Examen de ejemplo

December 11, 2017

Ejercicio1

Decimos que un entero positivo n es multiplicativamente perfecto si el producto de todos los divisores de n vale exactamente n^2. El ejemplo más sencillo de un número multiplicativamente perfecto es el producto $n = p \cdot q$ de dos primos distintos. Por tanto, existen infinitos enteros multiplicativamente perfectos. Queremos caracterizar los enteros multiplicativamente perfectos.

Primero define una función de Sage, de nombre $perfecto(n)$, que reciba como argumento un entero n y devuelva *Truo* o *False* según el número n sea multiplicativamente perfecto o no.

Ahora define una función de Sage, de nombre $perfectos(N)$, que reciba como argumento un entero N y devuelva la lista de todos los enteros multiplicativamente perfectos que pertenecen al intervalo $[1, N]$.

Usando las listas de enteros multiplicativamente perfectos que puedes obtener con la función del apartado anterior, produce una conjetura razonable acerca de qué enteros son multiplicativamente perfectos. Debes escribir explícitamente tu conjetura en una celda de texto.

Define una tercera función, de nombre $comprobar(N)$, que devuelva *True* si tu conjetura es correcta para enteros positivos menores que N, y *False* si no lo es.

Experimenta con valores de N suficientemente grandes para obtener el $N = 10^t$ (t entero) más grande tal que tu programa $comprobar(N)$ se ejecuta en menos de un minuto.

```
In [12]: #Ejercicio 1
         def perfecto(n):
             L=divisors(n)
             cont=1
             for i in srange(len(L)):
                cont=cont*L[i]
             if cont == n^2:
                 return True
             return False
```

```
In [16]: def perfectos(N):
             L=list()
             for i in srange(N):
                 if perfecto(i+1) == True:
                     L.append(i+1)
             return L
```

```
In [19]: perfectos (50)
```

```
Out[19]: [1, 6, 8, 10, 14, 15, 21, 22, 26, 27, 33, 34, 35, 38, 39, 46]
```

Son todos los de la forma n=p*q con p y q primos

```
In [24]: def comprobar(N):
             L=perfectos(N)
             for i in srange (len(L)):
                 if L[i] == 1:
                     continue
                 K=divisors(L[i])
                 if len(K) != 4:
                     return False
             return True
```

```
In [32]: time comprobar (1000)

CPU times: user 108 ms, sys: 0 ns, total: 108 ms
Wall time: 91.4 ms


Out[32]: True
```

Ejercicio 2

En este ejercicio queremos definir funciones para comparar listas de enteros, salvo el orden de sus elementos, es decir, diremos que dos listas son casi-iguales si tienen la misma longitud y los mismos elementos pero no necesariamente en los mismos lugares. Las funciones que vamos a definir son 'tests' de casi-igualdad de listas.

Primer método: después de comparar longitudes, y devolver *False* si son diferentes, recorrer los elementos de la primera lista comprobando si son elementos de la segunda también. Si encontramos un elemento que está en la primera lista y no en la segunda es claro que debemos devolver *False*, pero cada vez que encontramos un elemento que está en las dos listas £qué debemos hacer? Completa la idea para este método y define una función de Sage que lo implemente.

Segundo método: después de comparar longitudes, y devolver *False* si son diferentes, ordena las dos listas (puedes usar el método *sort* de Sage). Ahora compara las listas ordenadas recorriendo los elementos de la primera y comprobando si es igual al que ocupa el mismo lugar en la segunda. Define una función de Sage que implemente este método.

Compara la eficiencia de los dos métodos y discute los resultados obtenidos. Debes tener en cuenta que los tiempos obtenidos dependerán mucho de lo diferentes que sean las listas y debemos esperar que se obtengan tiempos mayores, y es el caso que nos conviene analizar, cuando una de las listas sea una reordenación de la otra. Para producir una reordenación aleatoria de una lista L se pueden ejecutar las siguientes instrucciones en una celda

import numpy as np

$L1 = np.random.permutation(L).tolist()$

y queda definida la lista $L1$ que es una permutación aleatoria de la lista L.

```
In [51]: #Ejercicio 2
         def primer_metodo(L,K):
             if len(L)!=len(K):
                 return False
```

```
            Flag=0
            for i in srange (len(L)):
                Flag=1
                for j in srange (len(K)):
                    if L[i]==K[j]:
                        Flag=0
                if Flag==1:
                    return False
            return True
```

In [52]:
```
L=[1,2,3,4,5,6,7,8,9]
K=[9,8,7,6,5,4,3,2,1]
J=[2,3,4,5,6,7,8,9,0]
R=[1,2,3]
print primer_metodo(L,K)
print primer_metodo(L,J)
print primer_metodo(L,R)
```

```
True
False
False
```

In [49]:
```
def segundo_metodo(L,K):
    if len(L)!=len(K):
        return False
    L.sort()
    K.sort()
    for i in srange (len(L)):
        if L[i]!=K[i]:
            return False
    return True
```

In [50]:
```
L=[1,2,3,4,5,6,7,8,9]
K=[9,8,7,6,5,4,3,2,1]
J=[2,3,4,5,6,7,8,9,0]
R=[1,2,3]
print segundo_metodo(L,K)
print segundo_metodo(L,J)
print segundo_metodo(L,R)
```

```
True
False
False
```

In [53]:
```
time primer_metodo(L,K)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 1.16 ms
```

```
Out[53]: True

In [54]: time segundo_metodo(L,K)

CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 195 µs


Out[54]: True
```

Ejercicio 3

En este ejercicio estudiamos la existencia de raíces k-ésimas en el anillo $\mathbb{Z}_m$ de clases de restos módulo un entero m . Como es natural, dados elementos $a, b \in \mathbb{Z}_m$, decimos que a es una raíz k-ésima de b si $a^k = b$ en $\mathbb{Z}_m$. Nuestro objetivo es, dado el entero m, determinar los valores de $k \leq m$ tales que todos los elementos de $\mathbb{Z}_m$ tienen una raíz k-ésima.

Define una función de Sage, con nombre $raices(m,k)$, que devuelva *True* si todas las clases de restos módulo m tienen una raíz k-ésima, con $k \leq m$, y *False* en caso contrario.

Aplica la función $raices(m,k)$ con m primo, por ejemplo $m = 23$, y trata de entender (explicita una conjetura) cuáles son los valores de k para los que se obtiene *True*. Define una nueva función de Sage, $comprobador(N)$ que sirva para comprobar si tu conjetura es cierta para todos los enteros primos del intervalo $[1, N]$. Mediante esta función comprueba tu conjetura hasta el $N = 10^t$ (t entero) más grande tal que la comprobación tarde menos de un minuto.

Si todavía tienes tiempo y ganas puedes estudiar el caso de enteros m que son producto de dos primos distintos.

```
In [67]: #Ejercicio 3
         # la clase del 0 (m) y del uno siempre tienen
         def raices (m,k):
             L=list()
             for j in srange(0,m):
                 puente=((j^k)%m)
                 L.append(puente)
             L.sort()
             for i in srange (len(L)):
                 if L[i]!=i:
                     return False
             return True


In [74]: for i in srange(100):
             if raices(23,i) == True:
                 print i

1
3
5
7
9
13
```

```
15
17
19
21
23
25
27
29
31
35
37
39
41
43
45
47
49
51
53
57
59
61
63
65
67
69
71
73
75
79
81
83
85
87
89
91
93
95
97
```

```
In [82]: def comprobadorej3(N):
             for j in srange(2,N+1):
                 if is_prime(j):
                     for i in srange(1,N+1):
                         if i%2 == 1:
                             if raices(j,i) == False:
                                 return False
```

```
                        elif raices(j,i) == True:
                            return False
            return True
```

In [83]:
```
comprobadorej3(100)
```

Out[83]:
```
False
```

In [ ]: