Ejercicio7

March 21, 2022

1 Ejercicio 7

Toma tu número n de la lista publicada para el ejercicio 3. Sea d el primer elemento de la sucesión 5, -7, 9, -11, 13, ... que satisface que el símbolo de Jacobi es (d|n) = -1.

[1]:
```
n=4935254983373112921871878907 33

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:]
    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
```

Ahora vamos a tomar el elemento d:

[2]:
```
i=3
contador=1
```

```
d=0
cambio_signo=False

while simbolo_jacobi(d,n)!=-1:
    i+=2
    if contador%2==0:
        d=-i
    else:
        d=i
    contador+=1

print("El d obtenido es: ", d)
```

```
El d obtenido es:  5
```

1. Con P = 1, Q = (1-d)/4, define el e.c. α y sus sucesiones de Lucas asociadas.

En nuestro caso P=1 y Q=(1-5)/4=-1.

Por otro lado $\Delta = P^2 - 4Q = 1 + 4 = 5$ Y como Q es impar, tenemos que $\alpha = \frac{P+\sqrt{\Delta}}{2} = \frac{1+\sqrt{5}}{2}$

Por otro lado las sucesiones de Lucas se definen como:

$V_n = PV_{n-1} - QV_{n-2} = V_{n-1} + V_{n-2}$

$U_n = PU_{n-1} - QU_{n-2} = U_{n-1} + U_{n-2}$

Y como $\alpha^i = \frac{V_i}{2} + \frac{U_i}{2}\sqrt{\Delta}$ tenemos que $V_0 = 2$, $U_0 = 0$, $V_1 = P$, $U_1 = 1$ con lo que podemos empezar la recurrencia para calcular las sucesiones.

El código para calcularlas es el siguiente:

[3]:
```
def sLucas(P,Q,l_max):
    i=2
    V=[2,P]
    U=[0,1]

    while i<l_max:
        V.append(P*V[i-1]-Q*V[i-2])
        U.append(P*U[i-1]-Q*U[i-2])
        i+=1

    return V,U
```

A modo de ejemplo vamos a calcular algunos elementos de las sucesiones de Lucas.

[4]:
```
u,v=sLucas(1,-1,50)
print("Sucesion U:",u)
print()
```

```
print("Sucesion V:",v)
```

```
Sucesion U: [2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364,
2207, 3571, 5778, 9349, 15127, 24476, 39603, 64079, 103682, 167761, 271443,
439204, 710647, 1149851, 1860498, 3010349, 4870847, 7881196, 12752043, 20633239,
33385282, 54018521, 87403803, 141422324, 228826127, 370248451, 599074578,
969323029, 1568397607, 2537720636, 4106118243, 6643838879, 10749957122,
17393796001]

Sucesion V: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352,
24157817, 39088169, 63245986, 102334155, 165580141, 267914296, 433494437,
701408733, 1134903170, 1836311903, 2971215073, 4807526976, 7778742049]
```

2. Si n primo ¿Que debería de pasarle a V_r , U_r , módulo n? ¿Y a $V_{\frac{r}{2}}, U_{\frac{r}{2}}$? Calcula los términos $V_{\frac{r}{2}}$, $U_{\frac{r}{2}}$, V_r , U_r , módulo n, de las sucesiones de Lucas. ¿ Tu n verifica el TPF para el entero cuadrático α ?

Si n fuese primo, por la Tercera Versión del TPF para e.c se tendría que:

$U_r = U_{n-(\frac{\Delta}{n})} = U_{n+1} = 0 \; mod \; n$

$V_r = V_{n-(\frac{\Delta}{n})} = V_{n+1} = 2Q \; mod \; n = -2 \; mod \; n$ En nuestro caso

Por otro lado si tenemos en cuenta las siguientes propiedades:

$U_{2k} = U_k V_k$

$V_{2k} = V_k^2 - 2Q^k$

Si en la última expresión sustituimos $K = \frac{r}{2}$ obtenemos:

$U_r = U_{\frac{r}{2}} V_{\frac{r}{2}} \implies U_{\frac{r}{2}} = \frac{U_r}{V_{\frac{r}{2}}}$

$V_r = V_{\frac{r}{2}}^2 - 2Q^{\frac{r}{2}} = V_{\frac{r}{2}}^2 - 2(-1)^{\frac{r}{2}} \implies V_{\frac{r}{2}}^2 = V_r + 2(-1)^{\frac{r}{2}}$

Como $r = n + 1$ tendríamos que $r = 4935254983373112921871878907 34$ y $\frac{r}{2} =$ 2467627491686556460935939453 67.

El cálculo de los términos de las sucesiones de Lucas que se piden se harán con la siguiente función:

[5]:
```
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
```

[6]:
```
r=n+1
P=1
Q=-1
un,a,v=sLucas_modificado(P,Q,r,n)
print("Sucesion U_r:",un)
print()
print("Sucesion V_r:",v)
```

```
Sucesion U_r: 0

Sucesion V_r: 493525498337311292187187890731
```

[7]:
```
un,a,v=sLucas_modificado(P,Q,r//2,n)
print("Sucesion U_r/2:",un)
print()
print("Sucesion V_r/2:",v)
```

```
Sucesion U_r/2: 0

Sucesion V_r/2: 9946914345835076827126948756
```

Como podemos ver el número n verifica el TP, pues $V_r \equiv 2Q \; mod \; n$ y $U_r \equiv 0 \; mod \; n$.

3. Factoriza r = n+1 y para cada factor primo p suyo, calcula $U_{r/p}$. ¿ Cuál es el rango de Lucas w(n) ?. ¿ Qué deduces sobre la primalidad de tu n ?

[8]:
```
primos_4_cifras=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,␣
 ↪59, 61, 67, 71, 73, 79,
83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251,
257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
```

```
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643,
647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743,
751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853,
857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953,
967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049,
1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129,
1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231,
1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319,
1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439,
1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523,
1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609,
1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709,
1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811,
1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913,
1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017,
2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113,
2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237,
2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333,
2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411,
2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539,
2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657,
2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719,
2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819,
2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927,
2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019, 3023, 3037, 3041,
3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169,
3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271,
3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361, 3371,
3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491,
3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581,
3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677,
3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793,
3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907,
3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003, 4007,
4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099, 4111,
4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229,
4231, 4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283, 4289, 4297, 4327, 4337,
4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421, 4423, 4441, 4447, 4451,
4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549, 4561,
4567, 4583, 4591, 4597, 4603, 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663,
4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733, 4751, 4759, 4783, 4787, 4789,
4793, 4799, 4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919,
4931, 4933, 4937, 4943, 4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999, 5003,
5009, 5011, 5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107,
5113, 5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233,
```

```
5237, 5261, 5273, 5279, 5281, 5297, 5303, 5309, 5323, 5333, 5347, 5351, 5381,
5387, 5393, 5399, 5407, 5413, 5417, 5419, 5431, 5437, 5441, 5443, 5449, 5471,
5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531, 5557, 5563, 5569,
5573, 5581, 5591, 5623, 5639, 5641, 5647, 5651, 5653, 5657, 5659, 5669, 5683,
