AI - 24 Oct 2023

Sudoku basics

A standard Sudoku puzzle is a square $N \times N$ grid where $N = n^2$ and n is typically 3. There are N distinct symbols that may be affixed to empty grid positions so that each grid position gets exactly one symbol and no row nor any column has any duplicate symbols. In addition, none of the N sub-blocks of size $n \times n$ may have any duplicate symbols either.

This may be generalized so that if a Sudoku puzzle of size $N \times N$ is given, then the N subblocks will be of size $h \times w$ where h is the largest integer not greater than $\sqrt{N}$ that evenly divides N, and then $w = N/h$. None of the N rows, N columns, and N subblocks may have any duplicate symbol.

There are several basic variables and structures to set up:

1) Accept a file name from the command line, and read in a list of sudoku puzzles from this file.

1A) If instead of a file name, a sudoku puzzle is given, print out the Sudoku puzzle in 2D (and its solution, too). You could print this out like a large slider puzzle, but in the case of Sudoku the subblocks matter, so it is probably best to also have some kind of separator to indicate the subblocks. Simple ways to do this are to use the pipe and hyphen, or perhaps simply a space. How will you know whether it's a file name or a sudoku puzzle? Make reasonable assumptions.

2A) Loop through each puzzle. Per puzzle, call `setGlobals()` (or `setLookupTables()`). `setGlobals()` should first determine the size, `N`, of the puzzle and the dimensions of the subblocks. Puzzles will always be square (eg. of length 81, 100, 144, 256) where the side length of the puzzle, `N`, will be a composite integer no larger than 25. From your slider puzzle days, you already have code to determine the subblock size. (eg. If the puzzle length is 144, the puzzle size is 12, and the subblock size is 3 high by 4 wide).

2B) `setGlobals()` should also determine the symbol set, `SYMSET`, of the puzzle. Not all puzzles include all N of their symbols. In those cases, your code should supply the missing symbol. The following conventions usually apply for the indicated N:

9:	{1-9}
12:	{1-9} ∪ {A-C}
16:	{0-9} ∪ {A-F} OR {1-9} ∪ {A-G}

2C) Determine the N constraint sets for rows, the N constraint sets for columns, and the N constraint sets for subblocks. A constraint set is a set of index positions such that no two squares at any of the index positions may have the same tile. Put each of the constraint sets into a global list (ie. a list of constraint sets). The constraint sets for the columns and rows will be straightforward, but for the subblocks it is a bit trickier, so make certain that you have them correct. A puzzle size of `N=6` may be useful for testing.

3) Write an `isInvalid(pzl)` function utilizing the list of constraint sets from the prior part

4) On each iteration through the main loop, feed the sudoku puzzle to `bruteForce()` and then display the results. Identify the puzzle number (starting from 1), the solved puzzle a check sum (see below), and the time it took to solve it.

5) Write a simple `checkSum(pzl)` function: determine the minimum ASCII value, m, over all symbols, and then sum the ascii value less m for each symbol in your puzzle. Do this as a simple way of double checking your Sudoku solution correctness – output this number next to your Sudoku solutions.

Your final output should resemble the following:

```
…
 9: 81.976532.5.1234 … 7439621426517983
    8149765326591234 … 7439621426517983 324 0.0s
10: 8..976532.5.1234 … 7439621426517983
    8149765326591234 … 7439621426517983 324 0.0s
11: ...976532.5.1234 … 7439621426517983
    8149765326591234 … 7439621426517983 324 0.0s
12: ..3.2.6..9..3.5. … .2.3..9..5.1.3..
    4839216579673458 … 4253769695417382 324 0.0s
13: 2...8.3...6..7.. … ..4..6...4.1...3
    2459813761692735 … 8349165654812793 324 0.0s
…
```

The puzzle number starts at 1, and notice that the solution puzzle is exactly aligned under the original puzzle.