GIIT_AMA_1819_T1_EJERCICIOS

Práctica de Ampliación de Matemáticas (Tema 1. Teoría de números)

Alumnos que integran el grupo

1. **Alfonso Nguema Ela Nanguan**
2. **Carlos Rebolledo Aliseda**
3. **David Pozo Nuñez**

A. Realizar los ejercicios del 1-5:

Ejercicios

1. Encontrar un número de modo que el Sage tarde en factorizarlo más de 2 segundos (en el desplegable de arriba tiene acción->interrumpir por si el
2. Convertir el algoritmo de Euclides en una función que reciba dos números enteros y devuelva el máximo común divisor.
3. Convertir el algoritmo de Euclides en una función recursiva que reciba dos números enteros y devuelva el máximo común divisor.
4. Calcular los números correspondientes a $ax+by$, con $a=24$, $b=14$ y x,y entre -10 y 10. ¿Cuál podría ser su máximo común divisor a ra
5. Crear una función que devuelva los números x,y dados por la Identidad de Bezout.

A.

Ejercicio 1.

```
%timeit
# Factorizar
list(factor(342343232948732423423894723242342343243243243211234124314351243512452125141511245 5))
```

```
    [(5, 1),
     (7, 1),
     (239, 1),
     (212418757011892433, 1),
     (19266504983838446836394567259002399191335508754348001057580 99, 1)]
    CPU time: 2.62 s,  Wall time: 2.62 s
```

Ejercicio 2.

```
x=6
y=3
```

```
def euclides(x,y):
    resto=x%y
    while resto!=0:
        x=y
        y=resto
        resto=x%y
    return y
```

```
euclides(x,y)
```

```
    3
```

Ejercicio 3.

```
x = 10
y = 15
def euclides_rec(x,y):
    resto=x%y
    if resto==0:
        return y
    else:
        x=y
        y=resto
    return euclides_rec(x,y)
```

```
euclides_rec(x,y)
```

```
5
```

Ejercicio 4.

```
# Tomamos dos números como ejemplo
n=24
m=14
# Guardamos una copia
n1=n
m1=m
```

```
# Calculamos la secuencia de cocientes y restos
resto=n%m
cocientes=[]
while resto!=0:
    cocientes=[n//m]+cocientes
    n=m
    m=resto
    resto=n%m
m
```

```
2
```

```
# Calculamos x e y a partir de los cocientes:
x=1
y=-cocientes[0]
for c in cocientes[1:]:
    x,y=y,x-c*y
print x,y
```

```
3 -5
```

Ejercicio 5.

```
#Ejercicio 5
var('a,b')
def bezout(a,b):
    r=a%b
    cocientesb=[]
    while r!=0:
        cocientesb=[a//b]+cocientesb
        a=b
        b=r
        r=a%b
    xb=1
    yb=-cocientesb[0]
    for c in cocientesb[1:]:
        xb,yb=yb,xb-c*yb
    return xb,yb
```

```
bezout(312,120)
```

```
(2, -5)
```

En los siguientes ejercicios se tratará de codificar funciones sencillas de teoría de números, para repasar los conceptos y algoritmos. Sage tiene predefinidas montones de funciones que hacen todas esas operaciones, pero el objetivo es que vosotros aprendáis cómo se trabaja con números enteros, por lo que en la resolución deberéis usar únicamente lo que se ha explicado en las prácticas (o las que aparecen al final de la práctica). Las funciones tienen que estar convenientemente explicadas, con un comentario al comienzo de la función como líneas de comentario en cada operación que se considere compleja (en particular en los bucles y llamadas recursivas).

B. Programar el apartado 1 y hacer uno de los ejercicios del 2 al 4.

1. Potenciación modular. Crea una función que reciba tres números, b,m,n y calcule el resto de dividir b^m entre n. En lugar de utilizar la función ϕ (para la que se necesita una factorización) usad el siguiente método recursivo: si m=1, devolverá el resto de dividir b entre n. Si m es par, calculará $B = b^{m/2}$ módulo n y devolverá el resto de dividir B^2 entre n. Si m es impar, calculará $B = b^{(m-1)/2}$ módulo n y devolverá el resto de dividir bB^2 entre n.
2. Decimos que un número m es pseudoprimo base b si $b^{m-1} \equiv_m 1$. Elige un número primo b menor que 20 (y mayor o igual a 3) y calcula un pseudoprimo (no primo) base b.
3. Los números de Carmichael son aquellos números n que verifican que $a^{n-1} \equiv_n 1$ para todo a tal que $mcd(a, n) = 1$. Encuentra un número de Carmichael que sea múltimo de tres primos, siendo el menor de ellos 7.
4. Un número n es de Poulet si $2^{n-1} \equiv_n 1$. Encontrar todos los números de Poulet menores que 1000 y determinar cuáles no son primos.

C. Algoritmo RSA

Comencemos con una introducción al problema:

El algoritmo RSA funciona del siguiente modo:

Partimos de dos números primos, p y q. Tomamos $n = pq$ como nuestro tamaño de palabra, es decir, tenemos que codificar el mensaje en un número en base n.

Supongamos que ya hemos hecho este proceso y que m es una palabra de nuestro mensaje (debería modificarse para ser prima con n, pero vamos a obviar este paso).

Además, necesitamos un número e coprimo con $\phi(n)$ y d, su inverso módulo $\phi(n)$ ($ed \equiv_{\phi(n)} 1$).

Para codificar nuestro mensaje, basta hacer $m_c = m^e$. Y m_c será el mensaje codificado.

Para decodificarlo, hacemos $m_c^d \equiv_n m^{de} \equiv_n m$.

La clave pública estará formada por n y e.

La clave privada será d.

Ejercicios:

a. Tomando p=4519602136809738190458662235449273316 1, q=173905113088091315885542665902416112387 y e=367, crea una función que reciba una cadena de texto y la devuelva como un número encriptado con RSA y otra que reciba el número encriptado y devuelva la cadena original.
b. Crea la clave pública y privada (usa el procedimiento que aparece en Wikipedia).

MISCELÁNEA

Las siguientes funciones son para simplificar los ejercicios del algoritmo RSA.

B.

1.

```
#Ejercicio 1 para calcular b^m modulo n
def funcion(b,m,n):
    if m==1:
        return b%n
    if m%2==0:
        B=funcion(b,m/2,n)
        return B^2%n
    if m%2==1:
        B=funcion(b,(m-1)/2,n)
        return b*B^2%n
```

```
funcion(3,4,6)
```

```
3
```

2.

```
resultado = 0
y = 1
while resultado != 1 or is_prime(y) == true:
    y = y+1
    x = y-1
    resultado = funcion(5, x, y)
print(y)
```

```
4
```

```
# Comprobamos que la solucion sea m = 4
funcion(5, y-1, y)
```

```
1
```

3.

```
#  creamos una función que nos diga si  es   un número carmichael o no
def carmichael(n):

    for a in [1..n-1]:
        if gcd(a,n)==1:
           if funcion(a,n-1,n)!=1:
             return false
    return true
```

```
[n for n in [7,14..7000] if carmichael(n)]
```

```
[7, 1729, 2821, 6601]
```

```
# estos tres valen
factor(1729),factor(2821),factor(6601)
```

```
(7 * 13 * 19, 7 * 13 * 31, 7 * 23 * 41)
```

C. Algoritmo RSA.

```
p=45196021368097381904586622354492733161
q=17390511308809131588554266590241611238 7
e=367
phi_n1=(p-1)*(q-1)
```

```
# Tomamos dos números como ejemplo
a=phi_n1
b=e
# Guardamos una copia
a0=a
b0=b
```

```
# Calculamos la secuencia de cocientes y restos
r=a%b
cocientes=[]
while r!=0:
    cocientes=[a//b]+cocientes
    a=b
    b=r
    r=a%b
b
```

```
1
```

```
# Calculamos x e y a partir de los cocientes:
x=1
y=-cocientes[0]
for c in cocientes[1:]:
    x,y=y,x-c*y
print x,y
```

```
-32
685324835500884297785555919290780979802310264759139328450295421849361867663
```

```
msg_numero=base10(texto_a_numero(u'¡Hola Mundo!'),255)
msg_numero
```

```
99121346785038495866901854 21
```

```
# Para codificar, elevamos a e
msg_codificado=funcion(msg_numero,e,p*q)
msg_codificado
```

```
95842663289200705633373876487636614893023774018901671401199894995809
7034345
```

```
d=685324835500884297785555919290780979802310264759139328450295421849361867663
```

```
# Para descodificar, elevamos a d (que habéis calculado antes con el algoritmo de Euclides)
msg_descodificado=funcion(msg_codificado,d,p*q)
print numero_a_texto(msg_descodificado.digits(255))
```

```
¡Hola Mundo!
```

```
%auto
def texto_a_numero(texto):
    # Recibe un texto y devuelve la lista de valores ASCII de sus caracteres.
    return [ord(k) for k in texto]
def numero_a_texto(numero):
    # Recibe una lista de valores ASCII y devuelve un texto
    texto=''
    for k in numero:
        texto+=unichr(k)
    return texto
```

```
%auto
def base10(cifras,base):
    n=len(cifras) # Obtenemos el número de cifras
    numero=0 # Aquí guardaremos el número
    for k in [1..n]: # Recorremos las cifras (están ordenadas de menor a mayor potencia.
        numero+=cifras[k-1]*base^(k-1) # Usamos la expresión del número en una base
    return numero
```

```
# Pasamos un mensaje a texto:
msg_numero=base10(texto_a_numero(u'¡Hola Mundo!'),255)
msg_numero
```

```
99121346785038495866901854 21
```

```
# Y recuperamos el mensaje:
print numero_a_texto(msg_numero.digits(255))
```

```
¡Hola Mundo!
```

```
# Para generar números aleatorios muy grandes
import random as rnd
Z = IntegerRing()
p=Z(rnd.getrandbits(128))
p
```

```
40469289386228221094370757167034237725
```

```
# Y para comprobar si el número generado es primo:
p.is_prime()
```

```
False
```

Planificación y trabajo en equipo

Trabajo en equipo

Cada miembro del equipo, deberá subir una memoria que contenga, al menos, los siguientes items:

- Metodología seguida. Indicar cómo se ha organizado el trabajo, en qué tareas se ha dividido y a quién han sido asignadas.
- Calendario de trabajo. Indicar las reuniones que se vayan a programar y cómo se realizará la coordinación entre los miembros del equipo.
- Incidencias.

Planificación

Cada alumno debe subir una memoria con los siguientes puntos:

- Planificación inicial. Una vez que se haya producido el reparto de tareas del equipo, deberá elaborar una planificación donde figurarán las distintas tareas que debe realizar, un tiempo estimado para cada una y el periodo de tiempo en el que lo realizará.
- Tiempo total empleado y tiempo empleado en cada tarea.

Normas de entrega

1. Rellenar en la parte superior el nombre de los integrantes del grupo.
2. Compartir esta hoja de Sage con el profesor (etlopez18).
3. En el desplegable "File" de la parte superior de la página, elegir "Print"
4. Imprimir la página resultante como pdf.
5. Subirla al campus virtual antes de la fecha límite, junto con la memoria de planificación y trabajo en equipo. Cada día a partir de la fecha límite, la nota sobre la que se puntúa el ejercicio bajará en un punto.

Criterios de evaluación

- 80% de la calificación que los resultados y métodos aplicados sean correctos.
- 20% de la calificación que los resultados estén correctamente explicados.

PARA APROBAR LAS PRÁCTICAS SERÁ NECESARIO HACER AL MENOS LOS EJERCICIOS DE LA PARTE A).