UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
SCHOOL OF COMPUTER SCIENCE & APPLIED MATHEMATICS

COMS1017A\COMS1021A
Intro. to Data Structures & Algorithms
Project 1: Sudoku

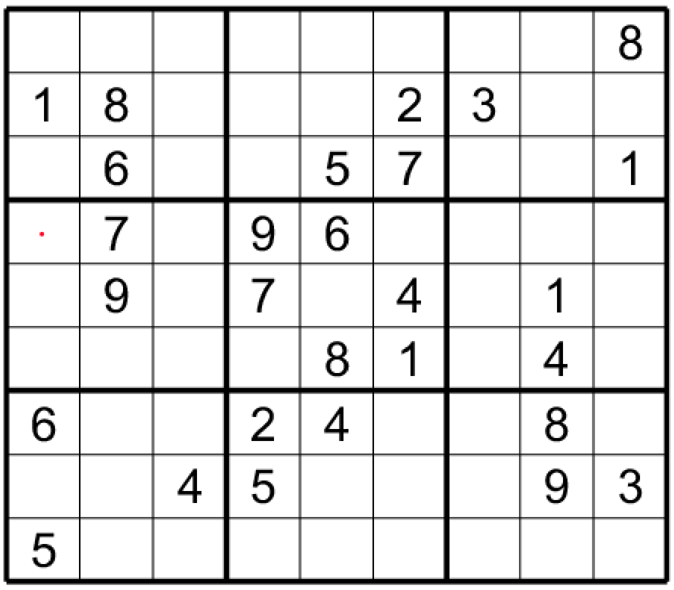
Prof. Richard Klein

Due Date: Monday 22 September 2023, 17:00

1 Description

A Sudoku is a number-placement puzzle. An $n \times n$ grid has n rows, n columns and n blocks (which is made of a $k \times k$ grid of cells, where $n = k^2$). Some cells will have numbers already, and the goal is to place a number from $\{1, \ldots, n\}$ into each cell, such that each row, column and block contains every number once and only once. A valid Sudoku puzzle should have one unique solution.

For example, in Figure 1, there are 9 rows and columns and 9 3×3 blocks. There are some cells with numbers in them already. The goal is to fill each empty cell with a number between 1 and 9 (inclusive). We can place 4 into the cell in the fourth row and first column (marked with a dot). This is a valid move because the relevant row and column do not currently contain a 4. The 3×3 block needs a 4, but there are already 4s in the fifth and sixth rows. Similarly, it cannot be placed to the right of the 7 as there is already a 4 in that column.



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

Figure 1: 9×9 Sudoku – https://en.wikipedia.org/wiki/Sudoku

You can use several logical techniques to narrow down potential values for a cell and solve a given Sudoku. However, in this project, you will implement a Depth First Search (DFS) technique. A DFS is a backtracking algorithm where we make a decision and stick with it until we either solve the problem or discover that it was an incorrect choice. If it was incorrect, we backtrack or undo that decision, make a different one, and then try again. We discussed this approach in Chapter 5 of the notes.

The Sudoku you will be dealing with will either have a unique or no solution.

2 Input / Output

Input

Your program must read a Sudoku from `stdin` (using `cin`). You will read n integers, each separated by spaces. It will contain values in $\{0, 1, \ldots, n\}$. If a cell contains a 0 then treat that cell as blank. Your program should be able to handle Sudoku of various sizes:

- 9×9 – $\{0, 1, \ldots, 9\}$,
- 16×16 – $\{0, 1, \ldots, 9, A, \ldots, F\}$, and
- 25×25 – $\{0, 1, \ldots, 9, A, \ldots, P\}$.

The input will always be valid, and you do not need to handle special cases with strange whitespace, etc. Note that in competitive programming competitions and when programming generally, you should *always* check that the input is both valid and formatted correctly.

Output

Your program should output the solved Sudoku with all the values separated by spaces, or if there is no solution, you should output the text `No Solution`.

Sample IO

```
Input [Easy]:         Input [Medium]:       Input [Hard]:         Input [Easy]:
0 4 0 0 0 0 1 7 9     0 9 0 3 8 4 0 0 0     7 9 0 0 0 0 0 0 3     0 0 0 0 0 0 0 0 7
0 0 2 0 0 8 0 5 4     0 0 2 0 7 0 0 0 0     0 0 0 0 0 0 0 6 0     7 2 0 3 0 9 0 0 1
0 0 6 0 0 5 0 0 8     0 0 0 0 0 0 0 7 1     8 0 1 0 0 4 0 0 2     0 0 8 7 0 5 0 6 0
0 8 0 0 7 0 9 1 0     5 0 0 0 0 3 2 4 0     0 0 5 0 0 0 0 0 0     5 0 2 8 9 0 0 0 0
0 5 0 0 9 0 0 3 0     0 3 0 0 0 0 0 0 0     3 0 0 1 0 0 0 0 0     0 4 0 5 0 1 0 9 0
0 1 9 0 6 0 0 4 0     0 0 1 0 0 5 0 9 0     0 4 0 0 0 6 2 0 9     0 0 0 0 6 3 7 0 5
3 0 0 4 0 0 7 0 0     0 0 0 8 0 0 0 0 0     2 0 0 0 3 0 0 0 6     0 3 0 9 0 6 1 7 0
5 7 0 1 0 0 2 0 0     7 0 6 5 2 0 0 0 0     0 3 0 6 0 5 4 2 1     2 0 0 1 0 7 0 5 3
9 2 8 0 0 0 0 6 0     0 0 0 0 0 6 4 0 0     0 0 0 0 0 0 0 0 0     9 0 0 0 0 0 0 0 0
Output:               Output:               Output:               Output:
8 4 5 6 3 2 1 7 9     1 9 7 3 8 4 5 6 2     7 9 2 5 6 8 1 4 3     No Solution
7 3 2 9 1 8 6 5 4     8 5 2 6 7 1 9 3 4     4 5 3 2 1 9 8 6 7
1 9 6 7 4 5 3 2 8     4 6 3 9 5 2 8 7 1     8 6 1 3 7 4 9 5 2
6 8 3 5 7 4 9 1 2     5 8 9 7 1 3 2 4 6     6 2 5 8 9 3 7 1 4
4 5 7 2 9 1 8 3 6     6 3 4 2 9 8 7 1 5     3 7 9 1 4 2 6 8 5
2 1 9 8 6 3 5 4 7     2 7 1 4 6 5 3 9 8     1 4 8 7 5 6 2 3 9
3 6 1 4 2 9 7 8 5     3 1 5 8 4 7 6 2 9     2 8 4 9 3 1 5 7 6
5 7 4 1 8 6 2 9 3     7 4 6 5 2 9 1 8 3     9 3 7 6 8 5 4 2 1
9 2 8 3 5 7 4 6 1     9 2 8 1 3 6 4 5 7     5 1 6 4 2 7 3 9 8
```

3 Algorithm

You should implement a Depth First Search using either recursion (i.e. the call stack) or iteratively (i.e. using `std::stack`. In the iterative approach, consider using `stack<pair<int,int>>`, which stores the row and column of the most recent cell to have been given a value. High-level pseudocode is given in Algorithm 1.

Algorithm 1 Iterative Depth First Search for Sudoku

```
function SOLVE(grid[N][N])
    Initialise a stack
    while the puzzle is not complete do
        if the puzzle is in an invalid state then
            The row/col for the most recent move is always at the top of the stack.
            while the most recent move involved entering an N do
                Pop from the stack and set the value at that position to be blank
            end while
            if the stack is empty then
                return no solution
            else
                Increment the value stored at the most recent position
            end if
        else
            Find an empty cell and insert the first value for that cell
            Push this location onto the stack
        end if
    end while
    return the resulting values
end function
```

You should consider how to represent the various structures in memory efficiently. You may use any built-in structures in the C++11 STL.

4 Submission and Grading

There are 6 submissions on Moodle; you should submit the same Moodle file to all of them. The first submission is about correctness, while the remaining ones are competitive. Think of them as various leagues.

1. The "correctness" submissions run standard Input/Output test cases to ensure you can solve the Sudoku of various sizes and difficulties correctly. The three correctness test cases make up 75% of your grade.

2. The remaining 25% of your grade will be competitive. There will be separate submissions for competitions at 9×9, 16×16 and 25×25; your ranking in these competitions will earn you the remaining grades. Watch how different algorithms or aspects of your algorithm perform at different sizes.
   In the competitions, your program will be run on many different sudoku, and Moodle will report the average time per sudoku. There will be a broad mix in terms of sudoku difficulty. You may implement *any* algorithmic optimisations that you like to improve the efficiency of your algorithm and outperform your friends!

Your ranking in the competition will change as others submit or resubmit their programs. You may make as many submissions as you require. Note that the test cases on Moodle are valid but unseen input.