Ejercicio8

March 21, 2022

1 Ejercicio 8

Toma tu número n de la lista publicada para este ejercicio.

1. Pasa algunos tests de primalidad para ver si n es compuesto.

[1]:
```
from math import gcd

def f(x):
    return x*x+1

def rho_de_polard(n,imprime=False):
    x=1
    y=1
    contador=0
    resultado=1

    if imprime:
        print("Iteracion ", contador)
        print("x: ",x," y:",y , " mcd: ", resultado)

    while resultado==1 or resultado == n:
        x=f(x)%n
        y=f(f(y))%n
        resultado=gcd(x-y,n)
        contador+=1

        if imprime:
            print("Iteracion ", contador)
            print("x: ",x," y:",y , " mcd: ", resultado)
        if 1<resultado<n:
            return resultado

    return "No hay divisores"

def exponenciacion_rapida_izda_dcha (a,exp,m,imprime=False):
    c=0
    num_binario= bin(exp)[2:]
```

```python
    acu=1
    for i in num_binario:
        c=2*c
        acu=(acu*acu)%m

        if i=='1':
            c+=1
            acu=(acu*a)%m

    return acu

def simbolo_jacobi(a,n):
    t=1
    m=abs(n)
    b=a%m

    while a != 0:
        while a%2==0:
            a=a/2
            if m%8==3 or m%8==5:
                t=-t



        aux=a
        a=m
        m=aux
        if a%4==m%4==3:
            t=-t


        a=a%m



    if m==1:
        return t
    else:
        return 0



def comprobacion_lucas_lehmer(a,divisores,n):
    resultado=True

    for i in divisores:
        if exponenciacion_rapida_izda_dcha(a,(n-1)//i,n) == 1:
            resultado=False
```

```python
    return resultado

def Lucas_Lehmer(n,divisores):
    i=1
    test1=False
    test2=False
    res=0

    while i>0:
        i+=1
        res=exponenciacion_rapida_izda_dcha (i,n-1,n)

        if res==1:
            test1=True

        if comprobacion_lucas_lehmer(i,divisores,n):
            test2=True

        if test1 & test2:
            print("El natural más pequeño cuya clase es primitiva: ", i)
            return True
        else:
            test1=False
            test2=False

def obtener_exponentes_millner_rabin(n):
    valor=n-1
    resultado=[valor]
    while valor%2==0:
        valor=valor//2
        resultado.append(valor)

    return resultado
```

Mi número es el siguiente:

[2]:
```python
n=130808809952929773666391103613183595579
```

En primer lugar vamos a probar con el TPF:

[3]:
```python
primos = [2,3,5,7,11]

for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,n-1,n)
    print(resultado)
    print("")
```

```
Probamos con la base 2
1

Probamos con la base 3
1

Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 11
1
```

Como vemos, se trata de un posible primo de Fermat para toda base probada, vamos a comprobar ahora si pasa el test de Solovay-Strassen.

Recordemos que El Test de Solovay-Strassen es el siguiente:

$$(\frac{a}{n}) \equiv a^{(n-1)/2} \ mod \ n$$

Se dirá que n es un posible primo de Euler para a, y en caso contrario se tendrá que n es compuesto.

Calculemos primero el símbolo de Jacobi con los 5 primeros primos:

[4]:
```
for i in primos:
    s=simbolo_jacobi(i,n)
    print("simbolo de Jacobi obtenido para", i,"/ n = ",s)
```

```
simbolo de Jacobi obtenido para 2 / n =  -1
simbolo de Jacobi obtenido para 3 / n =  1
simbolo de Jacobi obtenido para 5 / n =  1
simbolo de Jacobi obtenido para 7 / n =  -1
simbolo de Jacobi obtenido para 11 / n =  -1
```

Calculamos ahora la parte derecha de la equivalencia:

[5]:
```
exp=(n-1)//2

print("El exponente es:",exp)
print("")

for i in primos:
    print("Probamos con la base:", i)
    resultado=exponenciacion_rapida_izda_dcha(i,exp,n)
    print(resultado)
    print("")
```

```
El exponente es: 6540440497646488683319555180659179789

Probamos con la base: 2
13080880995292977366639110361318359578

Probamos con la base: 3
1

Probamos con la base: 5
1

Probamos con la base: 7
13080880995292977366639110361318359578

Probamos con la base: 11
13080880995292977366639110361318359578
```

Como podemos ver, ser verifica el test de Solovay-Strassen, por lo que n sería posible primo de Euler para toda base probada.

A continuación vamos a comprobar el test de **Millner-Rabin**, el cual se basa en que dado que $n-1 = 2^r m$ con m impar, para una base a, se calcula la siguiente a-sucesión para n:

$$a^m, \; a^{2m}, \; \ldots \; a^{2^r m} = a^{n-1}$$

Si esta sucesión no acaba en 1 o bien hay un 1 precedido de un número que no es +-1, se tendría que n es compuesto.

Por lo tanto en primer lugar calculamos los exponentes que necesitaremos para Millner-Rabin:

[6]:
```
exp=obtener_exponentes_millner_rabin(n)
print(exp)
```

```
[13080880995292977366639110361318359578, 6540440497646488683319555180659179789]
```

[7]:
```
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,n)
        print(resultado)
        print("")
```

```
Probamos con la base 2
13080880995292977366639110361318359578

Probamos con la base 2
1
```

```
Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 5
1

Probamos con la base 5
1

Probamos con la base 7
13080880995292977366639110361318359578

Probamos con la base 7
1

Probamos con la base 11
13080880995292977366639110361318359578

Probamos con la base 11
1
```

Como vemos pasa también el Test de Millner-Rabin, luego tendríamos que es muy probable que nuestro número sea primo pues la probabilidad de que lo sea es de $1 - 4^{-5} = 0.9990234375$.

2. En caso que tu n sea probable primo. Factoriza n + 1 encontrando certificados de primalidad para los factores mayores de 10000.

[8]:
```
r=n+1
```

Como r es par, podemos aplicar el método ρ de Polard:

[9]:
```
rho_de_polard(r)
```

[9]:
```
3
```

[10]:
```
r//3
```

[10]:
```
4360293665097659122213036787106119860
```

Luego hemos obtenido que $r = 3 \cdot 4360293665097659122213036787106119860$. Como 4360293665097659122213036787106119860 es par, volvemos a aplicarle el método de Polard:

[11]:
```
rho_de_polard(4360293665097659122213036787106119860)
```

[11]:
```
4
```

[12]:
```python
4360293665097659122213036787106119860//4
```

[12]: `1090073416274414780553259196776529965`

Por lo tanto $r = 2^2 \cdot 3 \cdot 1090073416274414780553259196776529965$. Como $1090073416274414780553259196776529965$ vuelve a ser compuesto, aplicamos de nuevo el método de Polard.

[13]:
```python
rho_de_polard(1090073416274414780553259196776529965)
```

[13]: `5`

[14]:
```python
1090073416274414780553259196776529965//5
```

[14]: `218014683254882956110651839355305993`

$r = 2^2 \cdot 3 \cdot 5 \cdot 218014683254882956110651839355305993$. Ahora debemos comprobar si $218014683254882956110651839355305993$ es primo o no, usaremos primero el TPF:

[15]:
```python
primos = [2,3,5,7,11]

for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,218014683254882956110651839355305992,2180146832
    print(resultado)
    print("")
```

```
Probamos con la base 2
146296000867116056997841804864874213

Probamos con la base 3
144954666334629043472539333582367226

Probamos con la base 5
24286857260354978067505378913260083

Probamos con la base 7
185633793891926762298887781772215527

Probamos con la base 11
4296304964425263672777396662616237
```

Concluimos que es compuesto pues no pasa el test para ninguna base y podemos aplicar el método de Polard:

[16]:
```python
rho_de_polard(218014683254882956110651839355305993)
```

[16]: 1801

[17]:
```
218014683254882956110651839355305993//1801
```

[17]: 121052017354182651921516845838593

Por lo tanto $r = 2^2 \cdot 3 \cdot 5 \cdot 1801 \cdot 121052017354182651921516845838593$. Sabemos que 1801 es primo de 4 cifras, luego veamos si 121052017354182651921516845838593 también lo es:

[18]:
```
primos = [2,3,5,7,11]

for i in primos:
    print("Probamos con la base",i)
    ␣
 ↪resultado=exponenciacion_rapida_izda_dcha(i,121052017354182651921516845838592,12105201735418
    print(resultado)
    print("")
```

```
Probamos con la base 2
96208733318604418860197433122660

Probamos con la base 3
116716116089104076217897879872112

Probamos con la base 5
75325448849223898757412478909611

Probamos con la base 7
29907212577479537225500009806593

Probamos con la base 11
79458804368534162791919036928598
```

Como vemos no pasa el TPF para ninguna base, luego podemos aplicarle el método de Polard:

[19]:
```
rho_de_polard(121052017354182651921516845838593)
```

[19]: 654856169969

[20]:
```
121052017354182651921516845838593//654856169969
```

[20]: 184852831668231956497

Obtenemos que $r = 2^2 \cdot 3 \cdot 5 \cdot 1801 \cdot 654856169969 \cdot 184852831668231956497$, vamos a comprobar entonces si 654856169969 es primo:

[21]:
```
primos = [2,3,5,7,11]

for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,654856169968,654856169969)
    print(resultado)
    print("")
```

```
Probamos con la base 2
1

Probamos con la base 3
1

Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 11
1
```

Como vemos se trata de un posible primo de Fermat para toda base probada. Vamos a comprobar también si pasa el test de Millner-Rabin:

[22]:
```
exp=obtener_exponentes_millner_rabin(654856169969)
print(exp)
```

```
[654856169968, 327428084984, 163714042492, 81857021246, 40928510623]
```

[23]:
```
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,654856169969)
        print(resultado)
        print("")
```

```
Probamos con la base 2
393025650191

Probamos con la base 2
654856169968

Probamos con la base 2
1
```

```
Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 3
210316932371

Probamos con la base 3
359226859968

Probamos con la base 3
393025650191

Probamos con la base 3
654856169968

Probamos con la base 3
1

Probamos con la base 5
393025650191

Probamos con la base 5
654856169968

Probamos con la base 5
1

Probamos con la base 5
1

Probamos con la base 5
1

Probamos con la base 7
156134446609

Probamos con la base 7
261830519778

Probamos con la base 7
654856169968

Probamos con la base 7
1
```

```
Probamos con la base 7
1

Probamos con la base 11
156134446609

Probamos con la base 11
261830519778

Probamos con la base 11
654856169968

Probamos con la base 11
1

Probamos con la base 11
1
```

Como vemos pasa el test de Millner-Rabin también, luego es muy probable que sea primo, vamos a buscar un certificado de primalidad con Lucas Lhemer, para ello necesitamos ver los factores primos de $p-1 = 654856169968$. Como es par podemos aplicar el método de Polard:

[24]:
```python
rho_de_polard(654856169968)
```

[24]: 112

[25]:
```python
654856169968//112
```

[25]: 5846930089

Tenemos que $p-1 = 112 \cdot 5846930089 = 2^4 \cdot 7 \cdot 5846930089$. Veamos si 5846930089 es primo:

[26]:
```python
primos = [2,3,5,7,11]

for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,5846930088,5846930089)
    print(resultado)
    print("")
```

```
Probamos con la base 2
2963798566

Probamos con la base 3
1094011374

Probamos con la base 5
2113302283
```

```
Probamos con la base 7
1975671894

Probamos con la base 11
3210754184
```

Como vemos no pasa el TPF, luego podemos aplicar el método de Polard:

[27]:
```python
rho_de_polard(5846930089)
```

[27]: 7

[28]:
```python
5846930089//7
```

[28]: 835275727

Tenemos que $p-1=2^4\cdot 7^2\cdot 835275727$. Veamos si 835275727 es primo:

[29]:
```python
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,835275726,835275727)
    print(resultado)
    print("")
```

```
Probamos con la base 2
38899210

Probamos con la base 3
205294246

Probamos con la base 5
527963903

Probamos con la base 7
661555569

Probamos con la base 11
792668668
```

No pasa el TPF, aplicamos Polard:

[30]:
```python
rho_de_polard(835275727)
```

[30]: 143

[31]:
```python
835275727//143
```

[31]: 5841089

Tenemos que $p-1 = 2^4 \cdot 7^2 \cdot 143 \cdot 5841089 = 2^4 \cdot 7^2 \cdot 11 \cdot 13 \cdot 5841089$. Veamos si 5841089 es primo:

[32]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,5841088,5841089)
    print(resultado)
    print("")
```

```
Probamos con la base 2
16964

Probamos con la base 3
2714499

Probamos con la base 5
3478744

Probamos con la base 7
2908994

Probamos con la base 11
4418515
```

Como vemos no pasa el TPF, aplicamos Polard:

[33]:
```
rho_de_polard(5841089)
```

[33]: 2153

[34]:
```
5841089//2153
```

[34]: 2713

Tenemos que $p-1 = 2^4 \cdot 7^2 \cdot 11 \cdot 13 \cdot 2153 \cdot 2713$, que como vemos es la descomposición en factores primos de $p-1$ pues tanto 2153 como 2713 son primos de 4 cifras.

A continuación aplicamos el método de Lucas Lhemer:

[35]:
```
divisores=[2,7,11,13,2153,2713]
if Lucas_Lehmer(654856169969,divisores):
    print("Es primo")
```

```
El natural más pequeño cuya clase es primitiva:  15
Es primo
```

Por lo que obtenemos un certificado de primalidad de 654856169969, así sólo nos queda ver si 184852831668231956497 es primo y así $r = 2^2 \cdot 3 \cdot 5 \cdot 1801 \cdot 654856169969 \cdot 184852831668231956497$

es una descomposició en factores primos.

[36]:
```
for i in primos:
    print("Probamos con la base",i)
    ␣
↪resultado=exponenciacion_rapida_izda_dcha(i,184852831668231956496,184852831668231956497)
    print(resultado)
    print("")
```

```
Probamos con la base 2
1

Probamos con la base 3
1

Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 11
1
```

Como vemos nuestro número pasa el TPF, veamos si pasa también el de Millner-Rabin:

[37]:
```
exp=obtener_exponentes_millner_rabin(184852831668231956497)
print(exp)
```

```
[184852831668231956496, 92426415834115978248, 46213207917057989124,
23106603958528994562, 11553301979264497281]
```

[38]:
```
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,184852831668231956497)
        print(resultado)
        print("")
```

```
Probamos con la base 2
120962128902418213954

Probamos con la base 2
184172644277461006623

Probamos con la base 2
184852831668231956496
```

```
Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 5
54884018299290647662

Probamos con la base 5
120962128902418213954

Probamos con la base 5
184172644277461006623

Probamos con la base 5
184852831668231956496

Probamos con la base 5
1

Probamos con la base 7
680187390770949874

Probamos con la base 7
184852831668231956496

Probamos con la base 7
1

Probamos con la base 7
1
```

```
Probamos con la base 7
1

Probamos con la base 11
120962128902418213954

Probamos con la base 11
184172644277461006623

Probamos con la base 11
184852831668231956496

Probamos con la base 11
1

Probamos con la base 11
1
```

Como vemos pasa también el Test, luego vamos a intentar buscar un certificado de primalidad para 184852831668231956497, para ello necesitamos primero calcular los factores primos de $q-1 = 184852831668231956496$.

Vamos a aplicar el método ρ de Polard para esto:

[39]:
```python
rho_de_polard(184852831668231956496)
```

[39]: 3

[40]:
```python
184852831668231956496//3
```

[40]: 61617610556077318832

Obtenemos así que $q-1 = 3 \cdot 61617610556077318832$. Como 61617610556077318832 es compuesto volvemos a aplicar el método de Polard:

[41]:
```python
rho_de_polard(61617610556077318832)
```

[41]: 16

[42]:
```python
61617610556077318832//16
```

[42]: 3851100659754832427

$q-1 = 2^4 \cdot 3 \cdot 3851100659754832427$. Veamos si 3851100659754832427 es primo:

[43]:
```python
for i in primos:
    print("Probamos con la base",i)
```

```
    resultado=exponenciacion_rapida_izda_dcha(i,3851100659754832426,3851100659754832427)
    print(resultado)
    print("")
```

```
Probamos con la base 2
1920985706696894229

Probamos con la base 3
1339894463389845543

Probamos con la base 5
3552726607533761854

Probamos con la base 7
1422948862535192848

Probamos con la base 11
884944629100827823
```

Como vemos no pasa el TPF, luego podemos aplicar Polard:

[44]:
```
rho_de_polard(3851100659754832427)
```

[44]: 37

[45]:
```
3851100659754832427//37
```

[45]: 104083801614995471

$q-1 = 2^4 \cdot 3 \cdot 37 \cdot 104083801614995471$. Veamos si 104083801614995471 es primo:

[46]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,104083801614995470,104083801614995471)
    print(resultado)
    print("")
```

```
Probamos con la base 2
73131072943177833

Probamos con la base 3
8742876990176037

Probamos con la base 5
91305305140424834
```

```
Probamos con la base 7
64859609630900826

Probamos con la base 11
25335950832149416
```

Como vemos no pasa el TPF, luego podemos aplicar Polard:

[47]:
```
rho_de_polard(104083801614995471)
```

[47]: 1209457

[48]:
```
104083801614995471//1209457
```

[48]: 86058290303

$q-1 = 2^4 \cdot 3 \cdot 37 \cdot 1209457 \cdot 86058290303$. Veamos si 1209457 es primo:

[49]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,1209456,1209457)
    print(resultado)
    print("")
```

```
Probamos con la base 2
1

Probamos con la base 3
1

Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 11
1
```

Como vemos pasa el TPF para todas las bases, luego vamos a comprobar Millner-Rabin:

[50]:
```
exp=obtener_exponentes_millner_rabin(1209457)
print(exp)
```

```
[1209456, 604728, 302364, 151182, 75591]
```

[51]:
```python
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,1209457)
        print(resultado)
        print("")
```

```
Probamos con la base 2
1077162

Probamos con la base 2
1124235

Probamos con la base 2
1209456

Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 3
132295

Probamos con la base 3
1124235

Probamos con la base 3
1209456

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 5
862158

Probamos con la base 5
1077162

Probamos con la base 5
1124235

Probamos con la base 5
```

```
1209456

Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 11
73487

Probamos con la base 11
113664

Probamos con la base 11
85222

Probamos con la base 11
1209456

Probamos con la base 11
1
```

Como vemos pasa el test también, luego vamos a buscar un certificado de primalidad para 1209457 con Lucas Lhemer.

Para ello necesitamos los factores primos de $x - 1 = 1209456$.

[52]:
```
rho_de_polard(1209456)
```

[52]: 3

[53]:
```
1209456//3
```

[53]: 403152

Obtenemos que $x-1=3\cdot 403152$, volvemos a aplicar Polard:

[54]:
```
rho_de_polard(403152)
```

[54]: 3

[55]:
```
403152//3
```

[55]: 134384

Obtenemos que $x-1=3^2\cdot 134384$, volvemos a aplicar Polard:

[56]:
```
rho_de_polard(134384)
```

[56]: 16

[57]:
```
134384//16
```

[57]: 8399

Obtenemos que $x-1=2^4\cdot 3^2\cdot 8399$. Veamos si 8399 es primo:

[58]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,8398,8399)
    print(resultado)
    print("")
```

```
Probamos con la base 2
7129

Probamos con la base 3
3438

Probamos con la base 5
2028

Probamos con la base 7
6001

Probamos con la base 11
544
```

Como vemos no pasa el test, luego podemos aplicar el método de Polard:

[59]:
```
rho_de_polard(8399)
```

[59]: 37

[60]:
```
8399//37
```

[60]: 227

Obtenemos que $x-1=2^4\cdot 3^2\cdot 37\cdot 227$, que son todos primos. Por lo tanto aplicamos Lucas lhemer para ver la primalidad de $x=1209457$

[61]:
```
divisores=[2,3,37,227]
if Lucas_Lehmer(1209457,divisores):
    print("Es primo")
```

```
El natural más pequeño cuya clase es primitiva:  15
Es primo
```

Como vemos obtenemos un certificado de primalidad y por lo tanto $x=1209457$ es primo.

Nos queda así que $q-1=2^4\cdot 3\cdot 37\cdot 1209457\cdot 86058290303$, y debemos comprobar si 86058290303 es primo o no.

[62]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,86058290302,86058290303)
    print(resultado)
    print("")
```

```
Probamos con la base 2
1

Probamos con la base 3
1

Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 11
1
```

Como vemos el número pasa el TPF, veamos si pasa el test de Millner-Rabin:

[63]:
```
exp=obtener_exponentes_millner_rabin(86058290303)
print(exp)
```

```
[86058290302, 43029145151]
```

[64]:
```
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,86058290303)
        print(resultado)
        print("")
```

```
Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 5
86058290302

Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 11
1

Probamos con la base 11
1
```

Como vemos pasa el test de Millner-Rabin, por lo que vamos a buscar un certificado de primalidad con Lucas-Lhemer, para ello necesitamos la factorización en primos de $y - 1 = 86058290302$.

Comenzamos aplicando el método de Polard:

[65]:
```
rho_de_polard(86058290302)
```

[65]: 2

[66]:
```
86058290302//2
```

[66]: 43029145151

Obtenemos que $y-1 = 2 \cdot 43029145151$, veamos si 43029145151 es primo o no:

[67]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,43029145150,43029145151)
    print(resultado)
    print("")
```

```
Probamos con la base 2
31881959703

Probamos con la base 3
34844636732

Probamos con la base 5
7075685855

Probamos con la base 7
27866873175

Probamos con la base 11
14205523525
```

Como vemos no pasa el TPF y podemos aplicar el método de Polard

[68]:
```
rho_de_polard(43029145151)
```

[68]: 11827

[69]:
```
43029145151//11827
```

[69]: 3638213

Obtenemos que $y-1 = 2 \cdot 11827 \cdot 3638213$, veamos si 11827 es primo o no:

[70]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,11826,11827)
    print(resultado)
    print("")
```

```
Probamos con la base 2
1

Probamos con la base 3
1
```

```
Probamos con la base 5
1

Probamos con la base 7
1

Probamos con la base 11
1
```

Como vemos pasa el TPF, veamos si pasa el Test de Millner-Rabin:

[71]:
```python
exp=obtener_exponentes_millner_rabin(11827)
print(exp)
```

```
[11826, 5913]
```

[72]:
```python
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,11827)
        print(resultado)
        print("")
```

```
Probamos con la base 2
11826

Probamos con la base 2
1

Probamos con la base 3
11826

Probamos con la base 3
1

Probamos con la base 5
11826

Probamos con la base 5
1

Probamos con la base 7
11826

Probamos con la base 7
1
```

```
Probamos con la base 11
1

Probamos con la base 11
1
```

Como vemos también pasa el test de Millner-Rabin, luego vamos a intentar buscar un certificado de primalidad con Lucas Lhemer. Para ello necesitamos la descomposición en primos de $z-1 = 11826$.

Aplicamos el método de Polard

[73]:
```
rho_de_polard(11826)
```

[73]:
```
3
```

[74]:
```
11826//3
```

[74]:
```
3942
```

$z-1 = 3 \cdot 3942 = 2 \cdot 3^4 \cdot 73$, que es una descomposición en factores primos, luego podemos aplicar ya el test de Lucas-Lhemer:

[75]:
```
divisores=[2,3,73]
if Lucas_Lehmer(11827,divisores):
    print("Es primo")
```

```
El natural más pequeño cuya clase es primitiva:  2
Es primo
```

Obtenemos un certificado de primalidad y nos quedaría por ver si de $y-1 = 2 \cdot 11827 \cdot 3638213$ el factor 3638213 es primo o no, para ello necesitamos una descomposición en factores primos de $a-1 = 3638212$.

[76]:
```
rho_de_polard(3638212)
```

[76]:
```
4
```

[77]:
```
3638212//4
```

[77]:
```
909553
```

Obtenemos así que $3638212 = 2^2 \cdot 909553$, veamos si 909553 es primo:

[78]:
```
for i in primos:
    print("Probamos con la base",i)
    resultado=exponenciacion_rapida_izda_dcha(i,909552,909553)
    print(resultado)
    print("")
```

