Ejercicio10

March 27, 2022

1 Ejercicio 10

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

[1]:
```
p=26505161

from math import gcd
import numpy as np
import sys

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



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
```

i) Calcula el símbolo de Jacobi($\frac{-11}{p}$). Si sale 1, usa el algoritmo de Tonelli-Shanks para hallar soluciones a la congruencia $x^2 \equiv -11 \; mod \; p$.

[2]:
```
jacobi_symbol(-11,p)
```

[2]: 1

Vamos a comprobar ahora si nuestro p es primo:

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

for i in primos:
    a=exponenciacion_rapida_izda_dcha(i,p-1,p)
    print(a)
```

```
1
1
1
1
1
```

Como vemos es posible primo de Fermat para todas las bases que hemos probado, vamos a aplicarle el método de Milner-Rabin:

[4]:
```python
exp=obtener_exponentes_millner_rabin(p)
print(exp)
```

```
[26505160, 13252580, 6626290, 3313145]
```

[5]:
```python
for j in primos:
    for i in reversed(exp):
        print("Probamos con la base",j)
        resultado=exponenciacion_rapida_izda_dcha(j,i,p)
        print(resultado)
        print("")
```

```
Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 2
1

Probamos con la base 3
5674162

Probamos con la base 3
3778773

Probamos con la base 3
26505160

Probamos con la base 3
```

```
1

Probamos con la base 5
26505160

Probamos con la base 5
1

Probamos con la base 5
1

Probamos con la base 5
1

Probamos con la base 7
3778773

Probamos con la base 7
26505160

Probamos con la base 7
1

Probamos con la base 7
1

Probamos con la base 11
22726388

Probamos con la base 11
26505160

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.

Para ello necesitamos encontrar los divisores primos de $p-1$, como es un proceso que ya hemos repetido en prácticas anteriores, vamos a utilizar una función de Sage para calcularlos directamente:

[6]:
```
prime_factors(p-1)
```

[6]: [2, 5, 11, 59, 1021]

Una vez calculados vamos a aplicar el algoritmo de Lucas-Lhemer:

[7]:
```
divisores=[2, 5, 11, 59, 1021]
if Lucas_Lehmer(p,divisores):
    print("Es primo")
```

```
El natural más pequeño cuya clase es primitiva:  13
Es primo
```

Obtenemos así un certificado de primalidad para nuestro p.

Vamos a calcular $p \ mod \ 8$

[8]:
```
p%8
```

[8]: 1

Como hemos obtenido 1, debemos aplicar el algoritmo de Tonelli-Shanks para calcular las soluciones de $x^2 \equiv -11 \ mod \ p$.

Como $p - 1 = 2^3 \cdot 3313145 = 2^e \cdot q$ sabemos que como mucho el algoritmo tiene $e = 3$ pasos y $q = 3313145$.

Para iniciar el algoritmo necesitamos un no residuo cuadrático módulo p.

[9]:
```
jacobi_symbol(3,p)
```

[9]: -1

[10]:
```
z=mod((3**3313145),p)
z
```

[10]: 5674162

El primero que se encuentra es n=3 ya que $(p/3) = -1$.

Entonces un generador del 2-subgrupo de Sylow $G \cong \mathbb{Z}_{2^3} = \mathbb{Z}_8$, es $n^q \equiv 5674162 \ mod \ p$.

Calculamos $(-11)^q \ mod \ p$:

[11]:
```
t=mod((-11)**3313145,p)
t
```

[11]: 3778773

A continuación iteramos para conseguir las soluciones:

[13]:
```
from algoritmos import *

tonelli_shanks(-11,p)
```

```
e = 3
q = 3313145
t mod p = 3778773
z mod p = 5674162
r mod p = 13486457

Inicio de bucle:
        Orden de t = 4 => i = 2
        b mod p = 5674162
        b^2 mod p = 3778773
        r_1 mod p = 19253206
        t_1 mod p = 26505160

Inicio de bucle:
        Orden de t = 2 => i = 1
        b mod p = 3778773
        b^2 mod p = 26505160
        r_1 mod p = 8670558
        t_1 mod p = 1
```

[13]:
```
[8670558, 17834603]
```

Obtenemos así que las soluciones serían $x_1 = 8670558,\ x_2 = 17834603$.

ii) Usa una de esas soluciones para factorizar el ideal principal, $(p) = (p, n+\sqrt{-11})(p, n-\sqrt{-11})$ **como producto de dos ideales.**

Tomamos la solución impar anterior $n = x_2 = 17834603$ y como p es primo e impar y no divide a -11 tendríamos que $(p) = (p, 17834603 + \sqrt{-11})(p, 17834603 - \sqrt{-11})$

iii) Aplica el algoritmo de Cornachia-Smith modificado a 2p y n para encontrar una solucíón a la ecuación diofántica $4p = x^2 + 11y^2$ **y la usas para encontrar una factorización de p en a.e. del cuerpo** $\mathbb{Q}[p]$**.**

El código para alplicar el algoritmo de Cornachia-Smith es el siguiente:

[14]:
```python
def cornacchia_smith_modificado(p,n):
    siguiente_Div = 2*p
    siguiente_div = n

    resto = 2*p
    i = 1

    while not resto < 2*sqrt(p):
        cociente, resto = divmod(siguiente_Div, siguiente_div)
        print("[" + str(i) + "] " + str(siguiente_Div) + " = " + str(cociente)+␣
↪"*" + str(siguiente_div) + " + " + str(resto))
        siguiente_Div = siguiente_div
        siguiente_div = resto
        i = i + 1
```

```
    return resto
```

[15]:
```
cornacchia_smith_modificado(p, 17834603)
```

```
[1] 53010322 = 2*17834603 + 17341116
[2] 17834603 = 1*17341116 + 493487
[3] 17341116 = 35*493487 + 69071
[4] 493487 = 7*69071 + 9990
```

[15]: 9990

[16]:
```
float(2*sqrt(p))
```

[16]: 10296.632653445495

[17]:
```
y=int(sqrt((4*p-(9990**2))/11))
y
```

[17]: 752

Comprobamos que los resultados obtenidos son correctos:

[18]:
```
4*p==9990^2+11*752**2
```

[18]: True

Por lo tanto concluimos que con el algoritmo de Cornacchia Smith modificado hemos conseguido después de 5 divisiones el primer resto $r = 9990$ menor que $2\sqrt{p} \simeq 10296.632653445495$.

Dicho resto nos da la factorización:

$4p = 9990^2 + 11 \cdot 752^2$ y entonces $p = (4995 + 376\sqrt{-11})(4995 - 376\sqrt{-11})$

iv) ¿ Son principales tus ideales $(p, n + \sqrt{-11})$ **y** $(p, n - \sqrt{-11})$ **?**

Sí, serían principales porque en la factorización del apartado anterior hemos encontrado un generador de cada uno de ellos.