Una aproximación al reto de Edouart Lucas. Cuadro Mágico de 3 por 3 de números enteros positivos al cuadrado.

INDICE

1.- Introducción

Parece ser que se conocen los Cuadros Mágicos (CM) desde hace algunos miles de años. No quiero ser yo el que os documente, os pondré a continuación algunos de los muchos enlaces que hay en Internet donde os contarán que es un CM.

https://es.wikipedia.org/wiki/Cuadrado_m%C3%A1gico

https://sites.google.com/site/loscuadradosmagicos/2-concepto

https://www.abc.es/ciencia/abci-cuadrados-magicos-nadie-podido-resolver-y-premian-6500-euros-201707251015_noticia.html

2.- Métodos de Resolución

De la misma manera, hay multitud de sitios web en Internet donde os informarán como construirlos, de orden 3x3, 4x4, 5x5 …

https://es.wikihow.com/resolver-un-cuadro-m%C3%A1gico

http://matematicasentumundo.es/TEXTOS/cuadrados%20magicos.pdf

3.- La casillas más significativa

Vamos a ir centrando nuestro objetivo, que no es más que un CM de orden 3x3 (no olvidar de 3x3), y después de ver algunos métodos de resolución, quizás el más interesante que he encontrado en Internet para nuestros propósitos, es decir no lo he inventado yo, sea el que os pongo a continuación, algunos puede que ya lo conozcan.

a + b	a - (b + c)	a + c
a - (b – c)	a	a + (b – c)
a - c	a + (b + c)	a - b

Veis, la resolución de un CM de 3x3 que se reduce a tres variables: a, b, c. Podemos jugar con esta tres variables, pero siempre con cuidado, u obtendremos números negativos, o alguna casilla con igual número que otra.
El CM más simple lo obtendremos asignando a las variables los siguientes valores: a= 5 , b= 3 y c = 1, que no es más que el CM más pequeño de orden 3x3. Ahí va:

8	1	6
3	5	7
4	9	2

Cualquier CM de orden 3x3 que encontréis se va a ajustar a este método de creación que hemos comentado, podréis deducir de estas 3 variables a, b, c.

Seamos más precisos, porque estoy viendo que a alguno se va a quejar.

Este método se construye partiendo de sus diagonales, que cubren 5 casillas de la 9 que tiene un CM de 3x3, las otras 4 casillas se deducen de las anteriores. Pongo un ejemplo.

a + b	a - (b - c)	a - c
a - (b + c)	a	a + (b + c)
a + c	a + (b - c)	a - b

Veis que hemos intercambiado la casilla (1,3) con la (3,1) con respecto al primer cuadro de obtención de CM que os presenté.

La casilla (1,2) se deduce de la (1,1) y de la (1,3) que no es más que restar o sumar lo que hemos añadido a la variable a. Si os fijáis la casilla (2,3) es restar en este caso lo que se sumo a la variable a en la casilla(1,2). De igual manera el resto de casillas que nos faltan. Para esta nueva presentación el CM quedaría de la siguiente forma:

8	3	4
1	5	9
6	7	2

Presentemos todas las posibilidades de las diagonales, vosotros podéis deducir el resto de casillas :

a+b		a+c
	a	
a-c		a-b

a+b		a-c
	a	
a+c		a-b

a-b		a+c
	a	
a-c		a+b

a-b		a-c
	a	
a+c		a+b

Y ahora contra estos 4 cuadros que he presentado intercambiando la variable b por la c. Tenemos en total 8 combinaciones.

Imagino que ya os habréis dado cuenta que **la casilla más significativa es la central la (2,2) la que corresponde a la variable (a)**, es mas, en nuestro método y sus variantes no cambian como lo puede hacer el resto de casillas.

Además (a) siempre será el valor medio (no la media) de las 9 valores de un CM de orden 3x3.

1, 2, 3, 4, 5, 6, 7, 8, 9 → a = 5
4, 5, 6, 7, 8, 9 ,10, 11 ,12 → a = 8

4.- El reto de Edouart Lucas

Edouart Lucas (https://es.wikipedia.org/wiki/%C3%89douard_Lucas) propuso en el año 1876 el siguiente reto que todavía anda por resolver.

“ Encontrar un Cuadro Mágico de orden 3 por 3 de números enteros positivos al cuadrado o demostrar que es matemáticamente imposible hallarlo.“

He aquí el cuadro a resolver

a^2	b^2	c^2
d^2	e^2	f^2
g^2	h^2	i^2

Como ya conoceréis nuestro cuadro de 3 por 3 a de cumplir las siguientes condiciones para ser considerado un Cuadro Mágico:

a, b, c, d, e, f, g, h, i → son todos distintos entre si y mayores que cero y además:

La suma de los elementos filas, columnas y diagonales ha de ser iguales:

Filas:

$$a^2+b^2+c^2=K$$

$$d^2+e^2+f^2=K$$

$$g^2+h^2+i^2=K$$

Columnas:

$$a^2+d^2+g^2=K$$

$$b^2+e^2+h^2=K$$

$$c^2+f^2+i^2=K$$

y las dos diagonales:

$$a^2+e^2+i^2=K$$

$$c^2+e^2+g^2=K$$

En la siguiente pagina web podéis encontrar 2 buenas aproximaciones:

http://www.multimagie.com/

373^2	289^2	565^2
360721	425^2	23^2
205^2	527^2	222121

127^2	46^2	58^2
2^2	113^2	94^2
74^2	82^2	97^2

Mejor la segunda, sólo falla en una de las diagonales.

5.- Lanzarse a lo bruto (a por la búsqueda de las diagonales)

Nosotros vamos a intentarlo con el método que comentamos anteriormente, lo volveré a presentar para números al cuadrado:

$a^2 + b$	$a^2 - (b + c)$	$a^2 + c$
$a^2 - (b – c)$	a^2	$a^2 + (b – c)$
$a^2 - c$	$a^2 + (b + c)$	$a^2 - b$

NOTA: Nosotros vamos a por todas, los cuadros que os presenté en el apartado anterior nunca llegarían a Cuadro Mágico, no cumplen el que el valor medio de los elementos ocupe la casilla central la (2,2). Los llaman Pseudo Mágicos. No obstante tiene un curro elevado llegar a ellos.

Entramos en materia:

Como ya habréis deducido la constante K, ahora es tres veces a^2.
Vamos ha empezar a lo bruto a por la diagonal.

Tomemos un número < a > elevemoslo al cuadrado y vayamos restándole desde el uno hasta el mismo número al cuadrado de modo que se cumpla las siguiente 2 condiciones:

1º $\sqrt{a^2-b}$ **su solución es un número entero.**

2º $\sqrt{a^2+b}$ **su solución es también un número entero**

Un par de ejemplos visuales de lo que pretendemos:

Fijaros en el número 5 (b=24):
$5^2 - 24 = 1 = 1^2$
$5^2 + 24 = 49 = 7^2$
$1^2 + 5^2 + 7^2 = 3 \times 5^2 = 75$

Ahora con el número 10 (b=96):
$10^2 - 96 = 4 = 2^2$
$10^2 + 96 = 196 = 14^2$
$2^2 + 10^2 + 14^2 = 4 + 100 + 196 = 3 \times 10^2 = 300$

Bien, ahora entraremos en el mundo de la programación. El lenguaje que utilizaré es python, un lenguaje interpretado multipropósito, no es el más indicado para estas labores, pero se construye con el muy rápido. Posiblemente mejor con otros lenguajes como: C o Fortran, este último con una sintaxis algo arcaica, pero mucho más rápidos en tiempo de ejecución. Junto a este documento os anexaré los diferentes programás con los que vamos obteniendo los diferentes resultados.
Comenzamos con el primero:

dia.py
Implementa lo comentado anteriormente.

En el encontrareis dos variables numéricas < ini > y < fin >, con las que podéis jugar para recoger hasta donde queréis llegar en la búsqueda. En mi caso lo he hecho en 3 bloques de 300. De 1 a 300 de 301 600 y de 601 a 900. Como algunos podéis imaginar los tiempos de ejecución se incrementan considerablemente cuando pedimos los bloques más altos. Acordaros de que por cada número hemos de empezar desde el uno hasta el cuadrado de ese número, . Veamos un trozo de una de las salidas de uno de los bloques, en donde las 3 primeras columnas representan cada uno de los 3 elementos de la diagonal siendo la del medio el valor de la (a) , la siguiente columna es nuestra (b ó c) y el el ultimo es la constante (K). He aquí el corte para que lo veáis.

```
182  130   26 b: 16224 K: 50700
189  135   27 b: 17496 K: 54675
184  136   56 b: 15360 K: 55488
193  137   17 b: 18480 K: 56307
196  140   28 b: 18816 K: 58800
187  143   77 b: 14520 K: 61347
161  145  127 b: 4896 K: 63075
167  145  119 b: 6864 K: 63075
203  145   29 b: 20184 K: 63075
205  145    5 b: 21000 K: 63075
206  146   14 b: 21120 K: 63948
188  148   92 b: 13440 K: 65712
191  149   89 b: 14280 K: 66603
186  150  102 b: 12096 K: 67500
210  150   30 b: 21600 K: 67500
207  153   63 b: 19440 K: 70227
217  155   31 b: 23064 K: 72075
204  156   84 b: 17280 K: 73008
217  157   47 b: 22440 K: 73947
219  159   51 b: 22680 K: 75843
224  160   32 b: 24576 K: 76800
```

Fijaros hay números que no nos dan ninguna diagonal, 131,132,133,134,139,141 …
Otros una sola diagonal 130,135, 136 …
Otros 2 diagonales como es el 150.

Quizás el más interesante sea el 145 hemos encontrado 4 conjuntos que cumplen las 2 condiciones.

Veamos el primer conjunto (161, 145, 127) :

$\sqrt{145^2+4896}=161$ y además $\sqrt{145^2-4896}=127$ y además $161^2+145^2+127^2=3\times145^2$

¿ Que hemos resuelto ?, siento defraudaros solo hemos conseguido, 4 de las 8 condiciones de un CM, os presento el cuadro apoyándonos en lo obtenido con el 145:

161^2	203^2	167^2
205^2	145^2	5^2
119^2	29^2	127^2

Diagonal 1ª	$161^2 + 145^2 + 127^2$	= 63075
Diagonal 2ª	$167^2 + 145^2 + 119^2$	= 63075
2ª Fila	$205^2 + 145^2 + 5^2$	= 63075
2º Columna	$203^2 + 145^2 + 29^2$	= 63075

El resto ni por asomo:

1º fila	$161^2 + 203^2 + 167^2$	= 95019
3º fila	$119^2 + 29^2 + 127^2$	= 31131
1º columna	$161^2 + 205^2 + 119^2$	= 82107
3º columna	$167^2 + 5^2 + 127^2$	= 44043

Ha sido un buen intento pero insuficiente. Además con este método en cuanto (a) va creciendo nuestro equipo necesita cada vez más tiempo para encontrar más números. Y no por llegar más lejos encontramos más conjuntos de números, de momento.

6.- Entender la generación de números al cuadrado en base a uno dado. Tanto crecientes como decrecientes. “La llave”

Entender estos conceptos nos va a permitir ser más eficientes a la hora de localizar nuestras diagonales.

Vamos a empezar en orden creciente:

Dado un número calcularemos su cuadrado y la sucesión de sus siguientes también al cuadrado. Tomaremos como ejemplo el número 650, esto que veremos con el 650 lo podéis hacer con cualquier numero. Os escribo directamente la sintaxis python dado su sencillez:

```
for i in range(650,660):
     print("{0:5d} {1:5d}".format(i, i**2) )
```

Nuestra salida sería:

```
650 422500
651 423801
652 425104
653 426409
654 427716
655 429025
656 430336
657 431649
658 432964
659 434281
```

Todavía no nos dice nada ¿verdad?.

Hagamos ahora lo siguiente: Tomemos como base el 650 al cuadrado y vayamos observando como se generan las siguientes restas:

$651^2 - 650^2$
$652^2 - 650^2$
$653^2 - 650^2$
….

Vuelvo a poneros la sintaxis en python de lo que vamos a hacer dada su sencillez:

```
base = 650**2
for i in range (651, 661):
      print("{0:5d} {1:5d}".format(i,(i**2)-base) )
```

Nuestra salida será:

```
  651  1301
  652  2604
  653  3909
  654  5216
  655  6525
  656  7836
  657  9149
  658 10464
  659 11781
  660 13100
```

Nos interesa ver como se construye la 2º columna, por tanto lo obtenido es equivalente a lo siguiente: Nos interesa la 2 columna, vamos a retorcerla un poco.

$651^2 = 650^2 + 1301$
$652^2 = 650^2 + 2604$
$653^2 = 650^2 + 3909$
$654^2 = 650^2 + 5216$
$655^2 = 650^2 + 6525$
$656^2 = 650^2 + 7836$
$657^2 = 650^2 + 9149$
….

Estrujemos la ultima columna un poco:

1301 = 1301 + 0
2604 = (2 x 1301) + 2
3909 = (3 x 1301) + 6
5216 = (4 x 1301) + 12
6525 = (5 x 1301) + 20
7836 = (6 x 1301) + 30
9149 = (7 x 1301) + 42
...

Ahora un poco mas:

1301 = (1 x 1300) + 1
2604 = (2 x 1300) + 4

3909 = (3 x 1300) + 9
5216 = (4 x 1300) + 16
6525 = (5 x 1300) + 25
7836 = (6 x 1300) + 36
9149 = (7 x 1300) + 49
...
Esto es bastante más interesante, pero aun lo vamos a hacer más interesante:

1301 = (1 x 2 x 650) + 1
2604 = (2 x 2 x 650) + 4
3909 = (3 x 2 x 650) + 9
5216 = (4 x 2 x 650) + 16
6525 = (5 x 2 x 650) + 25
7836 = (6 x 2 x 650) + 36
9149 = (7 x 2 x 650) + 49

Ya lo habéis visto, Ahora podemos deducir como generar los siguientes números cuadrados en base a uno dado en nuestro caso el 650:

$$650^2+(2\times n\times 650)+n^2$$

Por tanto nuestro generador para cualquier número será:

$$Base^2+(2\times n\times Base)+n^2$$

Ojo no olvidar que (n) siempre será mayor o igual a cero.
Os parece que lo probemos para el 123 y sus 10 cuadrados siguientes. Ahí va su código en python:

```
base = 123
for i in range(1,11):
      print( (base**2 + (2*i*base) + i**2)**0.5 )
```

Su salida como no podría ser de otra manera:

```
124.0
125.0
126.0
127.0
128.0
129.0
130.0
131.0
132.0
133.0
```

Ahora en orden decreciente:

Haremos algo similar a lo anterior, obtendremos los 10 cuadrados decrecientes al 650

```
for i in range(650,640,-1):
      print("{0:5d} {1:5d}".format(i, i**2) )
```

Nuestra salida sería:

```
 650 422500
 649 421201
 648 419904
 647 418609
 646 417316
 645 416025
 644 414736
 643 413449
 642 412164
 641 410881
```

Ahora restemos a 650^2 los siguientes cuadrados a el en orden decreciente algo así como:

$650^2 - 649^2$
$650^2 - 648^2$
$650^2 - 647^2$
…

Su código en python:

```
base = 650**2
for i in range(650,640,-1):
     print("{0:5d} {1:5d}".format(i, base - (i**2)) )
```

Nuestra salida sería:

```
650     0
649  1299
648  2596
647  3891
646  5184
645  6475
644  7764
643  9051
642 10336
641 11619
```

Nos interesa ver como se construye la 2º columna, por tanto lo obtenido es equivalente a lo siguiente:

$649^2 = 650^2 - 1299$
$648^2 = 650^2 - 2596$
$647^2 = 650^2 - 3891$
$646^2 = 650^2 - 5184$
$645^2 = 650^2 - 6475$
$644^2 = 650^2 - 7764$
$643^2 = 650^2 - 9051$
….

Estrujemos la ultima columna un poco:

1299 = (1 x 1299) - 0
2596 = (2 x 1299) - 2

3891 = (3 x 1299) - 6
5184 = (4 x 1299) - 12
6475 = (5 x 1299) - 20
7764 = (6 x 1299) - 30
9051 = (7 x 1299) - 42

Un poco mas:

1299 = (1 x 1300) - 1
2596 = (2 x 1300) - 4
3891 = (3 x 1300) - 9
5184 = (4 x 1300) - 16
6475 = (5 x 1300) - 25
7764 = (6 x 1300) - 36
9051 = (7 x 1300) - 49

Lo vais viendo verdad ?

1299 = (1 x 2 x 650) - 1^2
2596 = (2 x 2 x 650) - 2^2
3891 = (3 x 2 x 650) - 3^2
5184 = (4 x 2 x 650) - 4^2
6475 = (5 x 2 x 650) - 5^2
7764 = (6 x 2 x 650) - 6^2
9051 = (7 x 2 x 650) - 7^2

Podemos expresar la generación de cuadrados por debajo del 650 de la siguiente manera:

$650^2-((2\times n\times 650)-n^2)$ o también

$650^2-(2\times n\times 650)+n^2$

Generalizando:

$base^2-(2\times n\times base)+n^2$

Os parece que lo probemos para el 123 y sus 10 cuadrados anteriores. Ahí va su código en python:

```
base = 123
for i in range(1,11):
     print( (base**2 - (2*i*base) + i**2)**0.5 )
```

Su salida como no podría ser de otra manera:

```
122.0
121.0
120.0
119.0
118.0
117.0
116.0
```

```
115.0
114.0
113.0
```

7.- Con el conocimiento anterior, volver a lanzarse a por las diagonales.

Planteemos la ecuación de una diagonal:

$$(a^2+b)+a^2+(a^2-b)=3\times a^2$$

Así no vamos a llegar a nada, pero con los conocimientos anteriores sabemos que para que la siguiente expresión ($a^2 + b$) nos de raíces cuadradas perfectas se ha de cumplir que:

$$(a^2+b)=a^2+2na+n^2$$

De igual manera para que la siguiente expresión ($a^2 - b$) nos de raíces cuadradas perfectas se ha de cumplir que:

$$(a^2-b)=a^2-2pa+p^2$$

Sustituyamos lo obtenido en la ecuación de la diagonal:

$$a^2+2na+n^2+a^2+a^2-2pa+p^2=3a^2$$

Algo de reducción:

$$2na+n^2-2pa+p^2=0$$

Saquemos factor común a la (a):

$$a=(p^2+n^2)/2(p-n)$$

Olé, para un (a) definido hemos de buscar parejas (n,p) que cumplan nuestra ecuación. Tenemos algunos condicionantes, por ejemplo que (p-n) ha de ser siempre mayor que uno, no tiene sentido dividir por cero o por un número negativo. Bien y ahora el código que nos permita obtener esas parejas (p,n). En el código hemos pedido las 500 primeras parejas.

gen_p_n.py

Un trozo de su salida:

```
a:      5    p:      4    n:      2
a:      5    p:      6    n:      2
a:     13    p:      6    n:      4
a:     10    p:      8    n:      4
a:     25    p:      8    n:      6
a:     17    p:     10    n:      6
a:     41    p:     10    n:      8
a:     10    p:     12    n:      4
a:     15    p:     12    n:      6
a:     26    p:     12    n:      8
a:     61    p:     12    n:     10
```

```
a:     37    p:    14    n:    10
a:     85    p:    14    n:    12
a:     20    p:    16    n:     8
a:     50    p:    16    n:    12
a:    113    p:    16    n:    14
a:     15    p:    18    n:     6
a:     39    p:    18    n:    12
a:     65    p:    18    n:    14
a:    145    p:    18    n:    16
a:     13    p:    20    n:     4
a:     25    p:    20    n:    10
a:     34    p:    20    n:    12
a:     82    p:    20    n:    16
a:    181    p:    20    n:    18
a:    101    p:    22    n:    18
a:    221    p:    22    n:    20
a:     17    p:    24    n:     6
a:     20    p:    24    n:     8
a:     30    p:    24    n:    12
a:     52    p:    24    n:    16
a:     75    p:    24    n:    18
a:    122    p:    24    n:    20
a:    265    p:    24    n:    22
a:    145    p:    26    n:    22
a:    313    p:    26    n:    24
a:     29    p:    28    n:    12
a:     35    p:    28    n:    14
a:     74    p:    28    n:    20
a:    170    p:    28    n:    24
a:    365    p:    28    n:    26
...
```

Vemos por la derecha como evolucionan (p) y (n) y la izquierda las (a) que cumplen, la columna de las (a) no están en el mejor formato pues crecen y en un momento vuelve a un punto inferior y vuelve a crecer. Ya lo solucionaremos más adelante.

Analicemos (p y n) que observamos:

¿Por qué (p y n) son siempre parejas de números pares ?
Bien sin entrar en el porqué, podemos reducir las vueltas que tienen que dar (p y n) en el algoritmo haciendo que se incrementen de 2 en 2. El resultado será el mismo y el tiempo de proceso la mitad. Os anexo el código:

gen_p_n_2.py

Como podéis observar la salida es la misma.

Ahora vamos a optimizar el código de generación de diagonales. Para ello volveremos a analizar la ecuación:

$$a=(p^2+n^2)/2(p-n)$$

Separemos las (n) a un lado de la ecuación y a la otra las (p). Nos quedaría lo siguiente:

$$(2\times n\times a)+n^2=(2\times p\times a)-p^2$$

Analicemos la parte derecha de la ecuación:

$$(2\times p\times a)-p^2$$

Para que no nos de un valor negativo no podemos hacer crecer a (p) por encima de (2a). Si asignamos a (p=2a) nos dará cero, por encima un valor negativo.

$$(2\times 2a\times a)-(2a)^2=4a^2-4a^2$$

Reescribamos el código:

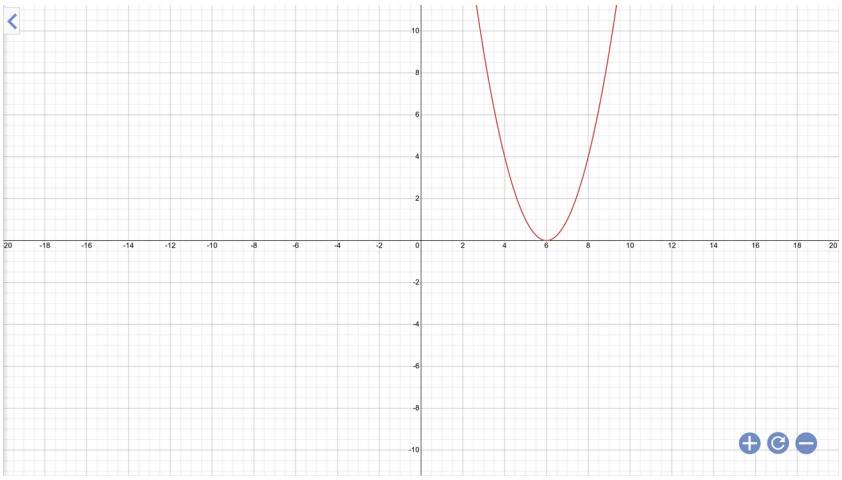
gen_p_n_3.py

El código esta preparado para decirnos qué números del uno al ochenta son capaces de darnos diagonales (podemos cambiar este número por el que queramos). Veamos un trozo de su salida:

```
a:     58   p:     60   n:     24   b /c:     3360
a:     60   p:     48   n:     24   b /c:     3456
a:     60   p:     72   n:     24   b /c:     3456
a:     61   p:     12   n:     10   b /c:     1320
a:     61   p:    110   n:     10   b /c:     1320
a:     65   p:     18   n:     14   b /c:     2016
a:     65   p:     30   n:     20   b /c:     3000
a:     65   p:     42   n:     24   b /c:     3696
a:     65   p:     52   n:     26   b /c:     4056
a:     65   p:     78   n:     26   b /c:     4056
a:     65   p:     88   n:     24   b /c:     3696
a:     65   p:    100   n:     20   b /c:     3000
a:     65   p:    112   n:     14   b /c:     2016
a:     68   p:     40   n:     24   b /c:     3840
a:     68   p:     96   n:     24   b /c:     3840
a:     70   p:     56   n:     28   b /c:     4704
a:     70   p:     84   n:     28   b /c:     4704
```

Vaya, parece que tenemos 8 diagonales para el número (a = 65), cuidado que no es real, la función que nos proporciona números al cuadrado en orden decreciente es una función cuadrática de la siguiente forma:

para
a=6



En nuestro caso toca el eje de abcisas para (p=6), o lo que es lo mismo siempre que (p = a), a partir de aquí los valores que nos devuelva pueden ser dobles. Volvamos al grupo de diagonales de (a = 65), fijaros como evolucionan (p y n), (p) en orden creciente y (n) crece llega a un punto lo repite y vuelve a decrecer con valores idénticos.
Si aplicáis la formula de raíces decrecientes para (a = 65) y (p = 18) y (a=65) y (p = 112), observareis que tenemos el mismo valor.

$65^2 - 2 . 18 . 65 + 18^2 = 65^2 - 2 . 112 . 65 + 112^2$ (2209)

Así entenderéis porque nos repite los valores de (n).

Por tanto descartemos los repetidos:

```
a:     65   p:     18   n:     14   b /c:     2016
a:     65   p:     30   n:     20   b /c:     3000
a:     65   p:     42   n:     24   b /c:     3696
a:     65   p:     52   n:     26   b /c:     4056
a:     65   p:     78   n:     26   b /c:     4056
a:     65   p:     88   n:     24   b /c:     3696
a:     65   p:    100   n:     20   b /c:     3000
a:     65   p:    112   n:     14   b /c:     2016
```

Otro detalle, si os fijáis (n) siempre crece, por tanto para mejorar el rendimiento no tenemos porque volver con (n) al principio, en cuanto encontremos una diagonal, podemos continuar con (n) en el valor que se nos quedó.

Volvamos al código y mejorémoslo con las observaciones anteriores.

gen_p_n_4.py

Bien hemos mejorado, la salida y el rendimiento.

8.- Lanzar la búsqueda de la diagonal a por uno de esos números que conocemos bien los informáticos: los binarios o múltiplos de 2.

Este apartado es muy corto, pero para mi muy significativo, pues a partir de aquí empiezan a aparecer cosas, desde mi punto de vista sorprendentes para los números primos.
Como indica el título, en mis enredos quise calcular las diagonales de los siguientes números: [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024], nuestro código lo he modificado para este experimento.

gen_p_n_5.py

¿Cual ha sido su salida?….NADA, ¿ que pasa con los múltiplos solo de dos, son incapaces de darnos una diagonal...?

9.- Lanzar la búsqueda de la diagonal contra una serie de números primos, observar que unos son capaces de darnos una diagonal, otros NO. ¿ Esta serie de unos y ceros es completamente aleatoria?.

Necesitamos una función que nos genere números primos para nuestro ensayo. He recorrido por Internet y en esta web, donde está alojada esta documentación, hemos localizado una función en: https://gist.github.com/categulario/2772442 de un tal Categulario (desde aquí le doy las gracias) que usaremos para generar números primos. Esta función nos generara los primos que hay a un número dado. Os adjunto el nuevo código.

gen_p_n_6.py

Bien, pero esto no es lo que nos interesa, de esta serie de primos hasta 1000, nos interesa saber que números primos son incapaces de darnos ninguna diagonal y de los que sí ¿cuántas?. Vuelvo a modificar el código y lo presento:

gen_p_n_7.py

Tenéis la salida en el archivo: < primos_1-1000.txt >, ya sabemos que primos son singulares porque al menos generan una diagonal y cuales no.
Y ahora la pregunta, sé que solo son hasta el 1000, pero ¿ la serie de unos y ceros de la columna de la derecha es una serie aleatoria o pseudoaleatoria ?

Nota: Posteriormente la obtuve la salida de 64.000. La tenéis en < **primos_1-64000.txt** >

10.- Ver que un número formado por primos no capaces, seguirá sin darnos una diagonal.

Ya lo vimos para la serie de múltiplos de dos, veamos algunos más. Me diréis que probar unos cuantos más no es significativo para deducir que nunca generaran diagonales, no obstante ahí lo dejo.

Pero antes de seguir, seguramente lo que vamos a pedir puede tardar mucho, como ya dijimos al principio, este lenguaje no es muy eficiente para procesos de bucles singulares, así que determinadas partes, las de más carga las recodificaremos con cython, un pseudo lenguaje python + C que nos va a permitir crear módulos que almacenara nuestra función en lenguaje C mucho más eficiente en tiempo de ejecución.

Modulo en cython: **primos_sn.pyx** una vez compilado obtendremos un modulo en C al que podremos llamar desde python : **primos_sn.so**

Nuestro código en python que llamará al modulo generado anteriormente. Analizaremos:
El 3 hasta su quinta potencia.
El 7 hasta su quinta potencia.
El 11 hasta su quinta potencia.
El 19 hasta su cuarta potencia.
El 23 hasta su cuarta potencia.

Comentaros que aunque hallamos optimizado el código con un modulo en C, para mi equipo bastante modesto, tardará.
gen_p_n_8.py

Su salida:

```
         3  0
         9  0
        27  0
        81  0
       243  0
         7  0
        49  0
       343  0
      2401  0
     16807  0
        11  0
       121  0
      1331  0
     14641  0
    161051  0
        19  0
       361  0
      6859  0
    130321  0
        23  0
       529  0
     12167  0
    279841  0
```

Hasta este punto NADA, más adelante con potencias superiores, ¿alguien puede responder?.

Veamos ahora que un número formado por diferentes primos incapaces no nos dará ninguna diagonal. Tomemos los siguientes

```
3 * 7 * 11 * 23 * 31
```

gen_p_n_9.py

Su salida:
164703 0 (también cero)

Planteo la siguiente pregunta, lo visto en este apartado es extrapolable para todos los números formados por primos incapaces ...?

11.- Porqué determinados primos nos dan tantas diagonales como múltiplos de si mismos tengan.

Como el código lo hemos reducido considerablemente, no considero necesario poner este, con cualquiera de los módulos anteriores podéis generar lo que estamos preguntándonos. Diferentes salidas del 5 y sus potencia y observamos que se mantiene lo comentado:

```
     3125  5
    15625  6
    78125  7
   390625  8
  1953125  9
```

Comentaros que los tiempos de ejecución aumentan considerablemente conforme va creciendo el numero.
Con el número 13 también se mantiene.

```
   169  2
  2197  3
 28561  4
371293  5
```

Con el número 17 ocurre los mismo,

```
   289  2
  4913  3
 83521  4
```

Veis que trato de no pasar de números por encima de 1500000, los tiempos de ejecución se alargan considerablemente.

Los números primos que he ido probando hasta ahora siguen este comportamiento.

Volvemos con la misma consideración del apartado anterior, este comportamiento es extrapolable hacia todos los primos capaces de darnos diagonales, y sus potencias nos dan diagonales en función de las misma ….?

12.- Ver que un número formado por primos capaces de generar diagonales, si lo modificamos con primos incapaces, el número de diagonales no varían. Sus valores si, (a,p,n) es como si los multiplicásemos por el/los primos incapaces .

De igual manera que en el apartado anterior con el ultimo código < **gen_p_n_9.py** > es sencillo ver si es cierto esta afirmación.

Yo lo probado con el 5 y los siguientes no capaces

5*3*7*11*19*23

Resultado:

```
504735  1
```

De igual manera que en los apartados anteriores podemos hacernos la pregunta: realmente esta afirmación es valida para cualesquiera de los números que cumplan estas condiciones ?

Veamos ahora el desplazamiento, para ello hemos creado un función con cython, que nos muestre los valores de la/s diagonal/es.

Dgnl.pyx

Y módulo en python que le llamará para mostranos los valores de la/s diagonal/es.

dia_fast.py

Os muestro el código, realmente sencillo:

```
import dgnl

l_num = [ 5*2*3*7*11*19 ]

for i in l_num:
    dgnl.n_dgnl(i)
```

Lo lanzaremos como está y después solo con el cinco y compararemos resultados:

Para 5*2*3*7*11*19:
```
a:    43890 p:    35112 n:    17556  b ó c:   1849278816  K:   5778996300
```

Para 5 :
```
a:        5 p:        4 n:        2  b ó c:           24  K:           75
```

p(43890) = p(5) * 2*3*7*11*19
35112 = 4 * 2*3*7*11*19

de igual manera,

n(43890) = n(5) * 2*3*7*11*19
17556 = 2 * 2*3*7*11*19

13.- La combinación de primos capaces, hasta cuanto aumenta el número de diagonales. Es posible determinarla

13.1 .- ¿ La combinación (producto) de 2 primos capaces siempre genera 4 diagonales ?

Os ajunto mi prueba y su salida:

El código:

```
import primos_sn

l_num = [5*13, 5*17, 5*29,  5*37, 5*41, 5*53, 5*61, 5*73,
         13*17, 13*29,  13*37, 13*41, 13*53, 13*61, 13*73,
         17*29,  17*37, 17*41, 17*53, 17*61, 17*73,
         29*37, 29*41, 29*53, 29*61, 29*73,
         37*41, 37*53, 37*61, 37*73,
         41*53, 41*61, 41*73,
         53*61, 53*73,
         61*73 ]

for i in l_num:
    primos_sn.max_n_p(i)
```

Su salida:

```
  65  4
  85  4
 145  4
 185  4
 205  4
 265  4
 305  4
 365  4
 221  4
 377  4
 481  4
 533  4
 689  4
 793  4
 949  4
 493  4
 629  4
 697  4
 901  4
1037  4
1241  4
1073  4
1189  4
1537  4
1769  4
2117  4
1517  4
1961  4
2257  4
2701  4
2173  4
2501  4
2993  4
3233  4
3869  4
4453  4
```

Vaya, hasta aquí se cumple y más allá ?

13.2 .- ¿ La combinación (producto) de 3 primos capaces siempre genera 13 diagonales ?

Os ajunto mi prueba y su salida:

El código:

```
import primos_sn

l_num = [ 5 *13* 17,
          5 *13* 29,
          5 *13* 37,
          5 *13* 41,
          5 *13*53,
          5 *13* 61,
          5 *13* 73,
          5 *17* 29,
          5 *17* 37,
          5 *17* 41,
          5 *17* 53,
          5 *17* 61,
```

```
          5 *17* 73,
          5 *29* 37,
          5 *29* 41,
          5 *29* 53,
          5 *29* 61,
          5 *29* 73,
          5 *37* 41,
          5 *37* 53,
          5 *37* 61,
          5 *37* 73,
          5 *41* 53,
          5 *41* 61,
          5 *41* 73,
          5 *53* 61,
          5 *53* 73,
          5 *61* 73 ]

for i in l_num:
    primos_sn.max_n_p(i)
```

Su salida:

```
 1105 13
 1885 13
 2405 13
 2665 13
 3445 13
 3965 13
 4745 13
 2465 13
 3145 13
 3485 13
 4505 13
 5185 13
 6205 13
 5365 13
 5945 13
 7685 13
 8845 13
10585 13
 7585 13
 9805 13
11285 13
13505 13
10865 13
12505 13
14965 13
16165 13
19345 13
22265 13
```

Vaya, hasta aquí se cumple y más allá ?

13.3 .- ¿ La combinación (producto) de 4 primos capaces siempre genera 40 diagonales ?

Os ajunto mi prueba y su salida:

El código:

```
import primos_sn

l_num = [5* 13* 17* 29,
             5* 13* 17* 37,
             5* 13* 17* 41,
             5* 13* 17* 53,
             5* 13* 17* 61,
             5* 13* 17* 73,
             5* 13* 29* 37,
             13* 17* 29* 73,
             13* 17* 37* 41,
             13* 17* 37* 53,
             13* 17* 37* 61,
             13* 17* 37* 73,
             13* 17* 41* 53,
             13* 17* 41* 61,
             13* 17* 41* 73,
             13* 17* 53* 61,
             13* 17* 53* 73,
             13* 17* 29* 73,
             13* 17* 37* 41,
             13* 17* 37* 53,
             13* 17* 37* 61,
             13* 17* 37* 73,
             13* 17* 41* 53,
             13* 17* 41* 61,
             13* 17* 41* 73,
             13* 17* 53* 61,
             13* 17* 53* 73 ]

for i in l_num:
    primos_sn.max_n_p(i)
```

Su salida (en mi equipo se ha tirado un buen rato):

```
   32045 40
   40885 40
   45305 40
   58565 40
   67405 40
   80665 40
   69745 40
  467857 40
  335257 40
  433381 40
  498797 40
  596921 40
  480233 40
  552721 40
  661453 40
  714493 40
  855049 40
  467857 40
  335257 40
  433381 40
  498797 40
  596921 40
  480233 40
  552721 40
  661453 40
  714493 40
```

```
855049 40
```

De igual manera hasta aquí se cumple, y más allá ?

13.4 .- ¿ La combinación (producto) de 5 primos capaces siempre genera 121 diagonales ?

Os ajunto mi prueba y su salida:

El código:

```
import primos_sn


l_num =  [5* 13* 17* 29* 37,
          5* 13* 17* 29* 41,
          5* 13* 17* 29* 53 ]

for i in l_num:
    primos_sn.max_n_p(i)
```

Su salida (en mi equipo se ha tirado un buen, buen... rato):

```
1185665 121
1313845 121
1698385 121
```

Hasta aquí se cumple, y más allá ?

NOTA: Ayer lance el proceso para 6 primos capaces (5 * 13 * 17 * 29 * 37 * 41) = 48.612.265, número excesivamente grande para mi equipo, tuve que abortar el proceso después de varias horas.

14.- Hasta aquí y mi CONCLUSIÓN:

Vamos a estudiar brevemente el espacio de los números, en nuestro caso aquellos con más probabilidades de generar Diagonales/Filas/Columnas (DFC).

DFC de 65 = (5*13): 4

```
a:      65 p:      18 n:      14  b ó c:      2016  K:      12675
a:      65 p:      30 n:      20  b ó c:      3000  K:      12675
a:      65 p:      42 n:      24  b ó c:      3696  K:      12675
a:      65 p:      52 n:      26  b ó c:      4056  K:      12675
```

Con cuatro que es capaz de generar, presenta el mínimo que necesitamos. No obstante, las condiciones que tienen que cumplir son simples pero difíciles de alcanzar. Os repito nuestro cuadro:

$a^2 + b$	$a^2 - (b + c)$	$a^2 + c$
$a^2 - (b - c)$	a^2	$a^2 + (b - c)$
$a^2 - c$	$a^2 + (b + c)$	$a^2 - b$

Podríamos cubrir perfectamente el cuadro y cumplir 4 de las 8 condiciones, pero no mas.

DFC de 1.105 = (5*13*17): 13

```
a:      1105 p:       48 n:       46  b ó c:      103776  K:      3663075
a:      1105 p:      110 n:      100  b ó c:      231000  K:      3663075
a:      1105 p:      182 n:      156  b ó c:      369096  K:      3663075
a:      1105 p:      296 n:      232  b ó c:      566544  K:      3663075
a:      1105 p:      306 n:      238  b ó c:      582624  K:      3663075
a:      1105 p:      510 n:      340  b ó c:      867000  K:      3663075
a:      1105 p:      572 n:      364  b ó c:      936936  K:      3663075
a:      1105 p:      650 n:      390  b ó c:     1014000  K:      3663075
a:      1105 p:      714 n:      408  b ó c:     1068144  K:      3663075
a:      1105 p:      738 n:      414  b ó c:     1086336  K:      3663075
a:      1105 p:      884 n:      442  b ó c:     1172184  K:      3663075
a:      1105 p:      950 n:      450  b ó c:     1197000  K:      3663075
a:      1105 p:     1032 n:      456  b ó c:     1215696  K:      3663075
```

Con 13 estamos en mejores condiciones, podemos cubrir las dos diagonales, y para el resto hemos de combinar pares de (b y c):
+ (b-c) con - (b-c)
- (b+c) con + (b+c)
¿ Dentro de que rango ? … fijaros el rango que tengo para conseguirlo va desde el menor valor de (b ó c): 103.776, al mayor de (b ó c): 1.215.696, encontrándose el rango en la diferencia entre el mayor y menor, en nuestro caso: 1.111.920

Tengo 13 DFC (posibilidades) en un rango de 1.111.920 números

DFC de 32.045 = (5*13*17*29): 40

```
a:     32045 p:      724 n:      708  b ó c:     45876984  K:    3080646075
a:     32045 p:     1392 n:     1334  b ó c:     87275616  K:    3080646075
a:     32045 p:     2358 n:     2196  b ó c:    145564056  K:    3080646075
a:     32045 p:     2490 n:     2310  b ó c:    153384000  K:    3080646075
a:     32045 p:     3190 n:     2900  b ó c:    194271000  K:    3080646075
a:     32045 p:     3978 n:     3536  b ó c:    239125536  K:    3080646075
a:     32045 p:     4200 n:     3710  b ó c:    251538000  K:    3080646075
a:     32045 p:     5278 n:     4524  b ó c:    310409736  K:    3080646075
a:     32045 p:     5746 n:     4862  b ó c:    335244624  K:    3080646075
a:     32045 p:     7486 n:     6042  b ó c:    423737544  K:    3080646075
a:     32045 p:     8092 n:     6426  b ó c:    453135816  K:    3080646075
a:     32045 p:     8584 n:     6728  b ó c:    476463504  K:    3080646075
a:     32045 p:     8874 n:     6902  b ó c:    489986784  K:    3080646075
a:     32045 p:     9372 n:     7194  b ó c:    512817096  K:    3080646075
a:     32045 p:     9750 n:     7410  b ó c:    529815000  K:    3080646075
a:     32045 p:     9996 n:     7548  b ó c:    540723624  K:    3080646075
a:     32045 p:    11700 n:     8450  b ó c:    612963000  K:    3080646075
a:     32045 p:    11856 n:     8528  b ó c:    619286304  K:    3080646075
a:     32045 p:    13600 n:     9350  b ó c:    686664000  K:    3080646075
a:     32045 p:    13858 n:     9464  b ó c:    696115056  K:    3080646075
a:     32045 p:    14790 n:     9860  b ó c:    729147000  K:    3080646075
a:     32045 p:    15640 n:    10200  b ó c:    757758000  K:    3080646075
a:     32045 p:    16588 n:    10556  b ó c:    787963176  K:    3080646075
a:     32045 p:    18850 n:    11310  b ó c:    852774000  K:    3080646075
a:     32045 p:    19822 n:    11594  b ó c:    877480296  K:    3080646075
a:     32045 p:    20148 n:    11684  b ó c:    885343416  K:    3080646075
a:     32045 p:    20706 n:    11832  b ó c:    898309104  K:    3080646075
a:     32045 p:    21402 n:    12006  b ó c:    913608576  K:    3080646075
a:     32045 p:    21964 n:    12138  b ó c:    925255464  K:    3080646075
```

```
a:    32045 p:    22294 n:    12212  b ó c:    931800024  K:   3080646075
a:    32045 p:    24024 n:    12558  b ó c:    962545584  K:   3080646075
a:    32045 p:    25636 n:    12818  b ó c:    985806744  K:   3080646075
a:    32045 p:    26208 n:    12896  b ó c:    992811456  K:   3080646075
a:    32045 p:    26640 n:    12950  b ó c:    997668000  K:   3080646075
a:    32045 p:    27550 n:    13050  b ó c:   1006677000  K:   3080646075
a:    32045 p:    28840 n:    13160  b ó c:   1016610000  K:   3080646075
a:    32045 p:    29014 n:    13172  b ó c:   1017695064  K:   3080646075
a:    32045 p:    29928 n:    13224  b ó c:   1022400336  K:   3080646075
a:    32045 p:    30940 n:    13260  b ó c:   1025661000  K:   3080646075
a:    32045 p:    31222 n:    13266  b ó c:   1026204696  K:   3080646075
```

Con 40 DFC estamos en mejores condiciones, veamos nuestras posibilidades:

Tengo 40 DFC (posibilidades) en un rango de 980.327.712 números

DFC de 1.185.665 = (5*13*17*29*37): 121

```
a:  1185665 p:     7878 n:     7826  b ó c:  18619274856  K: 4217404476675
a:  1185665 p:    12480 n:    12350  b ó c:  29438448000  K: 4217404476675
a:  1185665 p:    26788 n:    26196  b ó c:  62805591096  K: 4217404476675
a:  1185665 p:    43350 n:    41820  b ó c: 100917933000  K: 4217404476675
a:  1185665 p:    51012 n:    48906  b ó c: 118364061816  K: 4217404476675
a:  1185665 p:    51504 n:    49358  b ó c: 119480318304  K: 4217404476675
a:  1185665 p:    64498 n:    61166  b ó c: 148786050336  K: 4217404476675
a:  1185665 p:    72280 n:    68120  b ó c: 166175334000  K: 4217404476675
a:  1185665 p:    78880 n:    73950  b ó c: 180828456000  K: 4217404476675
a:  1185665 p:    84376 n:    78758  b ó c: 192964030704  K: 4217404476675
a:  1185665 p:    87246 n:    81252  b ó c: 199277192664  K: 4217404476675
a:  1185665 p:    92130 n:    85470  b ó c: 209982696000  K: 4217404476675
a:  1185665 p:   100572 n:    92684  b ó c: 228374673576  K: 4217404476675
a:  1185665 p:   104550 n:    96050  b ó c: 236991849000  K: 4217404476675
a:  1185665 p:   110142 n:   100746  b ó c: 249051768696  K: 4217404476675
a:  1185665 p:   118030 n:   107300  b ó c: 265956999000  K: 4217404476675
a:  1185665 p:   126616 n:   114342  b ó c: 284216707824  K: 4217404476675
a:  1185665 p:   133458 n:   119886  b ó c: 298661921376  K: 4217404476675
a:  1185665 p:   147186 n:   130832  b ó c: 327362858784  K: 4217404476675
a:  1185665 p:   147328 n:   130944  b ó c: 327657766656  K: 4217404476675
a:  1185665 p:   155400 n:   137270  b ó c: 344355522000  K: 4217404476675
a:  1185665 p:   195286 n:   167388  b ó c: 424950928584  K: 4217404476675
a:  1185665 p:   203580 n:   173420  b ó c: 441310545000  K: 4217404476675
a:  1185665 p:   209508 n:   177684  b ó c: 452919003576  K: 4217404476675
a:  1185665 p:   212602 n:   179894  b ó c: 458949890256  K: 4217404476675
a:  1185665 p:   226954 n:   190008  b ó c: 486674710704  K: 4217404476675
a:  1185665 p:   234192 n:   195024  b ó c: 500500622496  K: 4217404476675
a:  1185665 p:   244522 n:   202086  b ó c: 520051345776  K: 4217404476675
a:  1185665 p:   262276 n:   213962  b ó c: 553154246904  K: 4217404476675
a:  1185665 p:   268380 n:   217970  b ó c: 564389721000  K: 4217404476675
a:  1185665 p:   276982 n:   223554  b ó c: 580096697736  K: 4217404476675
a:  1185665 p:   277134 n:   223652  b ó c: 580372914264  K: 4217404476675
a:  1185665 p:   284548 n:   228404  b ó c: 593789644536  K: 4217404476675
a:  1185665 p:   299404 n:   237762  b ó c: 620342932104  K: 4217404476675
a:  1185665 p:   302634 n:   239768  b ó c: 626057745264  K: 4217404476675
a:  1185665 p:   308850 n:   243600  b ó c: 636996948000  K: 4217404476675
a:  1185665 p:   313296 n:   246318  b ó c: 644773820064  K: 4217404476675
a:  1185665 p:   317608 n:   248936  b ó c: 652278536976  K: 4217404476675
a:  1185665 p:   328338 n:   255374  b ó c: 670791907296  K: 4217404476675
a:  1185665 p:   337900 n:   261020  b ó c: 687095997000  K: 4217404476675
a:  1185665 p:   343474 n:   264272  b ó c: 696515811744  K: 4217404476675
a:  1185665 p:   346764 n:   266178  b ó c: 702046604424  K: 4217404476675
a:  1185665 p:   360750 n:   274170  b ó c: 725316735000  K: 4217404476675
a:  1185665 p:   369694 n:   279188  b ó c: 739992819384  K: 4217404476675
a:  1185665 p:   369852 n:   279276  b ó c: 740250641256  K: 4217404476675
a:  1185665 p:   385236 n:   287738  b ó c: 765114908184  K: 4217404476675
```

```
a:  1185665 p:   415162 n:   303606  b ó c: 812126619216  K: 4217404476675
a:  1185665 p:   422994 n:   307632  b ó c: 824134437984  K: 4217404476675
a:  1185665 p:   432900 n:   312650  b ó c: 839146347000  K: 4217404476675
a:  1185665 p:   438672 n:   315536  b ó c: 847802950176  K: 4217404476675
a:  1185665 p:   442078 n:   317226  b ó c: 852879865656  K: 4217404476675
a:  1185665 p:   448630 n:   320450  b ó c: 862580901000  K: 4217404476675
a:  1185665 p:   456550 n:   324300  b ó c: 874192809000  K: 4217404476675
a:  1185665 p:   482466 n:   336542  b ó c: 911312658624  K: 4217404476675
a:  1185665 p:   487200 n:   338720  b ó c: 917948136000  K: 4217404476675
a:  1185665 p:   503200 n:   345950  b ó c: 940043016000  K: 4217404476675
a:  1185665 p:   512746 n:   350168  b ó c: 952981511664  K: 4217404476675
a:  1185665 p:   513358 n:   350436  b ó c: 953804789976  K: 4217404476675
a:  1185665 p:   531030 n:   358050  b ó c: 977254509000  K: 4217404476675
a:  1185665 p:   536976 n:   360558  b ó c: 985004073504  K: 4217404476675
a:  1185665 p:   547230 n:   364820  b ó c: 998202243000  K: 4217404476675
a:  1185665 p:   557512 n:   369014  b ó c: 1011225300816  K: 4217404476675
a:  1185665 p:   563500 n:   371420  b ó c: 1018712205000  K: 4217404476675
a:  1185665 p:   568342 n:   373346  b ó c: 1024713805896  K: 4217404476675
a:  1185665 p:   578680 n:   377400  b ó c: 1037370702000  K: 4217404476675
a:  1185665 p:   595080 n:   383670  b ó c: 1057010850000  K: 4217404476675
a:  1185665 p:   603330 n:   386750  b ó c: 1066687440000  K: 4217404476675
a:  1185665 p:   613128 n:   390344  b ó c: 1078002875856  K: 4217404476675
a:  1185665 p:   613756 n:   390572  b ó c: 1078721587944  K: 4217404476675
a:  1185665 p:   630292 n:   396474  b ó c: 1097362323096  K: 4217404476675
a:  1185665 p:   640150 n:   399900  b ó c: 1108214877000  K: 4217404476675
a:  1185665 p:   670738 n:   410096  b ó c: 1140651676896  K: 4217404476675
a:  1185665 p:   680680 n:   413270  b ó c: 1150791642000  K: 4217404476675
a:  1185665 p:   687648 n:   415454  b ó c: 1157780559936  K: 4217404476675
a:  1185665 p:   697450 n:   418470  b ó c: 1167447606000  K: 4217404476675
a:  1185665 p:   708084 n:   421668  b ó c: 1177717880664  K: 4217404476675
a:  1185665 p:   716532 n:   424154  b ó c: 1185715720536  K: 4217404476675
a:  1185665 p:   733414 n:   428978  b ó c: 1201270525224  K: 4217404476675
a:  1185665 p:   745476 n:   432308  b ó c: 1212035136504  K: 4217404476675
a:  1185665 p:   749146 n:   433302  b ó c: 1215252654864  K: 4217404476675
a:  1185665 p:   755380 n:   434970  b ó c: 1220656311000  K: 4217404476675
a:  1185665 p:   766122 n:   437784  b ó c: 1229785163376  K: 4217404476675
a:  1185665 p:   783144 n:   442088  b ó c: 1243778336784  K: 4217404476675
a:  1185665 p:   791874 n:   444222  b ó c: 1250730140544  K: 4217404476675
a:  1185665 p:   801850 n:   446600  b ó c: 1258487538000  K: 4217404476675
a:  1185665 p:   808962 n:   448256  b ó c: 1263896342016  K: 4217404476675
a:  1185665 p:   812668 n:   449106  b ó c: 1266674730216  K: 4217404476675
a:  1185665 p:   824878 n:   451844  b ó c: 1275634232856  K: 4217404476675
a:  1185665 p:   829806 n:   452922  b ó c: 1279165864344  K: 4217404476675
a:  1185665 p:   834900 n:   454020  b ó c: 1282765407000  K: 4217404476675
a:  1185665 p:   842044 n:   455532  b ó c: 1287726100584  K: 4217404476675
a:  1185665 p:   850668 n:   457314  b ó c: 1293578502216  K: 4217404476675
a:  1185665 p:   862978 n:   459776  b ó c: 1301674592256  K: 4217404476675
a:  1185665 p:   871624 n:   461448  b ó c: 1307179742544  K: 4217404476675
a:  1185665 p:   878800 n:   462800  b ó c: 1311635364000  K: 4217404476675
a:  1185665 p:   888888 n:   464646  b ó c: 1317724904496  K: 4217404476675
a:  1185665 p:   889066 n:   464678  b ó c: 1317830525424  K: 4217404476675
a:  1185665 p:   931164 n:   471692  b ó c: 1341030733224  K: 4217404476675
a:  1185665 p:   948532 n:   474266  b ó c: 1349569432536  K: 4217404476675
a:  1185665 p:   959530 n:   475800  b ó c: 1354664454000  K: 4217404476675
a:  1185665 p:   965926 n:   476658  b ó c: 1357516264104  K: 4217404476675
a:  1185665 p:   969696 n:   477152  b ó c: 1359158883264  K: 4217404476675
a:  1185665 p:   985680 n:   479150  b ó c: 1365807492000  K: 4217404476675
a:  1185665 p:   995866 n:   480342  b ó c: 1369777831824  K: 4217404476675
a:  1185665 p:  1008304 n:   481712  b ó c: 1374344567904  K: 4217404476675
a:  1185665 p:  1019350 n:   482850  b ó c: 1378140813000  K: 4217404476675
a:  1185665 p:  1029558 n:   483836  b ó c: 1381432096776  K: 4217404476675
a:  1185665 p:  1036830 n:   484500  b ó c: 1383649635000  K: 4217404476675
a:  1185665 p:  1047228 n:   485394  b ó c: 1386636689256  K: 4217404476675
a:  1185665 p:  1056006 n:   486098  b ó c: 1388990035944  K: 4217404476675
a:  1185665 p:  1067080 n:   486920  b ó c: 1391739090000  K: 4217404476675
a:  1185665 p:  1073518 n:   487364  b ó c: 1393224542616  K: 4217404476675
a:  1185665 p:  1077312 n:   487614  b ó c: 1394061119616  K: 4217404476675
a:  1185665 p:  1084600 n:   488070  b ó c: 1395587358000  K: 4217404476675
a:  1185665 p:  1089802 n:   488376  b ó c: 1396611777456  K: 4217404476675
a:  1185665 p:  1107336 n:   489288  b ó c: 1399666059984  K: 4217404476675
```

```
a:  1185665 p:  1118430 n:   489770  b ó c: 1401280947000  K: 4217404476675
a:  1185665 p:  1127230 n:   490100  b ó c: 1402386843000  K: 4217404476675
a:  1185665 p:  1144780 n:   490620  b ó c: 1404129909000  K: 4217404476675
a:  1185665 p:  1155214 n:   490842  b ó c: 1404874228824  K: 4217404476675
a:  1185665 p:  1155882 n:   490854  b ó c: 1404914465136  K: 4217404476675
```

Vaya, ahora puedo jugar con 121 DFC, estamos en mejores condiciones que antes, veamos nuestras posibilidades:
Un inciso, la calculadora de mi ordenador no puede (1,40491446514e+12), tengo que hacerlo a mano pues al final me ha redondeado.

Tengo 121 DFC (posibilidades) en un rango de 557.111.514.960 números, uff... más de medio billón de números.

Mi humilde conclusión:

Resolver el Cuadro Mágico de Edouart Lucas es como que te regalen cuantos números de lotería quieras, pero cuantos más quieras, los números contra los que tengas que jugar, te los hagan crecer también pero de forma brutal.

Getafe 12 de enero de 2020.