OBJETIVO:

Resolver Sudokus cuadrados de cualquier dimensión. Si están bien planteados una única solución, si no es así, las que tuviera, ojo, podemos plantear un Sudoku vacío, no os lo aconsejo para dimensiones mayores de 4x4 (por cierto 288 soluciones), podríais colapsar vuestro equipo agotando la memoria disponible. Os comento algunos tipos de Sudokus cuadrados, los elementos posibles son los que propongo, podréis variarlos desde el fichero de configuración que comentaremos más adelante:

Dimensión: 4x4
Cadena base de elementos posibles: "1234"
Otra cadena: “ABCD”
Matriz de trabajo de 16 casillas

Dimensión: 9x9 (Sodoku clásico)
Cadena base de elementos posibles: "123456789"
Matriz de trabajo de 81 casillas

Dimensión: 16x16 (Mega Sodoku)
Cadena base de elementos posibles: "123456789ABCDFG"
Matriz de trabajo de 256 casillas

Dimensión: 25x25 (Alphadoku)
Cadena base de elementos posibles: "123456789ABCDEFGHIJKLMNOP"
Matriz de trabajo de 625 casillas

Dimensión: 36x36
Cadena base de elementos posibles: "123456789ABCDEFGHIJKLMNOPQRSTUVWXYZa"
Matriz de trabajo de 1296 casillas

Dimensión: 49x49
Cadena base de elementos posibles:
"123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmn"
Matriz de trabajo de 2401 casillas

NOTA: Los elementos de nuestros Sudokus están representados por un solo signo (un carácter). He encontrado Sudokus de 16x16, en que los elementos se representan mediante 16 números, en cuanto superamos el nueve ya tenemos 2 signos para los siguientes elementos. Si os apetece que lo resuelva el programa debéis convertirlo a un solo signo.

Este **reto** está resuelto personalmente en 2006 con ALASKA-SW (con 2 estrategias y Fuerza Bruta), mi intención es pasarlo a Python y con eso aprender este lenguaje (actualmente soy un neófito). La tercera estrategia que he implementado en la versión 4, nunca estuvo en ALASKA-SW.

Conceptos:

Se entiende que se conoce el paradigma del Sudoku y los conceptos de filas, columnas y cuadros/regiones. En la dirección de Dialnet que os marco podréis localizar el artículo de Abr-2016:

Dialnet-CuantaMatematicaHayEnLosSudokus-5998394.pdf

Os copio el Resumen (https://dialnet.unirioja.es/servlet/articulo?codigo=5998394)
Este artículo trata de realizar una descripción lo más completa posible del juego del Sudoku. Se comentan sus principales características, las estrategias de resolución, sus niveles de dificultad, sus múltiples variantes y, sobre todo, el fundamento matemático que subyace en el mismo. El principal objetivo que se persigue es mostrar al ciudadano normal, sin preparación específica en Matemáticas, que este juego es muy interesante, divertido y apto para ser considerado por cualquiera, a pesar de tratarse de un juego en el que "se incluyen números", con las connotaciones negativas que ese hecho supone para la mayor parte de las personas.

Haremos referencias a este artículo para entender que son las estrategias de resolución y algunos otros conceptos, nada de matemáticas.

Las Versiones:

Son las diferentes progresiones de este SW que se construyeron para solucionar el reto.

No esperéis un entorno GUI, se trata de programación estructurada apoyada con funciones en entorno de consola. Doy por supuesto que se tienen conocimientos de python, qué es una consola y como se lanza un programa en python desde la misma.

SDK_v01.py
Versión 01
Trata de resolver un Sudoku con 2 estrategias y no está implementado el módulo de Fuerza Bruta, además los valores de configuración del Sudoku están cargados en el propio programa, en este caso un Sudoku de 9 x 9. El planteamiento de resolución es independiente del tipo de Sudoku. Es capaz de solucionar algunos Sudokus en los que su nivel de dificultad no sea alto.

SDK_v02.py
Versión 02
Esta versión amplia la anterior implementando parte de la configuración a través del fichero <sdk.ini> y el módulo de resolución por Fuerza Bruta. Esta versión es Plenamente Operativa.

SDK_v03.py
Versión 03
En esta versión se pondrán en marcha todos los elementos del fichero de configuración <sdk.ini> y se almacenarán sus resultados en fichero. Esta versión es Completa y Plenamente Operativa.

SDK_v04.py
Versión 04

Esta versión amplia la versión anterior implementado una nueva estrategia de solución antes de pasar a Fuerza Bruta. Estrategia PAR DESNUDO o DOBLE PAREJA.

El planteamiento para solucionar el reto

En líneas generales se trata de agotar las estrategias de solución antes de pasar a Fuerza Bruta:
(Las explicaciones se harán sobre un Sudoku de 9x9)

1º Estrategia

Personalmente la llamé “SIMPLE” en 2006. En el artículo de Dialnet se hace referencia a ella en el apartado 4º (Métodos de Resolución del juego del Sudoku) con el nombre de “Único Desnudo”, nos da una breve descripción del mismo. Os la amplio a continuación:

Sobre una casilla, agrupar los elementos de fila, columna y cuadro, Ver cuantos faltan de la cadena base de elementos posibles. Si sólo nos falta uno, es el definitivo para esa casilla y si hubiera más de uno, serían candidatos a la casilla.

Os presento un Sudoku fácil de la página:
http://www.Sudokumania.com.ar/juegos/Sudoku , en donde aplicar esta estrategia.

	A	B	C	D	E	F	G	H	I
1			8		6			3	5
2		5		8		3	9		
3				1	5	9	7		
4					3	2		6	
5	4	3						7	
6	5		9		4		2		
7	3	7		9	1			4	
8				4	2				
9		4	5	3		7	6	9	

Veamos la casilla 9E, tomemos todos los elemento de su fila, columna y cuadro/región asociados:
Fila 9 → 453769
Columna E → 653412
Cuadro 8 → 914237
A continuación vayamos repasando sobre la cadena base de elementos posibles, en nuestro caso 123456789 y si de la unión de los elementos de Fila, Columna y Cuadro/Región falta alguno en la de cadena base.
Tenemos el 1 →Si
Tenemos el 2 → Si

Tenemos el 3 → Si … y así sucesivamente donde llegaremos al 8 y veremos que la respuesta es NO, y al 9 en que la respuesta es Sí
Hemos localizado una casilla con un único valor y por tanto se podrá asignar como Único.

2º Estrategia
Personalmente la llamé "SOLO PUEDE SER ESE" en 2006. En el artículo de Dialnet se hace referencia a ella en el apartado 4º (Métodos de Resolución del juego del Sudoku) con el nombre de "Única posición o Único Oculto", dándonos una descripción de la aplicación del mismo.

Os la amplío con otro ejemplo:

Esta estrategia tratará de buscar los elementos que faltan en el Sudoku de la siguiente manera:
Se basa en un estudio de filas, columnas o cuadros. Si al examinar en una de estas, la agrupación de candidatos de las casillas sin resolver, vemos que sólo se repite uno una vez, ese valor es el único posible para la casilla en donde se encuentra.

Ejemplo: Dada la columna 4º, sean las siguientes las casillas sin resolver
casilla[1,4]:='3456' candidatos posibles
casilla[6,4]:='36' candidatos posibles
casilla[7,4]:='46' candidatos posibles
casilla[9,4]:='34' candidatos posibles

Vemos que el valor 5 no está repetido en ninguna casilla más, por tanto, el candidato 5 es el único posible de la casilla [1,4].

3º Estrategia
Esta estrategia se implementó en Oct-2019. En el artículo de Dialnet se hace referencia a ella en el apartado 4º (Métodos de Resolución del juego del Sudoku) con el nombre de "Par Desnudo o Doble Pareja", donde se nos da una descripción amplia de la aplicación del mismo. No considero necesario ampliarla.

Fuerza Bruta (Recorrido Arbóreo)
Personalmente en el artículo de Dialnet se hace referencia a ella en la página 9 del pdf (120 / Revista de Pensamiento Matemático) con el nombre de *Técnicas de adivinación (Nisio, por ejemplo)*, pero no la describe.

Os traigo desde la página: http://www.playSudoku.biz/Nishio.aspx la descripción de dicho método:

" La técnica conocida como Nishio consiste en encontrar una celda que sólo tenga dos candidatos y quitar uno de los dos. Trabajando a partir de este punto, tendremos que comprobar que nos queda un tablero válido y con una única solución. Si es así querrá decir que hemos elegido correctamente. Si

nuestra elección fue la incorrecta, puede que nos demos cuenta con un par de movimientos o puede que nos demos cuenta al final, cuando lleguemos a un callejón sin salida.

Mucha gente sólo probará unos cuantos movimientos antes de eliminar el candidato y probar otro, por temas de rapidez.

Si encontramos una incompatibilidad de forma rápida, significa que podemos eliminar esta opción, y que la otra era la correcta. De alguna forma, elegir el número correcto es menos afortunado, porque no estaremos seguros de que lo es hasta que hayamos resuelto todo el sugoku."

Como veis es un proceso basado en especulaciones.

La Estructura del Programa **(sobre la versión: 4 SDK_v04.py)**

Ahora has de abrir el SW y seguirlo con las explicaciones que te presento. Verás que en el mismo he incluido numerosas aclaraciones, es mi deseo que te ayuden a entenderlo.

Dejaremos para más adelante las funciones de apoyo (todas tienen < docstring> para explicar su funcionamiento). Empezamos:

Como nuestro reto es para cualquier tipo de Sudoku cuadrado, lo primero que haremos será informar al programa el tipo de Sudoku que queremos resolver y cual es la cadena base de elementos posibles. Además le informaremos en que fichero le vamos a pasar el Sudoku a resolver, en que fichero queremos la salida de las soluciones, el máximo numero de soluciones que queremos por si el Sudoku tuviera más de una y otros parámetros que nos permitan visualizar (si quisiéramos) como va obteniendo las soluciones por estrategia y su grado de complejidad. Aclaro, la Complejidad es un concepto relativo asociado al SW de resolución y personalmente lo defino en base a las veces que hemos entrado en Fuerza Bruta y en que profundidad.

Como curiosidad, la resolución del famoso Sudoku del Dr.Arto Inkala mencionado en la página 11 del (Revista "Pensamiento Matemático " | 123) artículo de Dialnet, presenta grados de dificultad diferentes si lo resolvemos con versión-3 (SDK_v03.py) que con la versión-4 (SDK_v04.py). Indiscutiblemente utilizar más estrategias reduce el paso por Fuerza Bruta.

	Combinaciones Probadas	**Máxima Profundidad alcanzada**
SDK_v03.py	350	15
SDK_v04.py	188	11

Cuando comentemos la salida de las soluciones, ampliaremos estos dos conceptos mencionados anteriormente.

Por cierto, se menciona a una persona que fue capaz de resolverlo después de 10 horas y que no le gustó que tuviera más de una solución, y se menciona una referencia (web20 con enlace al final del artículo) a la que no he conseguido acceder donde se supone que presenta esa incoherencia (más de una solución). El Sudoku del Dr. Arto Inkala tiene una única solución.

El fichero de configuración se denomina <sdk.ini>, al abrirlo podéis observar aclaraciones de los parámetros de configuración, algunos son obligatorios y otros opcionales y lo que se pretende con ellos.

Una vez procesado el fichero de configuración almacenaremos sus parámetros en un diccionario (estructura de datos de python).

Después de este primer paso, crearemos las primeras variables críticas que vamos a utilizar a lo largo del programa :

cSDK = Cadena base de elementos posibles (ej: “123456789”)

nSDK = Tamaño del Sudoku (ej: → 9)

nRSDK = Raíz cuadrada del número de elementos (ej: → 3)
¿Porqué la raíz cuadrada? → nos ayudará a recorrer los elementos de los Cuadros/Regiones

aVar = Matriz cuadrada donde guardaremos por casilla la solución de la misma o posibles candidatos a la solución (lista de 81 casillas).

aFil = Matriz donde guardaremos por fila los valores de las casillas resueltas (9 elementos)

aCol = ídem por columnas

aInt = ídem por cuadro/región

Nuestro siguiente paso será pasar al SW el Sudoku a resolver, lo haremos a través de un fichero.

Veámoslo a través de algunos ejemplos:

1º Aspecto visual conocido:

```
 | | | |2| | |1| |
9| | | | |7| | | |
3| | |5| | | | | |
-----------------
 |3| |2| | | | |9|
 | | |7|6|1|4| |2|
 | | | |9|5| | | |
-----------------
4| |5| | |6|7|9| |
 | | | | | |1|4| |
8|1| |4|5| | | | |
-----------------
```

2º Como una sola fila:

```
       5     35 2    61   9   743   23    1 8665  2  17 61           2  9 4
```

3º Otra forma:

```
1    7 9
 3  2   8
  96  5
  53  9
 1  8   2
6    4
3      1
 4      7
  7   3
```

La carga sólo entiende la cadena alfanumérica: cSDK (123456789 caso Sudoku 9x9) y el carácter " " (espacio) como casilla a resolver; por tanto, tratará de leer tantos caracteres válidos como casillas tenga el Sudoku (81 casillas en el caso de un Sudoku de 9x9), así que podemos pasarle el Sudoku a resolver como más nos apetezca, sabiendo como lo va a leer.

Recibido, lo cargaremos en la variable aVar.

A continuación abriremos el fichero donde cargaremos la/s solución/es, en modo escritura, no lo pisaremos e iremos cargando al final nuestras salidas. Veamos una salida y su interpretación:

```
****************************************
* Fecha-Hora: 2019-11-07 14:01:28.948563
* Tipo: 9x9
* Elementos Base: 123456789
* Nº Max. de Soluciones Buscadas:100
*---------------------------------------
* Altura Maxima Probable:    29
* Altura Maxima Alcanzada:     3
* Combinaciones Posibles:
  |  |  |  |  |  |  |  |  |
 2| 2|  |  |  |  | 2|  | 2|
 2| 2|  |  |  |  |  |  |  |
---------------------------
  |  | 3|  |  |  |  | 3| 3|
 5| 5| 4|  | 2|  | 3| 4|  |
 3| 3|  |  | 2|  | 3|  |  |
---------------------------
 3| 3|  |  |  |  |  | 3| 3|
```

```
 3| 3| 3|  |  |  |  | 3|
  |  | 2|  |  |  | 2|  |
--------------------------
* Combinaciones Probadas:      14
* Soluciones Halladas:       8
**************************************
 67     4  2  3       983   1   5               9   7   425       7  1  1    68
367512984892473516451698327716825439249137658538964271984251763623789145175346892
367512984892473516541698327716825439439167258258934671984251763623789145175346892
367512984892473615451698327716825439945137268238964571684251793523789146179346852
367512984892473615451698327716825493945137268238964571684251739523789146179346852
367512984982473516451698327716825439249137658538964271894251763623789145175346892
367512984982473516541698327716825439439167258258934671894251763623789145175346892
367512984982473615541698327713825469495167238628934571834251796256789143179346852
367512984982473615541698327713825496495167238628934571834251769256789143179346852
```

En este caso hemos pasado un Sudoku no bien planteado (como mandan los puristas) pues tiene más de una solución.

Las tres primeras líneas indican:

- **Fecha y hora** de cuando se lanzó.
- **El tipo**: representado por el conjunto de los elementos base.
- **Elementos Base**: representado por la cadena de posibles elementos.
- **El máximo número de soluciones** que podríamos buscar.

Las siguientes 3 líneas junto con la matriz indican que tuvimos que entrar en Fuerza Bruta :

- **Altura Máxima Probable:** indica cuantas casillas sin resolver nos quedaban (29) antes de entrar en Fuerza Bruta.

- **Altura Máxima Alcanzada:** indica cual fue el pico mayor de especulaciones del total que tuvimos que hacer para llegar a la solución.
Una especulación es plantear una solución (válida o no) a una casilla no resuelta. Planteada una solución volvemos a lanzar las estrategias de resolución (1º, 2º, 3º) y si a pesar de ello seguimos sin solucionarlo, volvemos a plantear una nueva especulación y así sucesivamente. Si la especulación no fue válida daremos marcha atrás. En nuestro caso fueron 3 hacia delante.

- **Combinaciones Posibles:** nos presenta una matriz con las casillas no resueltas (29) y en cada una de ellas el número de candidatos posibles.

- **Combinaciones Probadas:** indica el número total de especulaciones que tuvimos que hacer para llegar a la solución.

Resto de líneas::
- El número de **Soluciones Halladas** (8)

- En formato linea , **el Sudoku a solucionar y sus soluciones**. Obsérvese como curiosidad que columnas varían en la evolución de las soluciones. Las que no deben cambiar nunca son las casillas con solución en el Sudoku propuesto.

Nuestro siguiente paso será verificar que al menos el Sudoku es congruente. Con la congruencia verificamos que por Fila, Columna, Cuadro/Región no tenemos elementos repetidos. Algunas veces podemos encontrarnos incongruencia más adelante en la ejecución del SW, se comentará.

NOTA: La numeración de los Cuadros/Regiones en el Sudoku van de izquierda a derecha y de arriba abajo. Igual sucede con las casillas de un cuadro.
Deténgase un momento a observar como se resuelve el recorrido de los Cuadros/Regiones y sus casillas independientemente del tipo de Sudoku cuadrado.

Seguimos cargando algunas variables, de las cuales la que más nos interesa es nvuelta. Si es mayor que 1 indica que en algún momento hemos entramos en Fuerza Bruta.

A continuación entramos en un bucle infinito donde nuestro primer paso será cargar las variables comentadas anteriormente: aFil, aCol, aInt

Para la matriz siguiente veamos algunos ejemplos:

```
 | | | |2| | |1| |
9| | | | |7| | | |
3| | |5| | | | | |
-----------------
 |3| |2| | | | |9|
 | | |7|6|1|4| |2|
 | | | |9|5| | | |
-----------------
4| |5| | |6|7|9| |
 | | | | | |1|4| |
8|1| |4|5| | | | |
-----------------
```

aFil[0] → “21”
aFil[9] → “8145”
aCol[4] → “2695”
aInt[8] → “7914”

Ojo: Recordar que en python los elementos de las listas, tuplas, conjuntos y diccionarios comienzan a enumerarse a partir del 0.

Inicializaremos la variable lsal que nos indicará el paso a las siguientes estrategias o a Fuerza Bruta.

Quiero detenerme para que entendáis lo que representan las variables nvuelta y lsal y como interaccionan en este bucle infinito.

Casuística:

nvuelta	lsal	Que nos indica
0	True	No hemos encontrado más casillas con un solo elemento: pasar a la siguiente estrategia
0	False	Se controla dentro de la función Bucle1() y representa una incongruencia: indica que el Sudoku no tiene soluciones.
>0	True	Entramos en algún momento en Fuerza Bruta y estamos trabajando con una especulación: pasamos a la siguiente estrategia.
>0	False	Entramos en algún momento en Fuerza Bruta y estamos trabajando con una especulación. Trabajando con la misma hemos encontrado una incongruencia, que nos indica que nuestra suposición (la especulación) no es correcta: pasamos a Fuerza Bruta para dar pasos atrás.

A continuación entraremos en la **1º estrategia.**

Las funciones con las que se implementa la 1º estrategia:
- Bucle1()
- est_smp()
- ele_comp()

La idea es que mientras vaya descubriendo elementos únicos en casilla, voy repitiendo la misma estrategia.
Si ya no hemos encontrado mas casillas únicas, paso a la siguiente estrategia, la 2º. Durante la ejecución del mismo cargaremos también todas las casillas con más de un candidato.

Las funciones con las que se implementa la **2º estrategia**:
- Bucle2()
- est_Cmp()

La idea es la siguiente: si descubro una casilla con un elemento único, salgo de esta estrategia y vuelvo a la 1º estrategia. En caso contrario, paso a la siguiente estrategia, la 3º.

Las funciones con las que se implementa la **3º estrategia:**
- Bucle3()
- est_Dpareja_Fil()
- est_Dpareja_Col()
- est_Dpareja_Cua()
- situacion()

Esta función tratará de aplicar la estrategia de Par Desnudo/Doble Pareja. Repetiremos el proceso mientras logremos reducir candidatos. Si en algún momento logramos encontrar el candidato único de una casilla, al finalizar todo el proceso de reducción de candidatos, iniciaremos el proceso desde la 1º estrategia.

En esta funcion encontraremos un nueva variable aPend que contiene una lista de ?x3 que recogerá de aVar las casillas no solucionadas, es decir, con más de un candidato, de la siguiente forma:
- Número de Ffila de casilla
- Número de Columna de casilla
- Cuantos candidatos a esta casillas

La función situacion() es la encargada de actualizar esta variable.
Esta nueva variable la reordenaremos en varias ocasiones por el tercer elemento "número de candidatos". Recordar que necesitamos casillas con dos candidatos para aplicar esta estrategia.

Casuística del retorno de Bucle3() :

Bucle3()	Que nos indica
0	No hemos conseguido reducir: pasamos a Fuerza Bruta
1	Hemos conseguido reducir casillas y al menos una de las reducciones es de un elemento: pasamos a 1º estrategia.
2	Hemos conseguido reducir casillas pero ninguna con un solo elemento: pasamos a Fuerza Bruta.
3	Hemos encontrado una incongruencia que nos indica que nuestra suposición (la especulación) no es correcta: pasamos a Fuerza Bruta dando pasos atrás. Aclaración: en 1º vuelta, es decir, con nvuelta = 0 nunca puede producirse una incongruencia, por tanto si salimos con un 3 es porque en algún momento anterior entramos en Fuerza Bruta.

En este punto preguntaremos que tal nos fue, prepararemos la salida de la solución/es y veremos si aún es posible encontrar más soluciones.

Si no aun no la hemos encontrado o tienes más soluciones entraremos en **Fuerza Bruta.**
Tenéis una amplia descripción en el propio código.

La idea es crear una pila (el último en entrar es el primero en salir) con una lista aBruta de ?x3 que contendrá:
- La lista matriz aVar en un instante determinado
- La lista aPend en un instante determinado (ordenada por el tercer elemento)
- Un contador
Este contador nos permitirá comprobar si especulamos con todos los candidatos a esa casilla.

Como se describe en la explicación del propio código podemos entrar si:
- Encontramos una incongruencia (nuestra especulación no fue la correcta) y debemos probar con los siguientes candidatos
-No hemos dado todavía con la solución e incrementamos aBruta con un nuevo elemento.

Recordar que al iniciar una nueva especulación probaremos si es válida iniciando todas las estrategias.

NOTA: El Sudoku planteado en la figura 8 del artículo de Dialnet parte izquierda, tiene 8 soluciones, lo que no nos sirve para compararlo con el que planteado en la derecha que sólo tiene una solución. Adjunto a continuación las soluciones de uno y otro.

Sudoku de la Izquierda:

```
 67     4  2  3       983   1   5               9   7   425       7  1  1    68
367512984892473516451698327716825439249137658538964271984251763623789145175346892
367512984892473516541698327716825439439167258258934671984251763623789145175346892
367512984892473615451698327716825439945137268238964571684251793523789146179346852
367512984892473615451698327716825493945137268238964571684251739523789146179346852
367512984982473516451698327716825439249137658538964271894251763623789145175346892
367512984982473516541698327716825439439167258258934671894251763623789145175346892
367512984982473615541698327713825469495167238628934571834251796256789143179346852
367512984982473615541698327713825496495167238628934571834251769256789143179346852
```

Sudoku de la Derecha:

```
 43     162  18 3 1  3   5    5  2    9   1    2  3    3   1  8 1 26  452     61
843752961625918437197346852361589274759624183482173596536491728918267345274835619
```

Aquí tenéis un Sudoku de 25x25

```
! | | | | ! | | | | ! | | | | ! | | | | ! | | | | ! |
! | | | | ! |M|O| | ! | | | | ! | | | | ! | | | |6| !
! | | | |G|! | | |4|N|!5| |J| | ! | |C| | ! |I|9| |F!
! | | | | !9| | | | !N|6|K| | ! |M| | |F!E| |G|D|4!
!O| |P| | ! | |5| | !I| | | | !H|1|6|E| !C| |J|N|A!
-------------------------------------------------
! | | | | ! |K| |B| ! | | | | ! | | |A|1!7|F|6|E|5!
! |F|L|G|2!I|1|A|5|E!K|7|N|8|P!6|9|H|O|M!3| |B|J|C!
!J|N|H| |E!C|3|F|M|8!O|L|9|1| !7|D|2|K|B! |4|P|I|G!
!I|K|5|P|9! |7|6|D|4!A|G| |B|M!3|E|J|8|C!1| |O|L|H!
!7|A|1|B|3!J|O|H|2|G!6|C| |E|I! |F|P|4|L!9|N|K|M|8!
-------------------------------------------------
! |3|C| |K!7|E| | |F! |5|P|2|J!8|L|G|9|H!D|O| |B| !
!2|6|I|E|J! | |8|A|5!D|O|1|C|K!B|P| | |4!M|H|L| |7!
!G|O|D|7|H! |4|B|I|L!F|N|3|6| ! | |K|5|A!8| |E| |P!
!P|5|B|L|1!O|C| |9|J!H|8|A|4|7!I|2|E|6|D!N|G|3|F|K!
!N|9| |F|M!D|2| |H|P!E|B|I|G|L!1|C|7|3|O!5|J|A| |6!
-------------------------------------------------
!8|1|2| |C!A|F|I|O| !G|K|4|D|N!L|7| |P|6!J|E|5|H|M!
!D|M|G|I|6!2| | | |K! |1| |O| !J| |4|H| ! | |C| | !
!L|7|J|5|B! | |G|E|C!3|P|M| | !D|N|F|1|8!O|A| | |9!
! |4| |O|N!5| |1|L|D! |H|B|J| !E|K|A|M| ! | |I| | !
!E|P|K|H|A! | | | |B! |F| |5|6!C|3|O|I| !G| |D| | !
-------------------------------------------------
! |G|N|3|L! |8| | | ! | | |A|C!K| | | | ! |P|7| | !
!F|H|7|8|4! | |2| |O! |D| |3| ! |I|5| | ! | |M| | !
!K| |M|A|P! | | | |I! |4| |N|G!F| |9| | ! | |H| | !
!1|B|E|D| ! | | | |A! |9| |K| !G| | | | ! |8| |C| !
!C|I| |J|O! | | | |H! |M| | | ! |6|D| | ! | |N| | !
-------------------------------------------------
```

El archivo que os dejo < sdk > contiene Sudokus cargados para su resolución.
También mis archivos de salida < sdk04.txt >, < sdk09.txt >, < sdk16.txt >, < sdk25.txt >, con diferentes informes de salida.

Getafe 18 de noviembre de 2019