5689, 5693, 5701, 5711, 5717, 5737, 5741, 5743, 5749, 5779, 5783, 5791, 5801,
5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857, 5861, 5867, 5869, 5879,
5881, 5897, 5903, 5923, 5927, 5939, 5953, 5981, 5987, 6007, 6011, 6029, 6037,
6043, 6047, 6053, 6067, 6073, 6079, 6089, 6091, 6101, 6113, 6121, 6131, 6133,
6143, 6151, 6163, 6173, 6197, 6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257,
6263, 6269, 6271, 6277, 6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343,
6353, 6359, 6361, 6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449, 6451, 6469,
6473, 6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581,
6599, 6607, 6619, 6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703,
6709, 6719, 6733, 6737, 6761, 6763, 6779, 6781, 6791, 6793, 6803, 6823, 6827,
6829, 6833, 6841, 6857, 6863, 6869, 6871, 6883, 6899, 6907, 6911, 6917, 6947,
6949, 6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013, 7019, 7027,
7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121, 7127, 7129, 7151, 7159, 7177,
7187, 7193, 7207, 7211, 7213, 7219, 7229, 7237, 7243, 7247, 7253, 7283, 7297,
7307, 7309, 7321, 7331, 7333, 7349, 7351, 7369, 7393, 7411, 7417, 7433, 7451,
7457, 7459, 7477, 7481, 7487, 7489, 7499, 7507, 7517, 7523, 7529, 7537, 7541,
7547, 7549, 7559, 7561, 7573, 7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639,
7643, 7649, 7669, 7673, 7681, 7687, 7691, 7699, 7703, 7717, 7723, 7727, 7741,
7753, 7757, 7759, 7789, 7793, 7817, 7823, 7829, 7841, 7853, 7867, 7873, 7877,
7879, 7883, 7901, 7907, 7919, 7927, 7933, 7937, 7949, 7951, 7963, 7993, 8009,
8011, 8017, 8039, 8053, 8059, 8069, 8081, 8087, 8089, 8093, 8101, 8111, 8117,
8123, 8147, 8161, 8167, 8171, 8179, 8191, 8209, 8219, 8221, 8231, 8233, 8237,
8243, 8263, 8269, 8273, 8287, 8291, 8293, 8297, 8311, 8317, 8329, 8353, 8363,
8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431, 8443, 8447, 8461, 8467, 8501,
8513, 8521, 8527, 8537, 8539, 8543, 8563, 8573, 8581, 8597, 8599, 8609, 8623,
8627, 8629, 8641, 8647, 8663, 8669, 8677, 8681, 8689, 8693, 8699, 8707, 8713,
8719, 8731, 8737, 8741, 8747, 8753, 8761, 8779, 8783, 8803, 8807, 8819, 8821,
8831, 8837, 8839, 8849, 8861, 8863, 8867, 8887, 8893, 8923, 8929, 8933, 8941,
8951, 8963, 8969, 8971, 8999, 9001, 9007, 9011, 9013, 9029, 9041, 9043, 9049,
9059, 9067, 9091, 9103, 9109, 9127, 9133, 9137, 9151, 9157, 9161, 9173, 9181,
9187, 9199, 9203, 9209, 9221, 9227, 9239, 9241, 9257, 9277, 9281, 9283, 9293,
9311, 9319, 9323, 9337, 9341, 9343, 9349, 9371, 9377, 9391, 9397, 9403, 9413,
9419, 9421, 9431, 9433, 9437, 9439, 9461, 9463, 9467, 9473, 9479, 9491, 9497,
9511, 9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623, 9629, 9631,
9643, 9649, 9661, 9677, 9679, 9689, 9697, 9719, 9721, 9733, 9739, 9743, 9749,
9767, 9769, 9781, 9787, 9791, 9803, 9811, 9817, 9829, 9833, 9839, 9851, 9857,
9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929, 9931, 9941, 9949, 9967, 9973]
```

[9]:
```python
def es_primo(n, primos):
    print("Analizando si ",n," es primo")
    primo=False

    if n<10000:
```

```python
        print("El posible primo que estudiamos es ",n)
        primo= n in primos
        print(primo)
        return primo
    else:
        print("---->Usamos Lucas_Lehmer para ver si ", n , " es primo")
        return Lucas_Lehmer(n)
```

[10]:
```python
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

    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
```

Como $r = n + 1$ es par, podemos aplicar el algoritmo de Polard:

[11]:
```
r=n+1
rho_de_polard(r,False)
```

[11]: 2

[12]:
```
r//2
```

[12]: 246762749168655646093593945367

Obtenemos así que el primer factor primo es 2, y por lo tanto $n = 2 * 246762749168655646093593945367$. A priori no sabemos si 246762749168655646093593945367 es primo o no, por lo que vamos a ver si pasa el test de Fermat del primer Tema:

[13]:
```
primos=[2,3,5,7,11]

for i in primos:
    ␣
 ↪a=exponenciacion_rapida_izda_dcha(i,246762749168655646093593945366,246762749168655646093593
    print(a)
```

```
318283050490257357770129059174
199662934664158145030868420630
77050022063174475368351194530
19620207212184836499245906782
33382069446477928051032821528
```

Como vemos no pasa el test con ninguna base, aunque solo nos bastaba con que fallara en un caso. Por lo tanto podemos aplicar el algoritmo de Polard:

[14]:
```
a=rho_de_polard(246762749168655646093593945367,False)

if a in primos_4_cifras:
    print(a," es primo")
else:
    print(a," no es primo")
```

```
61  es primo
```

[15]:
```
246762749168655646093593945367//61
```

[15]: 4045290969977961411370392547

Como vemos hemos encontrado otro factor primo de r por lo que tenemos que $r = 2 * 61 * 4045290969977961411370392547$. De nuevo comprobamos si 4045290969977961411370392547 es primo.

[16]:
```
for i in primos:
    ␣
 ↪a=exponenciacion_rapida_izda_dcha(i,4045290969977961411370392546,404529096997796141137039254
    print(a)
```

```
879611010007591772897027964
3841592401499695908496642886
1427952522187909585214237625
1538204315545510903638801805
3884190778209702393413375378
```

De nuevo podemos aplicar Polard, pues no pasa el test para ninguna base:

[17]:
```
a=rho_de_polard(4045290969977961411370392547,False)

if a in primos_4_cifras:
    print(a," es primo")

print(a)
```

```
50060591339659
```

[18]:
```
4045290969977961411370392547//50060591339659
```

[18]: 80807894228233

Como vemos hemos encontrado otro factor primo de r por lo que tenemos que \$r=2*61*50060591339659*80807894228233 \$. De nuevo comprobamos si 50060591339659 es primo.

[19]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,50060591339658,50060591339659)
    print(a)
```

```
1
1
1
1
1
```

Como vemos pasa el Test de Fermat para todas las bases, luego es un posible primo de Fermat. Vamos a comprobar la probabilidad de que sea primo con el algoritmo de Millner-Rabin.

[20]:
```
exp=obtener_exponentes_millner_rabin(50060591339659)
print(exp)
```

```
[50060591339658, 25030295669829]
```

[21]:
```
for j in primos:
    for i in reversed(exp):
```

```
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,50060591339659)
        print(resultado)
        print("")
```

```
Probamos con la base 2
50060591339658

Probamos con la base 2
1

Probamos con la base 3
50060591339658

Probamos con la base 3
1

Probamos con la base 5
1

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 para todas las bases, luego la probabilidad de que sea primo es de $1 - 4^{-5} = 0.9990234375$. Por ello vamos a intentar demostrar que es primo usando **Lucas-Lehmer**.

Por ello necesitamos Calcular los divisores primos de p-1 (siendo p=50060591339659).

[22]:
```
p=50060591339659
rho_de_polard(p-1)
```

[22]: 3

[23]:
```
50060591339659//3
```

[23]: 16686863779886

Luego tenemos que $p-1=3*16686863779886*. Claramente 16686863779886 es compuesto por ser par, luego aplicamos Polard de nuevo:

[24]:
```
rho_de_polard(16686863779886)
```

[24]: 2

[25]:
```
16686863779886//2
```

[25]: 8343431889943

Tenemos que $p-1 = 3*2*8343431889943$. Veamos si 8343431889943 es primo:

[26]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,8343431889942,8343431889943)
    print(a)
```

```
2252880263720
1852640627867
4034454017885
7743489965090
3762196906973
```

Obtenemos certificado de composición pues no pasa el test de Fermat.

[27]:
```
rho_de_polard(8343431889943)
```

[27]: 12637

[28]:
```
8343431889943//12637
```

[28]: 660238339

Tenemos que $p-1 = 3*2*12637*660238339$. Veamos si 12637 es primo:

[29]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,12636,12637)
    print(a)
```

```
1
1
1
1
1
```

Como vemos es un posible primo para las 5 primeras bases, por lo que no es recomendable aplicar el algoritmo ρ de Polard de primeras pues de ser primo no acabaría nunca. Vamos antes a calcular la probabilidad de que nuestro nuevo número sea primo usando **Millner-Rabin**.

