Practice Arena

Practice problems aimed to improve your coding skills.

Simple Sodoku

LAB-PRAC-12_FUN-STRUC

Simple Sodoku [20 marks]

Problem Statement
The standard Sodoku problem concerns a 9x9 grid and the goal is to fill up the grid with integers from 1, 2, ..., 9 so that each row, column and 3x3 block contains the digits 1 - 9 exactly once. In this problem we will consider a simpler version of the problem where the goal will be to fill up an n x n grid with the integers 1, 2, ... n where each row and column must contain each integer from 1, 2, ... n exactly once (i.e. the block constraint is removed in our simpler version of Sodoku). Let us call such an n x n grid a valid SS grid (SS stands for Simple Sodoku).

As input, we will give you a partially filled SS grid. Your job is to fill it in all valid ways possible and show us all the valid SS grids that you get. The first line of the input will give you n, a strictly positive integer. We assure you that n will be a single digit number i.e. n will be strictly less than 10. This will mean that we will work with an n x n SS grid in that test case.

In the next several lines, we will give you some entries of the SS grid that are already filled in. This will be done by giving in each line, three non-negative integers i, j, val. All three integers will be separated by a single space and will indicate that we have already filled in the entry SS[i][j] = val. The value val will be between 1 and n and the values i and j will be between 0 and n-1 since they denote an index in the SS grid. The list of partially filled entries will end when we give you the three numbers -1 -1 -1. This means that there are no more entries of the SS grid that we want to specify.

Your job is to output all possible completions of the SS grid given the already filled-in entries. To output a completion of the SS grid, first output all elements of the first row, then all elements of the second row then all elements of the third row and so on. There should be no spaces between elements of a row nor should there be any spaces or newlines between elements of two different rows i.e. the entire completion must be printed on a single line. Different completions must be printed in different lines.

Suppose there are multiple completions of the SS grid we have given you. In such a case, you have to output completions in lexicographically increasing order. Basically, notice that each completion of the SS grid looks like a giant number (recall that we promised that n will always be strictly positive but strictly less than 10). You have to view the completions as a number and output them in increasing order. In the last line of the output, print how many valid completions were there.

For example, if n = 2, then the following is a valid completed SS grid:
1 2
2 1
But the following is invalid
1 1
2 1
because the first row has the entry 1 repeated.

Caution

1. Warning: we may give you an empty SS grid to being with as well. This will happen if the very first line in the list is -1 -1 -1

2. We promise you that there will be at least one completion of the SS grid given the partially filled-in entries.
3. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You should try to solve this problem recursively. Write a function
int sudokuSolv(int** grid, int* missing, int done, int n, int m)
which takes in a partially filled in grid, a list of missing entries, the number of missing entries that have already been filled, the size of the grid n and m is the number of missing entries in the original board (including those that have already been filled and those that are remaining to be filled in).

The function returns the number of ways this partially filled grid can be completed to form a valid SS grid. The base case can be when done = m i.e. when we have already filled in all the entries. For a recursive case, choose a number to fill at the next unfilled entry and proceed.

To store the location of the missing entries in the original grid, you may either maintain 2 arrays each of size m (one storing row index of the missing entries and the other storing the column index of the missing entries). You should first store locations of all missing entries in row 1 (from left to right), then all missing entries in row 2 (from left to right) and so on. A different way (as used by the function declared above) is to index the 2D grid as a 1D array (as we saw in lectures and on Piazza) in which case you only have to maintain only one index for the missing entries since the entire grid is being represented as a 1D array.

EXAMPLE:
INPUT
```
3
0 0 1
-1 -1 -1
```

OUTPUT:
```
123231312
123312231
132213321
132321213
4
```

Explanation: The incomplete grid, as provided, looks like
```
1 * *
* * *
* * *
```
where * indicates an unfilled entry. There are exactly 4 ways of completing this grid with the given constraints. They are:
```
1 2 3
2 3 1
3 1 2

1 2 3
3 1 2
2 3 1
```

```
1 3 2
2 1 3
3 2 1

1 3 2
3 2 1
2 1 3
--------------------------------------------------------------------
```

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

Start Solving! (/editor/practice/6232)