

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

Reference: Lecture 8 notes

Learning objectives: Two-dimensional array; Algorithm design; Writing long programs

Estimated completion time: 180 minutes

Problem statement:

In Prime Minister's Founders Forum Smart Nation speech on April 20, 2015, Prime Minister Lee Hsien Loong revealed his secret love for coding. The last computer programme he wrote was a Sudoku solver, written in C++ several years ago:



Lee Hsien Loong
about 2 years ago



```
151
152 void Place(int S)
153 {
154     LevelCount[S]++;
155     Count++;
156
157     if (S >= 81) {
158         Succeed();
159         return;
160     }
161
162     int S2 = NextSeq(S);
163     SwapSeqIntries(S, S2);
164
165     int Square = Sequence[S];
166
167     int BlockIndex = InBlock(Square);
168     RowIndex = InRow(Square);
169     ColIndex = InCol(Square);
170
171     int Possibles = Block[BlockIndex] & Row[RowIndex] & Col[ColIndex];
172     while (Possibles) {
173         int valbit = Possibles & (~Possibles); // Lowest 1 bit in Possibles
174         Possibles &= ~valbit;
175         Entry[Square] = valbit;
176         Block[BlockIndex] &= ~valbit;
177         Row[RowIndex] &= ~valbit;
178         Col[ColIndex] &= ~valbit;
179
180         Place(S + 1);
181
182         Entry[Square] = BLANK; // Could be moved out of the loop
183         Block[BlockIndex] |= valbit;
184         Row[RowIndex] |= valbit;
185         Col[ColIndex] |= valbit;
186     }
187
188     SwapSeqIntries(S, S2);
189 }
190
191 int main(int argc, char* argv[])
192 {
193     int i, j, Square;
194
195     for (i = 0; i < 9; i++)
196         for (j = 0; j < 9; j++) {
197             Square = 9 * i + j;
198             InRow[Square] = i;
199             InCol[Square] = j;
200             InBlock[Square] = (i / 3) * 3 + (j / 3);
201         }
202
203     for (Square = 0; Square < 81; Square++) {
204         Sequence[Square] = Square;
205         Entry[Square] = BLANK;
206         LevelCount[Square] = 0;
207     }
208
209     for (i = 0; i < 9; i++)
210         Block[i] = Row[i] & Col[i] & 0x05;
211
212     ConsoleInput();
213     Place(SeqPtr);
214     printf("Unsuccessful Count = %d\n", Count);
215
216     return 0;
217 }
218
219
220
221
```

I told the Founders Forum two weeks ago that the last computer program I wrote was a Sudoku solver, written in C++ several years ago (<http://bit.ly/1DMK5Zk>). Someone asked me for it. Here is the source code, the exe file, and a sample printout - <http://bit.ly/1zAXbua>

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](https://en.wikipedia.org/wiki/Sudoku), shows a Sudoku puzzle and its solution.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A sudoku puzzle...

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

...and its solution numbers marked in red

Figure 1. A Sudoku puzzle and its solution

In this exercise, we shall solve mini-Sudoku puzzles on a **4×4 board** (easier than what PM has solved). 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	B
3	C	D	2
4	3	2	1
E	F	G	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	4
3	C	1	2
4	3	2	1
2	1	4	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**.

An almost empty skeleton program **PS3_Ex29_MiniSudoku.java** is given to you. Feel free to edit it. 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
```