[30]:
```python
exp=obtener_exponentes_millner_rabin(12637)
print(exp)
```

```
[12636, 6318, 3159]
```

[31]:
```python
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,12637)
        print(resultado)
        print("")
```

```
Probamos con la base 2
5554

Probamos con la base 2
12636

Probamos con la base 2
1

Probamos con la base 3
12636

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 5
7083

Probamos con la base 5
12636

Probamos con la base 5
1

Probamos con la base 7
12636

Probamos con la base 7
1

Probamos con la base 7
1
```

```
Probamos con la base 11
12636

Probamos con la base 11
1

Probamos con la base 11
1
```

Como vemos pasa el Test de Millner-Rabin para todas las bases, luego la probabilidad de que sea primo es de $1 - 4^{-5} = 0.9990234375$. Por ello vamos a intentar demostrar que es primo usando **Lucas-Lehmer**.

[32]:
```python
q=12637
rho_de_polard(q-1)
```

[32]: 3

[33]:
```python
12636//3
```

[33]: 4212

Luego $q - 1 = 3 * 4212$, y volvemos a aplicar ρ de Polard porque 4212 es compuesto al ser par.

[34]:
```python
rho_de_polard(4212)
```

[34]: 3

[35]:
```python
4212//3
```

[35]: 1404

Luego $q - 1 = 3^2 * 1404$, y volvemos a aplicar ρ de Polard porque 1404 es compuesto al ser par.

[36]:
```python
rho_de_polard(1404)
```

[36]: 3

[37]:
```python
1404//3
```

[37]: 468

Luego $q - 1 = 3^3 * 468$, y sabemos que $468 = 2^2 \cdot 3^2 \cdot 13$, luego $q - 1 = 2^2 * 3^5 * 13$.

Ya podemos aplicar Lucas Lhemer para 12637.

[38]:
```python
divisores=[2,3,13]
if Lucas_Lehmer(12637,divisores):
    print("Es primo")
```

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

Por lo tanto 12637 es primo, pues tenemos un certificado de primalidad.

Volviendo a $p - 1 = 3 * 2 * 12637 * 660238339$, nos quedaría ver si 660238339 es primo, para ello veamos si pasa el test de Fermat.

[39]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,660238338,660238339)
    print(a)
```

```
1
1
1
1
1
```

Como vemos pasa el test de Fermatpara todas las bases, luego es posible primo de Fermat para todas las bases probadas. Probamos **Milner-Rabin**.

[40]:
```
exp=obtener_exponentes_millner_rabin(660238339)
print(exp)
```

```
[660238338, 330119169]
```

[41]:
```
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,660238339)
        print(resultado)
        print("")
```

```
Probamos con la base 2
660238338

Probamos con la base 2
1

Probamos con la base 3
660238338

Probamos con la base 3
1

Probamos con la base 5
1

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
```

Pasa el Test, luego vamos a usar Lucas Lhemer para obtener un certificado de primalidad. Primero calculamos los divisores primos de z-1 ($z = 660238339$)

[42]:
```
z=660238339
rho_de_polard(z-1)
```

[42]: 3

[43]:
```
(z-1)//3
```

[43]: 220079446

Luego $z - 1 = 3 * 220079446$, y volvemos a aplicar ρ de Polard porque 220079446 es compuesto al ser par.

[44]:
```
rho_de_polard(220079446)
```

[44]: 2

[45]:
```
220079446//2
```

[45]: 110039723

Luego $z - 1 = 2 * 3 * 110039723$,veamos si 110039723 es primo .

[46]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,110039722,110039723)
    print(a)
```

```
99738845
11193526
3112996
78221205
34700494
```

No pasa el TPF, luego aplicamos Polard:

[47]:
```
a=rho_de_polard(110039723)

if a in primos_4_cifras:
    print(a, "es primo")
```

```
523 es primo
```

[48]:
```
110039723//523
```

[48]: 210401

Luego $z - 1 = 2 * 3 * 523 * 210401$,veamos si 210401 es primo .

[49]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,210400,210401)
    print(a)
```

```
1
1
1
1
1
```

Es un posible primo de Fermat para toda base probada. Pasamos a **Millner-Rabin**.

[50]:
```
exp=obtener_exponentes_millner_rabin(210401)
print(exp)
```

```
[210400, 105200, 52600, 26300, 13150, 6575]
```

[51]:
```
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,210401)
        print(resultado)
        print("")
```

```
Probamos con la base 2
26486

Probamos con la base 2
31262

Probamos con la base 2
210400

Probamos con la base 2
1
```

```
Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 3
202218

Probamos con la base 3
53971

Probamos con la base 3
77397

Probamos con la base 3
179139

Probamos con la base 3
210400

Probamos con la base 3
1

Probamos con la base 5
174618

Probamos con la base 5
133004

Probamos con la base 5
179139

Probamos con la base 5
210400

Probamos con la base 5
1

Probamos con la base 5
1

Probamos con la base 7
183915

Probamos con la base 7
31262
```

```
Probamos con la base 7
210400

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 11
179139

Probamos con la base 11
210400

Probamos con la base 11
1

Probamos con la base 11
1

Probamos con la base 11
1

Probamos con la base 11
1
```

Pasa el Test de **Milner-Rabin** luego vamos a buscar un certificado de primalidad.

[52]:
```
x=210401
rho_de_polard(x-1)
```

[52]: 32

[53]:
```
210400//32
```

[53]: 6575

$x - 1 = 2^5 * 6575 = 2^5 * 5^2 * 263$, luego ya tenemos la descomposición en primos y podemos aplicar Lucas Lhemer.

[54]:
```
divisores=[2,5,263]
if Lucas_Lehmer(210401,divisores):
    print("Es primo")
```

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

Obtenemos un certificado de primalidad, y por lo tanto Luego $z - 1 = 2 * 3 * 523 * 210401$ es la descomposición en primos de $z - 1$ y podemos aplicar Lucas Lhemer a $z = 660238339$.

[55]:
```
divisores=[2,3,523,210401]
if Lucas_Lehmer(660238339,divisores):
    print("Es primo")
```

```
El natural más pequeño cuya clase es primitiva:  3
Es primo
```

Luego obtenemos un certificado de primalidad de 660238339 y $p-1 = 3*2*12637*660238339$ sería una descomposición en factores primos de $p - 1$, luego podemos aplicar de nuevo Lucas Lhemer para obtener el certificado de primalidad de p.

[56]:
```
divisores=[2,3,12637,660238339]
if Lucas_Lehmer(50060591339659,divisores):
    print("Es primo")
```

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

Obtenemos así un certificado de primalidad de 50060591339659.

Volviendo a la descomposición de factores primos de $r = n + 1 = 2 * 61 * 50060591339659 * 80807894228233$ nos faltaría por ver si 80807894228233 es primo.

En primer lugar veamos si pasa el test de Fermat:

[57]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,80807894228232,80807894228233)
    print(a)
```

```
1
1
1
1
1
```

Como pasa el Test, Veamos con **Milner-Rabin**:

[58]:
```
exp=obtener_exponentes_millner_rabin(80807894228233)
print(exp)
```

```
[80807894228232, 40403947114116, 20201973557058, 10100986778529]
```

[59]:
```
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,80807894228233)
```

```python
        print(resultado)
        print("")
```

```
Probamos con la base 2
2334228425899

Probamos con la base 2
80807894228232

Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 3
78473665802334

Probamos con la base 3
80807894228232

Probamos con la base 3
1

Probamos con la base 3
1

Probamos con la base 5
5352059533074

Probamos con la base 5
78473665802334

Probamos con la base 5
80807894228232

Probamos con la base 5
1

Probamos con la base 7
80807894228232

Probamos con la base 7
1

Probamos con la base 7
1
```

```
Probamos con la base 7
1

Probamos con la base 11
31760473952997

Probamos con la base 11
2334228425899

Probamos con la base 11
80807894228232

Probamos con la base 11
1
```

Como vemos pasa el test y vamos a buscar un certificado de primalidad con Lucas Lhemer. Para ello buscamos la descomposición en factores primos de $y-1$ (con y=80807894228233)

[60]:
```
y=80807894228233
rho_de_polard(y-1)
```

[60]: 3

[61]:
```
(y-1)//3
```

[61]: 26935964742744

Luego tenemos que $y - 1 = 3 * 26935964742744$. Como 26935964742744 es compuesto aplicamos Polard de nuevo:

[62]:
```
rho_de_polard(26935964742744)
```

[62]: 3

[63]:
```
26935964742744//3
```

[63]: 8978654914248

Obtenemos así que $y-1 = 3^2 * 8978654914248$ como 8978654914248 es compuesto aplicamos Polard.

[64]:
```
rho_de_polard(8978654914248)
```

[64]: 3

[65]:
```
8978654914248//3
```

[65]: 2992884971416

$y - 1 = 3^3 * 2992884971416$, repetimos proceso con 2992884971416.

[66]:
```python
rho_de_polard(2992884971416)
```

[66]: 8

[67]:
```python
2992884971416//8
```

[67]: 374110621427

Obtenemos que $y-1 = 3^3 * 2^3 * 374110621427$, luego debemos comprobar si 374110621427 es primo, para ello vemos si pasa el TPF.

[68]:
```python
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,374110621426,374110621427)
    print(a)
```

```
54662346357
137713362646
207353360066
6310924808
71600240229
```

Como vemos no pasa el Test para ninguna base, luego podemos aplicar Polard.

[69]:
```python
rho_de_polard(374110621427)
```

[69]: 61

[70]:
```python
374110621427//61
```

[70]: 6132961007

Luego $y - 1 = 3^3 * 2^3 * 61 * 6132961007$. Veamos si 6132961007 es primo:

[71]:
```python
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,6132961006,6132961007)
    print(a)
```

```
4208046809
453173136
4787067676
2903667965
3713961829
```

No pasa el test de Fermat, aplicamos Polard:

[72]:
```python
rho_de_polard(6132961007)
```

[72]: 2593

[73]:
```
6132961007//2593
```

[73]: 2365199

Luego $y-1 = 3^3 * 2^3 * 61 * 2593 * 2365199$. Como 2593 es primo, veamos si lo es 2365199.

Como vemos pasa tanto TPF como Milner-Rabin, por lo que es muy probable que sea primo. Vamos a buscar un certificado con Lucas Lhemer, para ello primero descomponemos en factores primos $w-1 = 2592$

[74]:
```
for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,2365198,2365199)
    print(a)
```

```
1783686
2162292
1981592
160632
1322579
```

Como vemos es compuesto, pues no pasa el TPF, aplicamos Polard:

[75]:
```
rho_de_polard(2365199)
```

[75]: 619

[76]:
```
2365199//619
```

[76]: 3821

Así $y-1 = 3^3 * 2^3 * 61 * 619 * 2593 * 3821$,obtenemos así la descomposición en factores primos de $y-1$ y podemos aplicar Lucas Lhemer para ver si $y = 80807894228233$ es primo:

[77]:
```
divisores=[2,3,61, 619, 2593, 3821]
if Lucas_Lehmer(80807894228233,divisores):
    print("Es primo")
```

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

Como vemos obtenemos un certificado de primalidad para 80807894228233, y por lo tanto tenemos la descomposición en primos de $r = n+1 = 2 * 61 * 50060591339659 * 80807894228233$.

Calculamos a continuación los $U_{r/p}$:

[78]:
```
divisores=[1,2,61,50060591339659,80807894228233]
r=n+1
P=1
Q=-1
```

```
for i in divisores:
    un,a,v=sLucas_modificado(P,Q,r//i,n)
    print("Sucesion U_r/",i,":",un)
    print()
```

```
Sucesion U_r/ 1 : 0

Sucesion U_r/ 2 : 0

Sucesion U_r/ 61 : 103019351701660670879035336783

Sucesion U_r/ 50060591339659 : 374431693794292349194973673147

Sucesion U_r/ 80807894228233 : 97474365767954366766451485440
```

Como podemos ver el rango de Lucas $w(n)$ en este caso sería menor estricto que r y menor o igual que $r/2$ pues $U_{r/2}$ da 0 módulo n pero no podemos asegurar que no haya otro índice menor que $r/2$ en el que U_i se anule, por lo que no podemos afirmar que nuestro n sea primo, pues para los valores de P y Q utilizados no hemos obtenido un rango de lucas de $n + 1$.