## Problem Set 3 Exercise #29: Mini Sudoku [Challenge]

**Reference:** Lecture 9 notes

**Learning objective:** Algorithm design; Writing long programs

**Estimated completion time**: 140 minutes

**Problem statement:**

Sudoku is a popular logic-based number placement puzzle. The 9×9 board has 9 rows, 9 columns and 9 sections of 3×3 cells. The objective is to fill the board so that each row, each column and each section contains the digits from 1 to 9. There is only one unique solution. **Figure 1** below, taken from Wikipedia: Sudoku, shows a Sudoku puzzle and its solution.

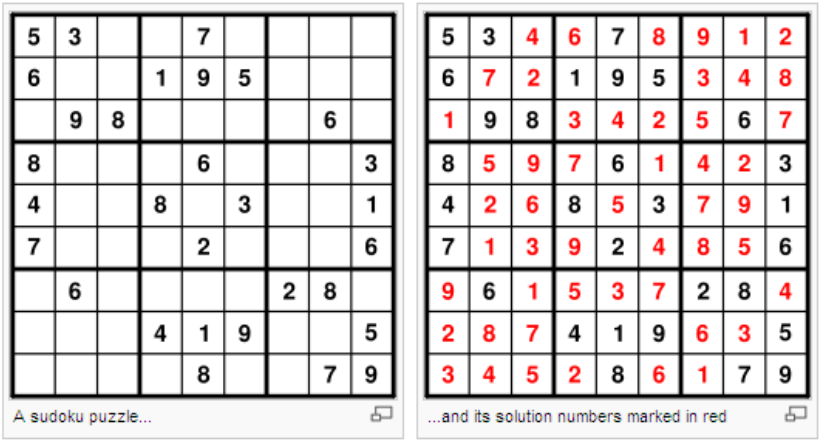


**Figure 1.** A Sudoku puzzle and its solution

In this exercise, we shall solve mini-Sudoku puzzles on a **4×4 board**. The board consists of digits from 0 to 4 where 0 represents a blank cell. The solver needs to replace all the 0s with the correct values (1 - 4).

We will only give you the simplest Sudoku puzzles to solve: these puzzles are such that at any time, there is at most one blank cell (0) in a certain row, a column, or a 2×2 section. Hence, you are able to fill in the blank cells progressively until the whole board is filled.

A very simple algorithm to solve mini-Sudoku is described below.

**Repeat the following until no more blank cells can be filled:**

- **For each row, check whether there is a single 0. If so, replace that 0 with the obvious value.**
- **For each column, check whether there is a single 0. If so, replace that 0 with the obvious value.**
- **For each 2×2 section, check whether there is a single 0. If so, replace that 0 with the obvious value.**

We will use an example to illustrate the above algorithm. **Figure 2a** below shows a puzzle with seven blank cells, labelled from ***A*** to ***G*** for easy reference.

| | | | |
|---|---|---|---|
| 1 | *A* | 3 | 4 |
| 3 | *C* | 1 | 2 |
| 4 | 3 | 2 | 1 |
| 2 | 1 | 4 | 3 |

**Figure 2a.** Sample image

| | | | |
|---|---|---|---|
| 1 | *A* | 3 | 4 |
| 3 | *C* | *D* | 2 |
| 4 | 3 | 2 | 1 |
| 2 | *F* | *G* | 3 |

**Figure 2b.**

| | | | |
|---|---|---|---|
| 1 | *A* | 3 | *B* |
| 3 | *C* | *D* | 2 |
| 4 | 3 | 2 | 1 |
| *E* | *F* | *G* | 3 |

**Figure 2c.**

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 3 | 4 | 1 | 2 |
| 4 | 3 | 2 | 1 |
| 2 | 1 | 4 | 3 |

**Figure 2d.** The solution

After examining each row, we cannot fill in any blank cell as none of the rows has a single blank cell.

We proceed to examine every column. In the first column, we find that we can fill ***E*** with 2, and in the fourth column, ***B*** with 4, as shown in **Figure 2b**.

We proceed to examine every 2×2 section. We find that we can fill ***D*** with 1, ***F*** with 1, and ***G*** with 4. See **Figure 2c**.

The puzzle is still not solved, so we repeat the process. As we examine the rows this time, we find that we are able to fill ***A*** with 2, and ***C*** with 4. This gives the solution as shown in **Figure 2d**.

You would probably want to write a modular program as the code is complex and modular design helps in plan and debugging.

**Sample run #1:**

```
Enter board (0 for blank cell):
1 0 3 0
3 0 0 2
4 3 2 1
0 0 0 3
The Sudoku puzzle solved:
1 2 3 4
3 4 1 2
4 3 2 1
2 1 4 3
```

**Sample run #2:**

```
Enter board (0 for blank cell):
0 1 3 2
2 0 1 0
1 0 0 3
3 4 2 1
The Sudoku puzzle solved:
4 1 3 2
2 3 1 4
1 2 4 3
3 4 2 1
```