SUDOKU SOLVER

Anusha Porwal

The method I've used to solve the sudoku involves constraint satisfaction till whenever possible, and then at the very end, when no more numbers can be filled using constraint satisfaction, the empty cells are filled using a better version of the backtracking algorithm.

The input to the function solving the sudoku (SolveSudoku_Full) is a 2D array with the initial state and 0s present in the empty spots. I've created a separate class called Possibilities which has the following data in it: row, column, flag (to check whether the number is set in that board position), an integer array called possibilityArray of size 9, and a integer called numberOfPossibilities, which keeps track of the number of possibilities the cell has.

In my code, each cell has a possibilities object, to keep track of all the data. A while loop is run and for each cell, the possibilities are filled up in the corresponding array, by checking the row, column and the sub-grid. If for a cell there is only one possibility, that number is set on the board.

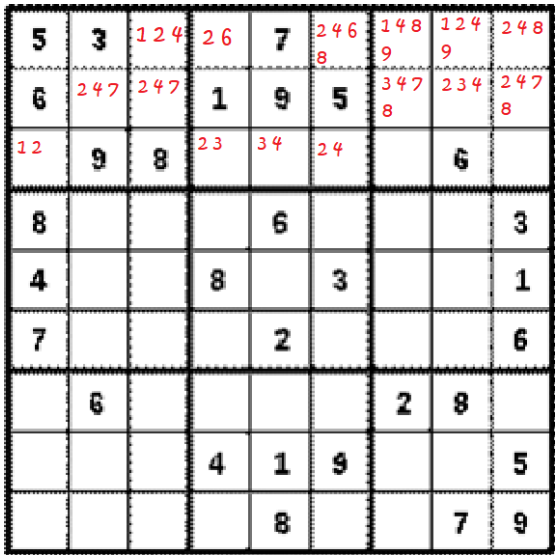
After every 3 rows, and subsequently every 3 columns, each complete sub-grid is checked. All the possibilities array of the sub-grid is checked for a number that occurs only one in all the arrays of the sub-grid. If such a number is found, it is set in that position.

Examples are given below:

5	3	124	26	7	2468	1489	1249	248
6	247	247	1	9	5	3478	234	2478
12	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

We start from the first cell and move right and continue row-wise. The numbers in red are the numbers that would be present in the possibilities array for the corresponding cell (if it is empty when found).

Here, the empty cells of the first sub-grid are filled and we see that there is no number that is there in only one of the arrays. This means that nothing can be fixed. So, we continue finding the possibilities arrays for the remaining cells.



After the possibilities of the second sub-grid are filled, it looks like this.

Now the program will go through the 5 possibilities Arrays of the sub-grid and find that 8 is only possible in one cell. So, the cell will be set with 8.

And then the program will continue moving row – wise from left to right.

When the whole array is traversed, the traversal will start from the beginning, to see if more spots can be fixed based on the new number found in the previous iteration.

We keep a count of the number of positions fixed in each iteration. If the number doesn't change in the next iteration the loop is exited, because only backtracking can solve the problem. If the count increases the loop is continued (the count will never decrease, since you can't have lesser numbers that are fixed in the new iteration than there were before).

In the backtracking part of the code, I take help of the possibilities array created in the previous steps to further narrow down and implement a slightly better version of backtracking. So, essentially it is not pure backtracking.

The final solution of the grid is shown on the GUI. The numbers in the initial state are shown on the grid in red colour, and the numbers found out by the algorithm are on the board in black.

Input can be given through the GUI or by a csv file. For the csv file, the input is the file location, and the initial state of the board will be taken from there.

Initial and final state of the code: