euler-6ej

November 4, 2017

0.1 Ejercicio 13

0.1.1 Apartado a)

Goldbach conjeturó que todo entero compuesto e impar es la suma de un primo y el doble de un cuadrado. Así, por ejemplo, $9 = 7 + 2 \cdot 1^2$, $15 = 7 + 2 \cdot 2^2$, $21 = 3 + 2 \cdot 3^2$, etc. Esta conjetura resultó ser falsa. Determina el menor entero que no cumple lo conjeturado por Goldbach.

```
In [2]: def goldbach(N):
            testigo = 0
            for n in xsrange(2,N):
                testigo = 0
                if (not is_prime(n)) and (n%2 == 1):
                    for p in prime_range(2,n):
                        if (n-p)%2 == 0 and is_square((n-p)//2):
                            #L.append(n)
                            testigo = 1
                            break
                    if testigo == 0:
                             return n
            return testigo
```

```
In [3]: goldbach(10000)
```

```
Out[3]: 5777
```

0.1.2 Apartado b)

Existen enteros, por ejemplo 145, que son iguales a la suma de los factoriales de sus dígitos. Determina **todos** los enteros con esta propiedad.

```
In [4]: def dig_fact(N):
            L = []
            for n in xsrange(3,N):
                L1 = n.digits()
                if n == sum(map(factorial,L1)):
                    L.append(n)
            return L
```

```
In [5]: dig_fact(10**4)
```

```
Out[5]: [145]
```

```
In [6]: dig_fact(10**5)
```

```
Out[6]: [145, 40585]
```

```
In [7]: %time dig_fact(10**6)
```

```
CPU times: user 10.9 s, sys: 148 ms, total: 11 s
Wall time: 11 s
```

```
Out[7]: [145, 40585]
```

```
In [8]: %time dig_fact(10**7)
```

```
CPU times: user 1min 55s, sys: 332 ms, total: 1min 56s
Wall time: 1min 55s
```

```
Out[8]: [145, 40585]
```

Pasar de 10^6 a 10^7 no aporta nuevas soluciones, pero para demostrar que no hay otras hay que probar *matemáticamente* que existe un N_0 tal que no hay soluciones para $N \geq N_0$. Conseguido ésto, tenemos que calcular las soluciones que pueda haber hasta N_0 usando el ordenador.

Para en entero de k cifras el valor máximo que puede tener la suma de los factoriales de sus cifras es $k \times 9! = k \cdot 362880$.

```
In [9]: print factorial(9);print 7*factorial(9);print 8*factorial(9)
```

```
362880
2540160
2903040
```

Ahora, el número de cifras decimales de $k \times 9!$ es la parte entera por exceso de $log_{10}(k) + log_{10}(362880)$ y basta determinar un k tal que $k - 1 > log_{10}(k) + log_{10}(362880)$.

```
In [10]: print (log(7,base=10)+ log(362880,base=10)).n(); print (log(8,base=10)+ log(362880,bas
```

```
6.40486107289105
6.46285301986874
```

Vemos entonces que para enteros de 8 o más cifras decimales es imposible que el entero sea igual a la suma de los factoriales de sus cifras, porque la suma de los factoriales de las cifras se escribe con menos cifras decimales que el número. Debemos buscar soluciones hasta $N = 10^8$, lo que puede tardar más de 15 minutos.

```
In [11]: %time dig_fact(10**8)
```

```
CPU times: user 20min 41s, sys: 7.33 s, total: 20min 49s
Wall time: 20min 41s
```

```
Out[11]: [145, 40585]
```

0.1.3 Apartado c)

Determina todas las tripletas de enteros primos de 4 cifras tales que cumplen las dos condiciones siguientes: 1) Los 3 enteros están en progresión aritmética, es decir, el segundo menos el primero es igual al tercero menos el segundo. 2) Los tres enteros de la tripleta tienen las mismas cifras y cada cifra aparece el mismo número de veces en cada uno de ellos.

```
Por ejemplo, $(1487, 4817, 8147)$ es una de las soluciones.
```

```
In [12]: def primer_intento():
             LL = []
             L = prime_range(1000,9999)
             n = len(L)
             for n1 in xsrange(n):
                 for n2 in xsrange(n1+1,n):
                     for n3 in xsrange(n2+1,n):
                         if L[n2]-L[n1]==L[n3]-L[n2]:
                             if L[n2].digits() in Permutations(L[n1].digits()) and \
                             L[n3].digits() in Permutations(L[n1].digits()):
                                 LL.append((L[n1],L[n2],L[n3]))
             return LL
```

```
In [13]: %time primer_intento()
```

```
CPU times: user 1min 13s, sys: 8 ms, total: 1min 13s
Wall time: 1min 13s
```

```
Out[13]: [(1487, 4817, 8147), (2969, 6299, 9629)]
```

£Podemos mejorar el tiempo? Quizá reduciendo el número de bucles anidados

```
In [14]: def segundo_intento():
             LL = []
             L = prime_range(1000,9999)
             n = len(L)
             print n
             for n1 in xsrange(n):
                 for k in xsrange(1,n):
                     if n1+k < n:
                         r = L[n1+k]-L[n1]
                         if L[n1+k]+r in L:
                             if L[n1+k].digits() in Permutations(L[n1].digits()) and \
                             (L[n1+k]+r).digits() in Permutations(L[n1].digits()):
                                 LL.append((L[n1],L[n1+k],L[n1+2*k]))
             return LL
```

```
In [15]: %time segundo_intento()
```

```
1061
CPU times: user 13 s, sys: 216 ms, total: 13.3 s
Wall time: 13.1 s
```

```
Out[15]: [(1487, 4817, 8521), (2969, 6299, 9811)]
```

0.1.4 Apartado d)

Sea (a, b, c) una tripleta de enteros positivos tal que existe un triángulo rectángulo con la longitud de los lados igual a los enteros de la tripleta. Podemos decir, por ejemplo, que una tal tripleta es **rectangular**. Sea p el perímetro de un tal triángulo. Para $p \leq 1000$ determina el perímetro p_m para el que existe el mayor número de tripletas rectangulares distintas con ese perímetro.

```
In [16]: def triangulos(n):
             '''n es el perimetro'''
             cont = 0
             for a in xsrange((n//2)+1):
                 for b in xsrange(a,n-a):
                     if n-a-b>0 and a**2+b**2==(n-a-b)**2:
                         cont += 1
             return cont,n

         def maximo(N):
             max = 0
             per = 3
             for ent in xsrange(3,N):
                 t,p = triangulos(ent)
                 if t > max:
                     max =t
                     per = p
             return max,per

         %time maximo(1000)
```

```
CPU times: user 1min 18s, sys: 24 ms, total: 1min 18s
Wall time: 1min 18s
```

```
Out[16]: (9, 840)
```

Con este planteamiento cada tripleta está ordenada en orden creciente $a < b < c$.

0.1.5 Apartado e)

Un **primo de Mersenne** es un entero primo de la forma $2^p - 1$ con p primo. No es difícil ver que si $2^n - 1$ es primo, entonces el exponente n debe ser también primo, pero el recíproco es falso. Los primos de Mersenne son los mayores conocidos porque hay criterios de primalidad bastante eficientes para candidatos a ser primo de Mersenne.

En 2004 se descubrió un primo muy grande que no es de Mersenne, concretamente se trata de $P := 28433 \times 2^{7830457} + 1$.

1. Determina el número de cifras decimales de P.
2. Determina las últimas, por la derecha, diez cifras decimales de P.
3. Determina las primeras, por la izquierda, diez cifras decimales de P.

Se entiende que, aunque debe ser posible calcular completamente P en los ordenadores del Laboratorio, debe hacerse este ejercicio sin pasar por ese cálculo. Puede usarse el cálculo completo de P para comprobar el resultado obtenido.

In [17]:
```
## Numero de digitos
# P = 28433*2^(7830457)+1

logP = log(28433,base=10)+7830457*log(2,base=10)
%time print floor(logP)+1
```

```
2357207
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 416 ţs
```

In [18]:
```
%time print (28433*power_mod(2,7830457,10^10)+1)%10^10
```

```
8739992577
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 196 ţs
```

In [19]:
```
## Comprobamos
P = 28433*2^(7830457)+1
%time print str(P)[-10:]
```

```
8739992577
CPU times: user 400 ms, sys: 8 ms, total: 408 ms
Wall time: 411 ms
```

In [20]:
```
def cifra_dom(k,n):
    '''Cifra dominante de k^n. Ver 7.5 de las notas del curso'''
    return floor(10^((n*log(k,base=10).n())-((n*log(k,base=10).n()).floor())))

def m_cifras_dom(m,k,n):
    '''m cifras mas dominantes de k^n. Ver 7.5 de las notas del curso'''
    return floor(10**m*10^((n*log(k,base=10).n())-((n*log(k,base=10).n()).floor())))

%time m_cifras_dom(15,2,7830457)
```

```
CPU times: user 4 ms, sys: 0 ns, total: 4 ms
Wall time: 1.24 ms
```

```
Out[20]: 2733738636872552
```

```
In [21]: R = RealField(prec=100)
         x =R(log(28433,base=10)+7830457*log(2,base=10))
         print floor(10**9*10^(x-(x.floor())))
```

```
7772839072
```

```
In [22]: ## Comprobamos
         P = 28433*2^(7830457)+1
         %time print str(P)[:10]
```

```
7772839072
CPU times: user 404 ms, sys: 4 ms, total: 408 ms
Wall time: 407 ms
```

0.1.6 Apartado f)

Hay polinomios de grado 2 que, como $p(x) := x^2 + x + 41$, toman valores primos para los primeros valores enteros consecutivos de x. El polinomio $p(x)$ indicado toma valores primos para $x = 0, 1, 2, \ldots, 39$, pero el valor es compuesto para $x = 40$. Entonces, para $p(x)$ se obtiene una sucesión de 40 primos, que por supuesto **no** son consecutivos, al evaluarlo en los enteros consecutivos del intervalo $[0, 39]$.

Determina, de entre todos los polinomios de la forma $x^2 + ax + b$, con a y b de valor absoluto menor o igual a 1000, el polinomio que produce el **mayor número de valores primos al evaluarlo en enteros consecutivos** $x = 0, 1, 2, \ldots$. Indica también el número de valores primos obtenido, que será mayor o igual a 40.

No se sabe si existen polinomios de grado dos que tomen infinitos valores primos, aunque se conjetura que los hay (ver https://en.wikipedia.org/wiki/Bunyakovsky_conjecture).

```
In [23]: def polinimios(N):
             R.<x> = ZZ[]
             L = prime_range(-N,N)
             max = 40
             pm = R(x^2+x+41)
             for b in L:
                 for a in xsrange(-N,N):
                     if is_prime(a+b+1):
                         p = R(x^2+a*x+b)
                         n = 0
                         valor = p(x=n)
                         while is_prime(valor):
                             n += 1
```

```
                    valor = p(x=n)
                if n > max:
                    max = n+1
                    pm = p
        return max,pm

    %time print polinimios(10**3)
```

```
(72, x^2 - 61*x + 971)
CPU times: user 1.22 s, sys: 4 ms, total: 1.22 s
Wall time: 1.22 s
```

In [ ]: