Ho Chi Low, hol31@pitt.edu

SUDOKU SOLVER IN PYTHON

The repository SUDOKU SOLVER IN PYTHON is a sudoku solver which shuffles possible outcomes to blanks of two feasible numbers (dual blanks). `sudoku3.py` is main file of the solver, while `sudoku3Fun.py` contains all essential functions supporting the main file.

1. The Solve Engine

Before applying any strategy to solve a sudoku puzzle, it is important to simplify the puzzle as much as possible. If there is any blank of the puzzle which contains 1 possible number only, then it is filled by this number. (PuzzleNo1001)

$$\left[\begin{array}{ccc|ccc|ccc}
9 & 0 & 0 & 6 & 4 & 0 & 0 & 0 & 3\\
2 & 7 & 0 & 0 & 9 & 0 & 5 & 8 & 0\\
0 & 1 & 0 & 5 & 8 & 0 & 0 & 0 & 0\\ \hline
0 & 9 & 0 & 0 & 0 & 0 & 7 & 0 & 0\\
0 & 0 & 7 & 9 & 6 & 5 & 8 & 0 & 0\\
0 & 0 & 2 & 0 & \mathbf{0} & 0 & 0 & 4 & 0\\ \hline
0 & 0 & 0 & 0 & 5 & 3 & 0 & 6 & 0\\
0 & 5 & 1 & 0 & 7 & 0 & 0 & 2 & 8\\
4 & 0 & 0 & 0 & 1 & 6 & 0 & 0 & 5
\end{array}\right] \Rightarrow \left[\begin{array}{ccc|ccc|ccc}
9 & 0 & 0 & 6 & 4 & 0 & 0 & 0 & 3\\
2 & 7 & 0 & 0 & 9 & 0 & 5 & 8 & 0\\
0 & 1 & 0 & 5 & 8 & 0 & 0 & 0 & 0\\ \hline
0 & 9 & 0 & 0 & 0 & 0 & 7 & 0 & 0\\
0 & 0 & 7 & 9 & 6 & 5 & 8 & 0 & 0\\
0 & 0 & 2 & 0 & \mathbf{3} & 0 & 0 & 4 & 0\\ \hline
0 & 0 & 0 & 0 & 5 & 3 & 0 & 6 & 0\\
0 & 5 & 1 & 0 & 7 & 0 & 0 & 2 & 8\\
4 & 0 & 0 & 0 & 1 & 6 & 0 & 0 & 5
\end{array}\right].$$

`firstfill(M, fillMatrix)` in `sudoku3Fun.py` is a function to fill any blank of single outcome. Another function `myExhaustFill(M, fillMatrix)` is a more sophisticated version of `firstfill`, which fills a number to an empty blank because this number fails to fit to any other blank of the same row/column/block. (PuzzleNo1007)

$$\left[\begin{array}{ccc|ccc|ccc}
0 & 3 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & 4\\
0 & 0 & 8 & 0 & 0 & 3 & 0 & 5 & 0\\
5 & 0 & 0 & 0 & 2 & 8 & 6 & 0 & 0\\ \hline
1 & 0 & 5 & 0 & 4 & 0 & 0 & 0 & 0\\
0 & 6 & 0 & 5 & 0 & 0 & 0 & 0 & 2\\
4 & 0 & 0 & 0 & 3 & 0 & 8 & 1 & 0\\ \hline
0 & 0 & 9 & 0 & 0 & 2 & 7 & 0 & 0\\
0 & 5 & 0 & 3 & 7 & 0 & 0 & 0 & 6\\
3 & 0 & 0 & 6 & 0 & 0 & 5 & 2 & 0
\end{array}\right]$$

The (1,8)-entry of the above puzzle accepts number 7, 8 or 9. However, the number 8 on the first row can only be filled in the 8-th column. So the (1,8)-entry must be '8'.

Bootstrapping is also carried out in the solve engine. The function `myBootstrap(M,fillMatrix)` in `sudoku3Fun.py` is designed for finding one solution to the prescribed sudoku puzzle. Suppose that a sudoku puzzle is left with blanks in the entries (i_1, j_1), (i_2, j_2), ... , (i_N, j_N). Let

$$E_{(i_k,j_k)} = \Big\{ a^n_{(i_k,j_k)} \mid a^n_{(i_k,j_k)} \text{ is a feasible number in entry } (i_k, j_k). \Big\} \quad \text{with} \quad s_k = |E_{(i_k,j_k)}| > 1.$$

The total number of combinations to fill the remaining blanks of the puzzle, is $T = \prod_{k=1}^N s_k$. A possible combination is represented by a sequence of N numbers:

$$\epsilon^t = \Big(a^t_{(i_1,j_1)}, a^t_{(i_2,j_2)}, \cdots , a^t_{(i_{N-1},j_{N-1})}, a^t_{(i_N,j_N)}\Big),$$

where $a^t_{i_k,j_k}$ belongs to $E_{(i_k,j_k)}$ and $t = 0, 2, \cdots , (T - 1)$. For simplicity, we write $a^t_k = a^t_{i_k,j_k}$. These combinations could be ordered from ϵ^0 to $\epsilon^{(T-1)}$ in the following sense.

Position	Modulo	Choices
(i_1, j_1)	T/s_1	$a^0_1, a^1_1, \cdots , a^{(s_1-1)}_1$
(i_2, j_2)	$T/(s_1 s_2)$	$a^0_2, a^1_2, \cdots , a^{(s_2-1)}_2$
$\vdots$	$\vdots$	$\vdots$
(i_{N-1}, j_{N-1})	$T/(s_1 s_2 \cdots s_{N-1}) = s_N$	$a^0_{N-1}, a^1_{N-1}, \cdots , a^{(s_{N-1}-1)}_{N-1}$
(i_N, j_N)	$T/(s_1 s_2 \cdots s_{N-1} s_N) = 1$	$a^0_N, a^1_N, \cdots , a^{(s_N-1)}_N$

For every t, a^t_1 is the n_1-th possible number in $E_{(i_1,j_1)}$ counting from 0 to $(s_1 - 1)$, where

$$n_1 = \left[\frac{t}{(T/s_N)}\right].$$

We may let $t = n_1(T/s_N) + r_1$ for some non-negative integer r_1 which is also smaller than (T/s_N). For the second position (i_2, j_2), we choose the n_2-th possible number in $E_{(i_2,j_2)}$ such that

$$n_2 = \left[\frac{r_1}{\big(T/(s_{(N-1)} s_N)\big)}\right].$$

In general, given $0 \le t \le (T-1)$, the combination ϵ^t consists of the numbers $\big(a^{n_k}_k\big)^N_{k=1}$. $a^{n_k}_k$ is the n_k-th number in $E_{(i_k,j_k)}$, where n_k's are defined by

$$t = (T/s_N)\, n_1 + \big(T/(s_{(N-1)} s_N)\big)\, n_2 + \big(T/(s_{(N-2)} s_{(N-1)} s_N)\big)\, n_3 + \cdots + s_1\, n_{(N-1)} + n_N.$$

If we set $r_k = t - (T/s_N)\, n_1 - \cdots - \big(T/(s_{(N-k+1)} s_{(N-k+2)} \cdots s_N)\big)\, n_k$, then we require

$$r_k < \left(\frac{T}{(s_{N-k+1})(s_{N-k+2}) \cdots (s_N)}\right) \quad \text{for every } 1 \le k \le (N-1).$$

The core strategy of the solve engine is to make use of up to 15 (adjustable) dual blanks in the puzzle and then fill them in every possible way. The total number of feasible combinations drops significantly if numbers are decided in all dual blanks, so the method of exhaustion or bootstrapping could be working more effectively until a solution to the puzzle is found.

In the following example, the sudoku puzzle (PuzzleNo1008) is given by

$$\left[\begin{array}{ccc|ccc|ccc}
8 & 7 & 1 & 3 & 4 & 5 & 0 & 0 & 9 \\
0 & 4 & 0 & 0 & 0 & 0 & 1 & 0 & 8 \\
0 & 0 & 0 & 0 & 9 & 0 & 3 & 0 & 4 \\
\hline
0 & 3 & 0 & 4 & 7 & 9 & 5 & 8 & 0 \\
0 & 9 & 8 & 0 & 0 & 2 & 4 & 0 & 7 \\
7 & 0 & 4 & 0 & 0 & 0 & 9 & 0 & 0 \\
\hline
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
0 & 0 & 0 & 6 & 8 & 0 & 0 & 4 & 3 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right].$$

Reference: www.sudoku.com

There are (at least) 10 dual blanks in the puzzle. Their positions and respective possible outcomes are listed on the table below.

Position	:	Numbers
(1,7)	:	2 or 6
(1,8)	:	2 or 6
(2,1)	:	3 or 9
(2,3)	:	3 or 9
(4,1)	:	1 or 2
(4,3)	:	2 or 6
(5,4)	:	1 or 5
(5,8)	:	1 or 3
(8,7)	:	2 or 7
(9,9)	:	1 or 2

By choosing 1 number from each position, we obtain $2^{10} = 1024$ separate cases such that these 10 empty blanks are filled. For example, one case is:

$$\left[\begin{array}{ccc|ccc|ccc}
8 & 7 & 1 & 3 & 4 & 5 & \mathbf{2} & \mathbf{6} & 9 \\
\mathbf{3} & 4 & \mathbf{9} & 0 & 0 & 0 & 1 & 0 & 8 \\
0 & 0 & 0 & 0 & 9 & 0 & 3 & 0 & 4 \\
\hline
\mathbf{1} & 3 & \mathbf{2} & 4 & 7 & 9 & 5 & 8 & 0 \\
0 & 9 & 8 & 0 & \mathbf{5} & 2 & 4 & \mathbf{1} & 7 \\
7 & 0 & 4 & 0 & 0 & 0 & 9 & 0 & 0 \\
\hline
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
0 & 0 & 0 & 6 & 8 & 0 & \mathbf{7} & 4 & 3 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{2}
\end{array}\right].$$

The puzzle appears to be reasonable at this stage. We proceed to the functions `myExhaustFill` and `myBootstrap` to complete the remaining blanks. If a solution to the sudoku puzzle doesn't exist after a dual-blank filling, then we have to move to the next case (out of 1024 cases) of the dual-blank filling.

The efficiency of the dual-blank filling is restricted by the number of dual blanks (N) in the sudoku puzzle. If $N = 15$, then $2^{15} \approx 3 \times 10^4$ and the total number of combination (T) is reduced to $T/(3 \times 10^4)$. The sudoku puzzle (puzzleNo1008) has a value of $T \approx 2.5 \times 10^{23}$. Therefore, under every case of dual-blank filling, we still have to consider $T' \approx 8.3 \times 10^{18}$ ways of finishing the rest of the puzzle. It would not be efficient to carry ou bootstrapping immediately after the dual-blank filling.

To address this limitation of the dual-blank filling, recursion is performed in the core function `mySolveEngine(M, fillMatrix, valid, total)` of the solve engine.

$$\begin{tikzcd}
\text{Fill by exhaustion} \arrow[r] & \text{DB-filling} \arrow[r] \arrow[l, bend left] & \text{Bootstrapping}
\end{tikzcd}$$

Given a sudoku puzzle of T possible combinations of numbers in empty blanks, it is simplified by `myExhaustFill` before a step of dual-blank filling is carried out. Let T' be the new total number of combinations after the DB-filling. If $T' < 1 \times 10^6$, we perform bootstrapping (`myBootstrap`) instantaneously to finish the puzzle, or go back to proceed the next case of the DB-filling. If $T' > 1 \times 10^6$, we do a recursion within the core function `mySolveEngine`.

2. Procedure

The main file `sudoku3.py` can be executed directly to solve a sudoku puzzle stored in the file `mySudokuBank.txt`. Every sudoku puzzle stored is labeled by a puzzle number, e.g. `PuzzleNo1001`. This number has to be typed to access the corresponding puzzle.

Type the puzzle number: 1001

The puzzle is shown in form of a 9×9 matrix. The difficulty of the puzzle is evaluated by the number of blanks and the total number of combinations (T), which are printed after the matrix. There are four options to choose in the program.

Option 1: myBootstrap
Option 2: myExhaustFill
Option 3: mySolveEngine
Option 4: NDBsolveEngine
Option 0: QUIT

Typing '0' to choose Option 0, allows the user to quit the program.

Choose Option: 0

Option 1 and Option 2 are to solve the puzzle by the function `myBootstrap` and `myExhaustFill`. Neither of the options is desirable to solve a rather complicated sudoku puzzle. Users are advised to be cautious about running the function `myBootstrap` if the total number of combination exceeds 1×10^{10}.

Option 3 is to solve a sudoku puzzle by the solve engine. It would run automatically and display the solution found by the program. At the end, the main menu will appear again, and the user may choose Option 0 to quit the program.

End

To Be Continued