és *dins* un context que n'hi doni.

Una cosa diferent és parlar de *la probabilitat que la tria a l'atzar (i d'una manera ben determinada) d'un nombre d'un conjunt on n'hi hagi de primers i de no primers proporcioni*

un nombre primer; la probabilitat és associada a la tria, no al nombre. Notem que, a més a més, cal establir clarament el procés de com es fa la tria del nombre, a fi que l'espai de probabilitat quedi fixat.

Per exemple, com triem un nombre natural arbitrari (o primer)? Què vol dir obtenir el nombre? Potser, donar-ne una expressió decimal (o en una base qualsevol)? I si és així, com triem les xifres? I quantes xifres hem de triar? De fet, si ens creiem que a l'Univers hi ha una quantitat finita de partícules elementals, no podem tenir "escrites" simultàniament més xifres que aquest nombre de partícules, de manera que el nombre no pot ser "qualsevol" nombre natural. Per tant, la tria difícilment pot ser d'un nombre natural "arbitrari"...

Així, doncs, la frase *"probablement, aquest nombre és primer"* pot tenir sentit, mentre que la frase *"aquest nombre és probablement primer"* no en té.

1.1.1. (Pseudo)definició

A l'hora d'establir algoritmes per a determinar la primeritat o no d'un nombre natural, es fan servir diferents tipus d'assaigs. Alguns, són determinístics; és a dir, determinen amb tota seguretat que el nombre és primer. Altres són probabilístics; però, què vol dir això? Que la propietat se satisfà "probablement"? Veurem més avall que això només pot tenir sentit en contextos molt restringits. La terminologia usual que es fa servir és la següent.

Un *test* (probabilístic) de primeritat és un algoritme (entès com una prova o un conjunt determinat de proves) que, aplicat a un nombre natural, determina amb total seguretat que el nombre no és primer o bé, amb una certa probailitat d'error, que el nombre és primer.

Un *certificat* (probabilístic) de primeritat és un algoritme que, aplicat a un nombre natural, determina amb total seguretat que el nombre és primer o bé, amb una certa probailitat d'error, que el nombre no és primer.

En els tests de primeritat que es fan servir habitualment, la probailitat que, aplicats a un nombre natural no primer n>0 el test no ho detecti es pot fer, a base d'aplicar-lo repetidament, tan petita com es vulgui.

1.2. Test de Solovay-Strassen

En aquesta secció presentem una implementació bàsica del test de primeritat de Solovay-Strassen. La teoria en què es fonamenta es pot veure en **[Tr-Ar**, cap. VI, secció 3**]**; a continuació, en fem un resum.

Teorema. Per a un nombre enter senar n>1, el nombre n és primer si, i només si, el morfisme de grups $\psi_n : b \mapsto b^{\frac{n-1}{2}} \left(\frac{b}{n}\right)$, de $(\mathbb{Z}/n\mathbb{Z})^*$ en $(\mathbb{Z}/n\mathbb{Z})^*$ i on $\left(\frac{b}{n}\right)$ és el símbol de Jacobi, és el morfisme trivial.

Tenim, també, que per a un nombre enter *senar* n>1, si n és compost, la probabilitat que f elements invertibles $b \in (\mathbb{Z}/n\mathbb{Z})^*$ triats a l'atzar, de manera independent, i amb probabilitat equirepartida, siguin tots del nucli del morfisme ψ_n és menor o igual que 2^{-f}.

1.2.0. El test

La funció següent, **SolovayStrassenTest(nn,ff)**, implementa aquests resultats. S'aplica a un nombre enter nn>0 i una fita ff per al nombre de proves a l'atzar i proporciona **True** per a nn=2, 3, 5, o 7, **False** per a nn=1 o nn compost, o **'Indeterminat'** en cas que no s'hagi provat que nn és compost en ff proves a l'atzar. També avisa si no es vol fer cap prova (o sigui, si ff < 1).

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'

```

1.2.1. Exemples

Provem el test per a us quants valors de nn i comprovem que la sortida és correcta.

In [2]:
```
print([[k,SolovayStrassenTest(k,10)] for k in range(10)])
```

```
[[0, False], [1, False], [2, True], [3, True], [4, False], [5, Tru
e], [6, False], [7, True], [8, False], [9, False]]
```

In [3]:
```
print([[k,SolovayStrassenTest(k,1/2)] for k in [9,10,11,12]])
```

```
[[9, 'Cal fer alguna prova.'], [10, False], [11, 'Cal fer alguna pr
ova.'], [12, False]]
```

Notem que per a n=1, per als nombres primers menors que 10, o per als nombres parells, no cal fer cap prova.

1.2.2. Aplicació

Podem usar el test per a trobar el primer nombre primer (en l'ordre natural i amb una certa probabilitat d'error) a partir d'un determinat nombre.

In [4]:
```
p1=328752101956024652001111111089
p2=328752101956024652001111111207
p2-p1
```

Out[4]: 118

In [5]:
```
print([SolovayStrassenTest(p1+i,10) for i in range(p2-p1+1)])
```

```
['Indeterminat', False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False
, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False
, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False
, False, False, False, False, 'Indeterminat']
```

Podem dir, amb una certa probabilitat d'error, menor que $2^{-10} = \frac{1}{1024}$ per a cadascun, que p1 i p2 són primers, i amb total seguretat que els nombres intermedis són compostos.

Notem que a partir del nombre primer p1 hi ha n=p2-p1-1=117 nombres consecutius compostos, i que p1 és molt més petit que (n+1)!+2=118!+2, que és el nombre que es fa servir usualment per a provar aquest resultat (cf. **[Tr-Ar**, cap. II, secció 5.1**]**).

In [6]:
```
factorial(118)+2
```

Out[6]:
```
468452584975429065657431236280838416439267950499862031533310318788629800927518416622330123618486343228862579684398745837012213486653229822121742374957258403779058860032000000000000000000000000002
```

1.2.2.0. Observació

Aquest valor de p1 ha estat fruit d'una casualitat. Volia trobar un nombre primer gran a partir d'escriure un nombre amb el teclat tot pitjant tecles de manera aleatòria (si és que això té sentit). El nombre inicial ha estat el nombre 328752101956024652001; en provar-hi el test, sortia compost, i he anat afegint una xifra 1 cada vegada. Però aviat me n'he cansat i, a partir del nombre 328752101956024652001111111111 (que acaba en 9 uns), he cercat quin és el menor nombre que passava el test; el resultat ha estat el nombre p2. I a partir d'aquí, la recerca a la inversa ha permès trobar fàcilment p1.

El resultat han estat els nombres p1=328752101956024652001111111089 i p2=328752101956024652001111111207.

1.2.3. Observació

Per a aquesta tasca, *SageMath* té implementada la funció **next_prime( )**.

In [7]:
```
1 next_prime(p1)
```

Out[7]:
```
328752101956024652001111111207
```

1.2.4. Exercici

Quantes xifres 1 cal afegir al nombre 32875210195602465200 a fi d'otenir un nombre primer?

1.2.4.0. Solució

In [8]:
```
 1 m=328752101956024652 00
 2 f=0
 3 ff=50
 4 while f<ff:
 5     m=10*m+1
 6     if SolovayStrassenTest(m,10)==false:
 7         f=f+1
 8     else:
 9         print('Trobat!')
10         print(f,m)
11 print('Fi')
```

```
Trobat!
23 328752101956024652001111111111111111111111111
Fi
```

1.2.4.1. Resposta

23 (com a mínim)

1.3. Test de Miller-Rabin

En aquesta secció presentem una implementació bàsica del test de primeritat de Miller-Rabin; la teoria en què es fonamenta es pot veure en **[Tr-Ar**, cap. VI, seccions 4 i 5**]**. A continuació en presentem un resum.

Per a tot nombre enter senar n>1, que escrivim en la forma $n = 1 + 2^v m$, amb $v \geq 1$ i m senar, definim el subconjunt S_n de $(\mathbb{Z}/n\mathbb{Z})^*$ format pels elements b tals que $b^m \equiv 1 \pmod{n}$, o bé existeix k, $0 \leq k \leq v-1$, tal que $b^{2^k m} \equiv -1 \pmod{n}$.

Es té que el nombre n és primer si, i només si, $S_n = (\mathbb{Z}/n\mathbb{Z})^*$ (cf. **[Tr-Ar**, cap. VI, punts 4.4 i 4.12**]**).

I si n>3 és senar i compost, la probabilitat que f nombres enters b, $2 \leq b \leq n-2$ i triats a l'atzar i amb probabilitat equirepartida, pertanyin tots a S_n és estrictament menor que 4^{-f}

(cf. **[Tr-Ar**, cap. VI, punt 5.3**]**).

1.3.0. El test

La funció següent, **MillerRabinTest(nn,ff)**, implementa aquests resultats. S'aplica a un nombre enter nn>0 i una fita ff per al nombre de proves a l'atzar, i proporciona True per a nn=2, 3, 5, o 7, False per a nn=1 i per a nn compost, i 'Indeterminat' en cas que no s'hagi provat que nn és compost en ff proves a l'atzar. També avisa si no es vol fer cap prova (o sigui, si ff < 1).

In [9]:
```
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'

```

1.3.1. Exemples

Provem el test per a us quants valors de nn i comprovem que la sortida és correcta.

In [10]:
```
print([[k,MillerRabinTest(k,10)] for k in range(10)])
```

```
[[0, False], [1, False], [2, True], [3, True], [4, False], [5, Tru
e], [6, False], [7, True], [8, False], [9, False]]
```

In [11]:
```
print([[k,MillerRabinTest(k,1/2)] for k in [9,10,11,12]])
```

```
[[9, 'Cal fer alguna prova.'], [10, False], [11, 'Cal fer alguna pr
ova.'], [12, False]]
```

In [12]:
```
1 print([MillerRabinTest(p1+i,10) for i in range(p2-p1+1)])
```

```
['Indeterminat', False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False
, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False
, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False
, False, False, False, False, 'Indeterminat']
```

Obtenim els mateixos resultats que amb el test de Solovay-Strassen (com hauria de ser!).

Podem dir, però ara amb una probabilitat d'error menor que $4^{-10} = \frac{1}{1024^2} = \frac{1}{1048576}$ per a cadascun, que p1 i p2 són primers; i, amb total seguretat, que els nombres intermedis són compostos.

1.4. Certificació de nombres compostos

Els tests de primeritat de Solovay-Strassen o de Miller-Rabin poden proporcionar un certificat quan el nombre al qual s'apliquen és compost.

En efecte, notem que en cas que el nombre sigui parell i més gran que 2, g=2 és un valor de g per al qual el test de Solovay-Strassen permet assegurar que nn és compost. I si nn és senar compost, això succeeix per al valor de g triat a l'atzar per al qual és $y \neq x$. Per tant, a la sortida de la funció podem afegir-hi g com a certificat. I análogament amb el test de Miller Rabin.

1.4.0. Certificat de Solovay-Strassen

La funció següent, **SolovayStrassenCert(nn,ff)**, afegeix el certificat al test de Solovay-Strassen. S'aplica a un nombre enter nn i una fita ff per al nombre de proves a l'atzar, i proporciona [nn,True,nn-1,[nn-1]] per a nn=2 o 3, [5,True,2,[2]], per a nn=5, [7,True,3,[2,3]], per a nn=7, [1,False,1] per a n=1, [nn,False,g], per a nn compost, on g és un valor que certifica que nn és compost, o 'Indeterminat' en cas que no s'hagi provat que nn és compost en ff proves a l'atzar.

In [13]:
```
def SolovayStrassenCert(nn,ff):
    if nn==1:
        return [nn,false,1]
    if nn==2 or nn==3:
        return [nn,true,nn-1,[nn-1]]
    if nn==5:
        return [nn,true,2,[2]]
    if nn==7:
        return [nn,true,3,[2,3]]
    if is_even(nn):
        return [nn,false,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 [nn,false,g]
        else:
            return [nn,false,g]
        f=f+1
    return 'Indeterminat'

```

1.4.1. Exemples

In [14]:
```
print([SolovayStrassenCert(k,10) for k in range(10)])
```

```
[[0, False, 2], [1, False, 1], [2, True, 1, [1]], [3, True, 2,
[2]], [4, False, 2], [5, True, 2, [2]], [6, False, 2], [7, True, 3,
[2, 3]], [8, False, 2], [9, False, 2]]
```

In [15]:
```
print([[k,SolovayStrassenCert(k,10)] for k in [11,15,21]])
```

```
[[11, 'Indeterminat'], [15, [15, False, 4]], [21, [21, False, 18]]]
```

In [16]:
```
print([[k,SolovayStrassenCert(p1+k,10)] for k in range(5)])
```

```
[[0, 'Indeterminat'], [1, [3287521019560246520011111090, False,
2]], [2, [3287521019560246520011111091, False, 3222633882854087627
7429281805]], [3, [3287521019560246520011111092, False, 2]], [4,
[3287521019560246520011111093, False, 3226840386449548992735671464
9]]]
```

In [17]:
```
SolovayStrassenCert(p1*p2,10)
```

Out[17]:
```
[10807794454050442782030139970083604289200164448872098744 23,
 False,
 39348711510873776204605334690918289194193511699556847 5878]
```

In [18]:
```
1 SolovayStrassenCert(p1*p2+1,10)
```

Out[18]:
```
[108077944540504427820301399700836042892001644488720987442 4, False, 2]
```

In [19]:
```
1 SolovayStrassenCert(p1*p2+18,10)
```

Out[19]:
```
'Indeterminat'
```

1.4.2. Certificat de Miller-Rabin

La funció següent, **MillerRabinCert(nn,ff)**, afegeix el certificat al test de Miller-Rabin. S'aplica a un nombre enter nn i una fita ff per al nombre de proves a l'atzar, i proporciona [nn,True,nn-1,[nn-1]] per a nn=2 o 3, [5,True,2,[2]], per a nn=5, [7,True,3,[2,3]], per a nn=7, [1,False,1] per a n=1, [nn,False,g], per a nn compost, on g és un valor que certifica que nn és compost, o 'Indeterminat' en cas que no s'hagi provat que nn és compost en ff proves a l'atzar.

In [20]:
```
def MillerRabinCert(nn,ff):
    if nn==1:
        return [nn,false,1]
    if nn==2 or nn==3:
        return [nn,true,nn-1,[nn-1]]
    if nn==5:
        return [nn,true,2,[2]]
    if nn==7:
        return [nn,true,3,[2,3]]
    if is_even(nn):
        return [nn,false,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 [nn,false,g]
        f=f+1
    return 'Indeterminat'

```

1.4.3. Exemples

In [21]:
```
1 print([MillerRabinCert(k,10) for k in range(10)])
```

```
[[0, False, 2], [1, False, 1], [2, True, 1, [1]], [3, True, 2,
[2]], [4, False, 2], [5, True, 2, [2]], [6, False, 2], [7, True, 3,
[2, 3]], [8, False, 2], [9, False, 4]]
```

In [22]:
```
1 print([[k,MillerRabinCert(k,10)] for k in [11,15,21]])
```

```
[[11, 'Indeterminat'], [15, [15, False, 3]], [21, [21, False, 8]]]
```

In [23]:
```
1 print([[k,MillerRabinCert(p1+k,10)] for k in range(5)])
```

```
[[0, 'Indeterminat'], [1, [3287521019560246520011111090, False,
2]], [2, [3287521019560246520011111091, False, 1590954128295591567
1071508697]], [3, [3287521019560246520011111092, False, 2]], [4,
[3287521019560246520011111093, False, 7418021020734309235311707743]]]
```

In [24]:
```
1 MillerRabinCert(p1*p2,10)
```

Out[24]:
```
[10807794454050442782030139970083604289200164448872098744 23,
 False,
 72743628344051211765890408898287971796489926263729833 2772]
```

In [25]:
```
1 MillerRabinCert(p1*p2+1,10)
```

Out[25]:
```
[10807794454050442782030139970083604289200164448872098744 24, False,
2]
```

In [26]:
```
1 MillerRabinCert(p1*p2+2,10)
```

Out[26]:
```
[10807794454050442782030139970083604289200164448872098744 25,
 False,
 49501024468969729375743598140000696961025337539450786719 6]
```

In [27]:
```
1 MillerRabinCert(p1*p2+2,10)
```

Out[27]:
```
[10807794454050442782030139970083604289200164448872098744 25,
 False,
 10734837494088450446316050172600463788799236007013430085 90]
```

In [28]:
```
1 MillerRabinCert(p1*p2+18,10)
```

Out[28]:
```
'Indeterminat'
```

1.5. Comparació dels tests

Es tracta de comparar els tests de primeritat de Solovay-Strassen i de Miller-Rabin (de fet, les implementacions que n'hem fet) en dues situacions diferents.

En primer lloc, construirem una llista aleatòria de 10000 nombres naturals menors que 10^{30} i compararem els resultats dels dos tests. Proporcionen el mateix resultat? I, d'altra banda, quant de temps triga cadascun?

Després construirem una llista aleatòria de 10000 nombres naturals primers, també menors

que 10^{30}, i compararem, com per a la llista anterior de nombres naturals, els resultats dels dos tests i els temps que triga cadascun.

Podem dir que algun dels mètodes és millor que l'altre? Per què? Quin tarda més?

Notem que en tots els casos es poden fer les proves amb una fita ff=2. Però a fi de tenir la mateixa probabilitat d'error, convé que la fita del test de Solovay-Strassen sigui el doble que la fita del test de Miller-Rabin.

In [29]:
```
1 n=10000
```

In [30]:
```
1 valor=10^30
```

1.5.0. Per a nombres arbitraris

In [31]:
```
1 nr=[ZZ.random_element(1,valor) for i in range(n)]
```

In [32]:
```
1 t=[[a:=SolovayStrassenTest(nr[i],1),b:=MillerRabinTest(nr[i],1),
```

In [33]:
```
1 comp=[[i,t[i][0],t[i][1]] for i in range(len(t)) if t[i][2]==fal
```

In [34]:
```
1 comp
```

Out[34]:
```
[]
```

In [35]:
```
1 %time x=[SolovayStrassenTest(nr[i],2) for i in range(len(nr))]
```

```
CPU times: user 2.02 s, sys: 24.2 ms, total: 2.05 s
Wall time: 2.05 s
```

In [36]:
```
1 %time y=[MillerRabinTest(nr[i],2) for i in range(len(nr))]
```

```
CPU times: user 2.12 s, sys: 24.1 ms, total: 2.14 s
Wall time: 2.14 s
```

In [37]:
```
1 %time x=[SolovayStrassenTest(nr[i],4) for i in range(len(nr))]
```

```
CPU times: user 1.98 s, sys: 24 ms, total: 2.01 s
Wall time: 2.01 s
```

In [38]:
```
1 %time x=[SolovayStrassenTest(nr[i],4) for i in range(len(nr))]
```

```
CPU times: user 4.22 s, sys: 40.2 ms, total: 4.26 s
Wall time: 4.26 s
```

1.5.1. Per a nombres primers

In [39]:
```
1 %time pr=[random_prime(valor) for i in range(n)]
```

```
CPU times: user 1min 20s, sys: 20.4 ms, total: 1min 20s
Wall time: 1min 20s
```

In [40]:
```
t2=[[a:=SolovayStrassenTest(pr[i],1),b:=MillerRabinTest(pr[i],1)
```

In [41]:
```
comp2=[[i,t2[i][0],t2[i][1]] for i in range(len(t2)) if t2[i][2]=
```

In [42]:
```
comp2
```

Out[42]: []

In [43]:
```
%time x1=[SolovayStrassenTest(pr[i],2) for i in range(len(pr))]
```

```
CPU times: user 3.96 s, sys: 44 ms, total: 4.01 s
Wall time: 4 s
```

In [44]:
```
%time y1=[MillerRabinTest(pr[i],2) for i in range(len(pr))]
```

```
CPU times: user 3.76 s, sys: 48 ms, total: 3.81 s
Wall time: 3.81 s
```

In [45]:
```
%time x1=[SolovayStrassenTest(pr[i],4) for i in range(len(pr))]
```

```
CPU times: user 9.71 s, sys: 108 ms, total: 9.82 s
Wall time: 9.82 s
```

In [46]:
```
%time x1=[SolovayStrassenTest(pr[i],4) for i in range(len(pr))]
```

```
CPU times: user 7.52 s, sys: 44 ms, total: 7.56 s
Wall time: 7.56 s
```

In [47]:
```
%time y1=[MillerRabinTest(pr[i],2) for i in range(len(pr))]
```

```
CPU times: user 4.92 s, sys: 60 ms, total: 4.98 s
Wall time: 4.98 s
```

1.5.2. Conclusions

(aquestes conclusions són per a algunes proves a l'atzar; altres proves proporcionen conclusions diferents!)

Per tant, caldria fer proves més concloents!

Per a nombres naturals triats a l'atzar, el test de Miller-Rabin ha trigat més, fins i tot en el cas d'aplicar el de Solovay-Strassen amb una fita doble a fi de millorar la probabilitat d'aquest fins a la probabilitat de Miller-Rabin.

En canvi, si passem els tests només a nombres primers, en tots els casos sembla que el test de Miller-Rabin funciona més ràpidament.

A més a més, les diferències en els temps no són, probablement, gaire significatives, perquè per al mateix càlcul se n'obtenen de molt diferents.

Això suggereix que, per a fer servir el test com a garbell per a determinar nombres primers d'entre nombres naturals arbitraris, sembla millor usar el test de Solovay-Strassen; en canvi, si es tracta de "confirmar" que certs nombres "són" primers, sembla millor usar el test de Miller-Rabin.

Notem que en tots els casos ens referim a aquestes implementacions dels tests; per a altres implementacions, els resultats poden ser diferents (i, fins i tot, contradictoris amb aquests).

Fi del capítol 1