```
Probamos con la base 2
381897

Probamos con la base 3
895142

Probamos con la base 5
813554

Probamos con la base 7
147676

Probamos con la base 11
55946
```

Como vemos no pasa el TPF, luego podemos aplicar el método de Polard:

[79]:
```
rho_de_polard(909553)
```

[79]: 461

[80]:
```
909553//461
```

[80]: 1973

Obtenemos así que $3638212 = 2^2 \cdot 461 \cdot 1973$, que es una descomposición en factores primos, luego podemos aplicar ya el test de Lucas-Lhemer para ver si 3638213 es primo.

[81]:
```
divisores=[2,461,1973]
if Lucas_Lehmer(3638213,divisores):
    print("Es primo")
```

```
El natural más pequeño cuya clase es primitiva:  2
Es primo
```

Obtenemos así un certificado de primalidad de 3638213. Y por lo tanto $q - 1 = 2^4 \cdot 3 \cdot 37 \cdot 1209457 \cdot 86058290303$ es una descomposición en factores primos de 184852831668231956496, luego podemos aplicar Lucas-Lhemer para comprobar si $q = 184852831668231956497$ es primo o no:

[82]:
```
divisores=[2,3,37,1209457,86058290303]
if Lucas_Lehmer(184852831668231956497,divisores):
    print("Es primo")
```

```
El natural más pequeño cuya clase es primitiva:  5
Es primo
```

Luego $q = 184852831668231956497$ es primo, y así obtenemos una descomposición en factores primos de $r = 2^2 \cdot 3 \cdot 5 \cdot 1801 \cdot 654856169969 \cdot 184852831668231956497$.

3. Con P = 1, encuentra el menor Q natural mayor o igual que 2, tal que definan una s.L. que certifique la primalidad de n.

En primer lugar definimos las funciones que nos van a hacer falta:

[83]:
```
def sLucas_modificado(P,Q,r,n):
    U_0=0
    U_1=1
    V=0
    k=0
    aux1=0
    aux2=0
    num_binario= bin(r)[2:]
    for i in num_binario:
        if i=='0':
            aux1=(2*U_0*U_1-P*U_0**2)%n
            aux2=(U_1**2-Q*U_0**2)%n
            U_0=aux1
            U_1=aux2

        if i=='1':
            aux1=(U_1**2-Q*U_0**2)%n
            aux2=(P*U_1**2-2*Q*U_0*U_1)%n
            U_0=aux1
            U_1=aux2

    V=(2*U_1-P*U_0)%n


    return U_0,U_1,V
```

[84]:
```
def comprueba_condicionsLucas(vector_un):
    condicion=False
    for cont,i in enumerate(vector_un,start=1):
        if cont==1 and i==0:
            condicion=True
        else:
            if i==0:
                condicion=False

    return condicion


def certificado_sLucas(P,n,factores_primos):
    vector_un=[]
    Q=1
    r=n+1
    while comprueba_condicionsLucas(vector_un)==False:
```

```
        Q+=1
        vector_un=[]
        for i in factores_primos:
            un,a,v=sLucas_modificado(P,Q,r//i,n)
            vector_un.append(un)

    return P,Q,vector_un
```

Ahora iniciamos el algoritmo con $P = 1$ y $Q = 2$, ponemos el 1 entre los factores primos para calcular el U_r:

[85]:
```
factores_primos=[1,2,3,5,1801,654856169969,184852831668231956497]
P=1
P,Q,vector_un=certificado_sLucas(P,n,factores_primos)
print("P: ",P,"Q:",Q)

cont=0
for i in factores_primos:
    print("U_r/",i,"=",vector_un[cont])
    cont+=1
```

```
P:  1 Q: 17
U_r/ 1 = 0
U_r/ 2 = 1049583371766981984888420001031229994З
U_r/ 3 = 12411904197933058750280893208552100635
U_r/ 5 = 1931227732273175775883114353496327256
U_r/ 1801 = 2949168743207142262611885631504406593
U_r/ 654856169969 = 2398565120431048908090345196961250514
U_r/ 184852831668231956497 = 269542250112392588122960526931976296
```

Como podemos observar hemos encontrado un certificado de primalidad para n con $P = 1$ y $Q = 17$.