ALEJANDRO_SANTORUM-labot-ex3-2016(EJERCICIO)

March 4, 2018

Por favor, antes de empezar el examen cambia el nombre de la hoja (File>Rename worksheet) poniendo en lugar de "nombre.apellido" los tuyos tal como aparecen en tu dirección de correo electrónico de la UAM. El final del nombre de la hoja, -labot-ex3-2016, dejálo como está.

Una vez hayas terminado el examen, salva la hoja (File>Save worksheet to a file....) y déjala en la carpeta en tu escritorio que tiene nombre "ENTREGA.....".

Ejercicio 1

(4 puntos) En la carpeta que contiene este archivo se encuentra el el mismo *PALABRAS2* que se usó en la hoja 84-CRIPT-matricial-fuerzabruta para romper un sistema criptográfico mediante fuerza bruta. El archivo contiene una lista de palabras, una en cada línea, en inglés. Para hacer este ejercicio debes definir y ejecutar funciones para responder a la siguiente pregunta: £Cuántas tripletas de letras tienen la propiedad de aparecer al principio (es decir, como tripleta inicial) de todas las palabras de la lista PALABRAS2 en que aparecen como tripleta de letras consecutivas en cualquier posición? No consideramos palabras en la lista de longitud cuatro o menor.

Por ejemplo, en la lista aparece la palabra GEOMETRY y la tripleta MET no es una de las que podemos contar, es decir no contribuye a nuestro contador, porque aparece al menos una vez en una situación que no es inicio de palabra.

Para poder comprobar que el programa funciona bien, modificarlo para que cada vez que se incremente el contador se añada la clave a una lista, y luego efectuar varias comprobaciones convincentes (£A quén hay que convencer? £A quén va a ser? Pues a mí).

```
In [1]: def diccionario(L):
            dicc = {}
            infile = open("PALABRAS2.txt","r")
            for palabra in infile.readlines():
                if len(palabra[:-1])<5:
                    '''Despreciamos palabras de 4 o menos caracteres.'''
                    continue
                elif dicc.has_key(palabra[:3]):
                   dicc[palabra[:3]].append(palabra[:-1])
                else:
                    dicc[palabra[:3]]=[palabra[:-1]]
            for palabra in L:
                '''Este bucle es para añadir al diccionario unas pocas
                    palabras que podrían faltar. NO es importante'''
                dicc[palabra]=[palabra]
            infile.close()
            return dicc
```

```
In [7]: dicc=diccionario([]) #obtenemos el diccionario

In [9]: keys = dicc.keys() #obtenemos las claves (que son las tripletas a comprobar)

In [11]: values = dicc.values() # obtenemos los valores.
                                # para cada clave, hay una lista de palabras que empiezan por

In [45]: # Función que dada una tripleta, determina si está repetida
         # en otra parte de la palabra que no sea la inicial
         # devuelve False si está la tripleta presente en otra parte de la palabra
         # devuelve True de lo contrario.
         def tripletaValida(tri, palabra):
             i = 1
             l = len(palabra)
             while(1):
                 j = i+3
                 if(j > l):
                     return True
                 if palabra[i:j] == tri:
                     return False
                 else:
                     i += 1
                     continue

In [63]: # Cada clave (tripleta inicial), es evaluada con la funcion anterior
         # con cada palabra del diccionario.
         def contador(keys, values):
             cont = 0
             L = list()
             for key in keys:
                 flag = 1
                 for value in values:
                     for unit in value:
                         if tripletaValida(key, unit)==False:
                             flag = 0
                             break

                     if flag == 0:
                         break

                 if flag != 0:
                     L.append(key)
                     cont += 1

             return cont, L

In [73]: %time cont, L = contador(keys, values)
         print ("Número de tripletas que cumplen lo especificado: "+str(cont))
         print L
```

```
CPU times: user 1min 19s, sys: 124 ms, total: 1min 19s
Wall time: 1min 41s
Número de tripletas que cumplen lo especificado: 292
['NYQ', 'KYM', 'KYT', 'BYZ', 'BYW', 'BYP', 'BYG', 'MAO', 'BEJ', 'BEW', 'VLA', 'VLT', 'OXP', 'F
```

```
In [75]: def buscadorContraejemplo(keys, values):
             cont = 0
             L = list()
             for key in keys:
                 flag = 1
                 for value in values:
                     for unit in value:
                         if tripletaValida(key, unit)==False:
                             flag = 0
                             L.append([key, unit])
                             cont += 1
                             break
                     if flag == 0:
                         break
             return cont, L
```

Vamos a utilizar la función anterior para buscar un contraejemplo para cada tripleta que no cumpla lo especificado

```
In [76]: %time contContra, LC = buscadorContraejemplo(keys, values)
         print ("Número de tripletas que no cumplen lo especificado: "+str(contContra))
         print LC
```

```
CPU times: user 1min 15s, sys: 268 ms, total: 1min 15s
Wall time: 1min 15s
Número de tripletas que no cumplen lo especificado: 3133
[['AGO', 'OCTAGON'], ['AGN', 'SPHAGNOUS'], ['AGM', 'NYSTAGMIC'], ['AGL', 'MAGLEMOSIAN'], ['OCR
```

```
In [83]: print("El número de keys es igual a la suma de las tripletas que lo cumplen más las qu
         print("Número de tripletas que lo cumplen: "+str(cont)+"\n")
         print("Número de tripletas que no lo cumplen: "+str(contContra)+"\n")
         print("Suma = "+str(cont+contContra)+"\n")
         print("Número de tripletas totales: "+str(len(keys))+"\n")
         print(len(keys) == (cont + contContra))
```

```
El número de keys es igual a la suma de las tripletas que lo cumplen más las que no?:

Número de tripletas que lo cumplen: 292

Número de tripletas que no lo cumplen: 3133

Suma = 3425
```

```
Número de tripletas totales: 3425
```

```
True
```

Esto creo que es suficiente para convencerte.

Ejercicio 2

(3 puntos) En este ejercicio hay que programar funciones para encriptar y desencriptar (no se excluye que la función que sirve para encriptar también sirva para desencriptar) un texto usando el siguiente sistema clásico: La clave es un entero d y una permutación π del conjunto $X_d := \{1, 2, \ldots, d\}$ (i.e. una función biyectiva de X_d en X_d). Encriptamos dividiendo el texto en bloques de longitud d y encriptando cada bloque colocando la letra que ocupa la posición i en el lugar $\pi(i)$. Debes comprobar que funciona utilizando el texto suministrado. Observa que la longitud del texto debe ser un múltiplo de d, y si no lo es debemos poner al final basura hasta que la longitud sea un múltiplo de d.

En este sistema las frecuencias de las letras en el texto original y en el encriptado son iguales, de forma que no se puede atacar en la misma forma que la cifra de César o la cifra de permutación. £Qué opinas de la seguridad de este método?

```
In [30]: texto = 'THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCO\
         UNTINGCALCULATIONMEASUREMENTANDTHESYSTEMATICSTUDYOFTHESHAPESANDMOTIONSOF\
         PHYSICALOBJECTSPRACTICALMATHEMATICSHASBEENAHUMANACTIVITYFORASFARBACKASWR\
         ITTENRECORDSEXISTRIGOROUSARGUMENTSFIRSTAPPEAREDINGREEKMATHEMATICSMOSTNOT\
         ABLYINEUCLIDSELEMENTSMATHEMATICSDEVELOPEDATARELATIVELYSLOWPACEUNTILTHERE\
         NAISSANCEWHENMATHEMATICALINNOVATIONSINTERACTINGWITHNEWSCIENTIFICDISCOVER\
         IESLEDTOARAPIDINCREASEINTHERATEOFMATHEMATICALDISCOVERYTHATCONTINUESTOTHE\
         PRESENTDAYZ'

In [31]: alfb = "ABCDEFGHIJKLMNOPQRSTUVWXYZ." # el punto es la basura introducida

In [32]: L_alfb = list(alfb)
         print(len(L_alfb))
```

```
27
```

```
In [33]: def ord2(c):
             return L_alfb.index(c)

In [34]: def chr2(n):
             return L_alfb[n]

In [35]: l = len(texto)
         l
```

```
Out[35]: 514
```

```
In [36]: d = randint(5,27)
         d
```

```
Out[36]: 6
```

```
In [37]: def permutacion(d): # crea una permutacion de {1, ..., d} elementos
             L = []
             while(len(L)<d):
                 a = randint(0, d)
                 if a not in L:
                     L.append(a)
             return L

         def esPermutacionCorrecta(L, d): # comprueba que la permutacion creada arriba es corr
             for i in xsrange(0, d):
                 if i not in L:
                     return False
             return True
```

```
In [38]: PERM = permutacion(d)
         print esPermutacionCorrecta(PERM, d)
         print PERM

True
[4, 3, 2, 5, 1, 0]
```

```
In [39]: def prepararTexto(texto, d):
             l = len(texto)
             L = list(texto)

             while(len(L)%d != 0):
                 L.append(".")

             textoPreparado = "".join(L)
             return textoPreparado
```

```
In [40]: texto2 = prepararTexto(texto, d)
         print len(texto2)
         print texto2

516
THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCOUNTINGCALCULATIONMEASURE
```

```
In [43]: def encriptar(texto, d, perm):
             L = list(texto)
             M = list()
             for i in xsrange(0, len(L)):
                 M.append("a")

             for i in xsrange (0, len(L)):
```

```
                indice = i%d
                bloque = floor(i/d)
                nuevo = perm[indice]
                M[(bloque*d) + nuevo] = L[i]
            cad = "".join(M)
            return cad
```

```
In [44]: encrip = encriptar(texto2, d, PERM)
         print encrip
```

```
GURHTOSUHTHESBFOEAITARTCLDANONLAIGOCNOAERSTAGNIMITMEHAEVDSCEFDPOLEUOMORCCGITNNALCLAUEMOITNMEUSA
```

```
In [45]: def desencriptar(texto, d, PERM):
             L = list(texto)
             M = list()
             for i in xsrange(0, len(L)):
                 M.append("a")

             for i in xsrange (0, len(L)):
                 indice = i%d
                 bloque = floor(i/d)
                 nuevo = PERM.index(indice)
                 M[(bloque*d)+nuevo] = L[i]
             cad = "".join(M)
             return cad
```

```
In [46]: desen = desencriptar(encrip, d, PERM)
         print desen
```

```
THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCOUNTINGCALCULATIONMEASURI
```

```
In [59]: def limpiarTexto(texto):
             L = list(texto)

             i = len(L)-1
             while(1):
                 if L[i] == ".":
                     L.pop(i)
                     i = i - 1
                 else:
                     break
             cad = "".join(L)
             return cad
```

```
In [60]: textoLimpio = limpiarTexto(desen)
         print textoLimpio
```

```
THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCOUNTINGCALCULATIONMEASURI
```

```
In [61]: print texto == textoLimpio

True
```

Ejercicio 3

(3 puntos) En este otro ejercicio hay que encriptar, y desencriptar, un texto corto usando una variante de RSA que se explica a continuación.

En primer lugar necesitamos una función, $raiz(c, p)$, que reciba como argumentos un primo p y un entero $0 < c < p$ y devuelva una lista con las dos raíces cuadradas de c en $\mathbb{Z}_p$ si existen o bien una lista vacía. Como sabemos, las dos raíces son opuestas una de la otra, es decir, se pueden representar mediante dos enteros positivos que suman p. Programa una tal función mediante un método de fuerza bruta.

Necesitamos una segunda función, $chinodelresto(a, b, p, q)$, que reciba como argumentos dos enteros positivos a y b y dos primos p y q, y devuelva un entero n tal que su resto módulo p es igual al de a y su resto módulo q es igual al de b . Programa una tal función mediante un método de fuerza bruta.

Las funciones anteriores se usarían para desencriptar, pero para valores grandes de p y q no servirían. Sage dispone de funciones eficientes para hacer esto: para calcular las raíces cuadradas de c en $\mathbb{Z}_p$ usamos la combinación $R = Integers(p); Lp = R(c).sqrt(all = True)$ y Lp sería la misma lista que nos devuelve $raiz(c, p)$, y para calcular el n que devuelve la función $chinodelresto(a, b, p, q)$ basta usar la función de Sage $crt(a, b, p, q)$ (teorema chino del resto). OBSERVACIÓN IMPORTANTE: la primera función devuelve un par de clases de restos y la segunda necesita que sus argumentos sean enteros. Para que la segunda funcione sobre lo que devuelve la primera hay que forzar las clases de restos al anillo de los enteros.

Convertimos el texto en un entero (codificación) con el mismo procedimiento que en RSA, con alfabeto de 26 letras.

Elegimos un entero suficientemente grande n que sea el producto de dos primos distintos y bastante distantes. Este n será la clave pública. £Cuán grande debe ser n? Igual que en RSA tiene que ser suficientemente grande para que todos los mensajes posibles de la longitud del que queremos encriptar, supongamos que es N, se puedan representar, mediante el paso 3, como un entero m menor que n. No es necesario definir funciones que generen los primos p y q aleatoriamente dentro de rangos grandes (ésto serviría para generar claves seguras) sino que basta con encontrar primos que cumplan las condiciones requeridas en el punto 6.

Encriptamos m elevándolo al cuadrado módulo N. De la misma forma que en RSA, podemos convertir el entero resultante en texto (descodificación) de $N + 1$ caracteres si hemos tenido cuidado de elegir n entre 26^N y 26^{N+1}. El texto resultante será el mensaje encriptado.

La clave privada está formada por los dos factores (p, q) de n. Conocidos los dos primos y el mensaje encriptado $m_e := m^2$, podemos calcular las raíces cuadradas a y b de m_e módulo p y q y aplicando la función del teorema chino del resto recuperar m. Hay que tener en cuenta que como m_e tiene dos raíces módulo p y otras dos módulo q, en total hay cuatro pares de combinaciones de una raíz módulo p y otra módulo q, y sólo uno de los cuatro posibles m obtenidos es el correcto. Finalmente hay que descodificar los cuatro posibles m, como en RSA, para obtener cuatro textos y el que sea legible debe ser el correcto.

```
In [135]: texto = 'ESTEEXAMENESINHUMANOYSILOSENOVENGO'
```

```
In [136]: def raiz(c, p):
              L = list()
              for i in xsrange (-p, p):
                  a = i%p
                  for j in xsrange(-p, p):
                      b = j%p
                      if(a+b == p and (power_mod(a, 2, p)==c or power_mod(b, 2, p)==c)):
                          L = [a,b]
                          return L
              return L

In [137]: def chinodelresto(a, b, p, q):
              n = 1
              while(1):
                  if((n%p == a%p) and (n%q == b%q)):
                      return n
                  n += 1

In [138]: p = 19
          c = 5

In [139]: raiz(c, p)

Out[139]: [9, 10]

In [140]: R = Integers(p)
          Lp = R(c).sqrt(all=True)
          print Lp

[9, 10]
```

Podemos observar que nuestra funcion consigue respuestas correctas (poco eficientes al ser con fuerza bruta).

```
In [141]: a=randint(1, 100)
          b=randint(1,100)
          p=nth_prime(randint(1, 100))
          q=nth_prime(randint(1, 100))
          chinodelresto(a, b, p, q)

Out[141]: 13077

In [142]: crt(a, b, p, q)

Out[142]: 13077
```

Parece que nuestra segunda función también trabaja bien

```
In [165]: alfb = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

```
In [166]: L_alfb = list(alfb)

In [167]: def ord2(c):
              return L_alfb.index(c)

In [168]: def chr2(n):
              return L_alfb[n]

In [169]: COD = [ord2(c) for c in texto]
          print COD

[4, 18, 19, 4, 4, 23, 0, 12, 4, 13, 4, 18, 8, 13, 7, 20, 12, 0, 13, 14, 24, 18, 8, 11, 14, 18,


In [170]: N = len(texto)
          print N

34


In [171]: p = next_prime(26^(N//2)+32)
          q = next_prime(26^((N//2)+1)+123)
          print p
          print q

113382731538515072555422 7
294795102000139188644087 39


In [172]: n = p*q
          n

Out[172]: 33424673908950949331179005963247101732650437189753

In [173]: m = ZZ(COD, 26)
          print m

704601755783994605038376672032337899044472564292


In [174]: m2 = power_mod(m, 2, n)
          m2

Out[174]: 32582865167192486272984763523348811229422217628802

In [175]: DIG = m2.digits(base=26)
          print DIG

[14, 12, 23, 0, 2, 15, 12, 12, 19, 17, 14, 12, 18, 12, 6, 20, 1, 4, 1, 25, 17, 19, 20, 10, 8,
```

```
In [176]: from string import *
          M_ENC = join([chr2(item) for item in DIG],sep="");M_ENC

Out[176]: 'OMXACPMMTROMSMGUBEBZRTUKIPJFFPYIZIZ'

In [177]: M_ENC2 = [ord2(c) for c in M_ENC];
          print M_ENC2

[14, 12, 23, 0, 2, 15, 12, 12, 19, 17, 14, 12, 18, 12, 6, 20, 1, 4, 1, 25, 17, 19, 20, 10, 8,

In [178]: m2 = ZZ(M_ENC2,26); m2

Out[178]: 3258286516719248627298476352334881122942221762880 2

In [179]: R = Integers(p)
          Lp = R(m2).sqrt(all=True)
          R = Integers(q)
          Lq = R(m2).sqrt(all=True)
          print Lp
          print Lq

[52259270530139259273829 7, 61123461008375813281593 0]
[33385944977887966315975 9, 26140915702225122232811 80]

In [185]: resto1 = crt(Lp[0], Lq[0], p, q)
          resto2 = crt(Lp[0], Lq[1], p, q)
          resto3 = crt(Lp[1], Lq[0], p, q)
          resto4 = crt(Lp[1], Lq[1], p, q)

        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-185-a6ac0b4a479d> in <module>()
    ----> 1 resto1 = crt(Lp[Integer(0)], Lq[Integer(0)], p, q)
          2 resto2 = crt(Lp[Integer(0)], Lq[Integer(1)], p, q)
          3 resto3 = crt(Lp[Integer(1)], Lq[Integer(0)], p, q)
          4 resto4 = crt(Lp[Integer(1)], Lq[Integer(1)], p, q)

        /usr/local/SageMath/local/lib/python2.7/site-packages/sage/arith/misc.pyc in crt(a, b,
       2760         a = Integer(a) # otherwise we get an error at (b-a).quo_rem(g)
       2761     g, alpha, beta = XGCD(m, n)
    -> 2762     q, r = (b - a).quo_rem(g)
       2763     if r != 0:
       2764         raise ValueError("No solution to crt problem since gcd(%s,%s) does not divi
```

```
/usr/local/SageMath/src/sage/structure/element.pyx in sage.structure.element.Element._
1314             return (<Element>left)._sub_(right)
1315         if BOTH_ARE_ELEMENT(cl):
-> 1316             return coercion_model.bin_op(left, right, sub)
1317
1318         try:


/usr/local/SageMath/src/sage/structure/coerce.pyx in sage.structure.coerce.CoercionMode
1102         # We should really include the underlying error.
1103         # This causes so much headache.
-> 1104         raise bin_op_exception(op, x, y)
1105
1106     cpdef canonical_coercion(self, x, y):


TypeError: unsupported operand parent(s) for -: 'Ring of integers modulo 2947951020001
```

Ni idea como solucionar esto

In [ ]: