Problem 1

Goal

Work with exploring state space.

Background

Games are a domain in which the player searches through a set of possible local solutions to find an answer that satisfies all the puzzle's constraints.

A Mathdoku puzzle(see http://www.mathdoku.com) has some similarities with a sudoku puzzle.

In short, you are given a square n x n grid. Within the grid, each cell is identified as part of some grouping. Each grouping is a connected set of 1 or more cells and each grouping is given an operator and a result of the operator. Figure 1 shows a sample puzzle.

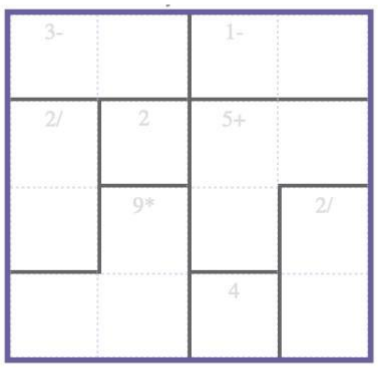

Figure 2 Sample Mathdoku puzzle from mathdoku.com

The task is to put the integers 1 to n into the cells of the grid so that:

- No integer appears twice in any row
- No integer appears twice in any column
- Applying the operator to all the values in one grouping gives the result assigned to the operator.

Valid operators are + for addition, - for subtraction, * for multiplication, / for division, and = to specify that a grouping (of one cell) has a given value. When applying operators, all results must be integers. When applying the – and / operators, the larger integer is always the leftmost operand of the operation.

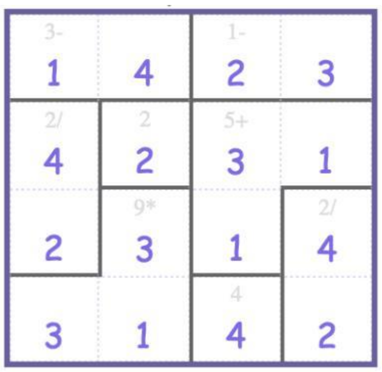

Figure 1 Solution to the Mathdoku puzzle from Figure 1.

Figure 2 shows a solution to the puzzle in Figure 1.

These puzzles also appear under the name of kenken (https://www.kenkenpuzzle.com).

TO DO:

Write a class called “Mathdoku” that accepts a puzzle and ultimately solves the puzzle.

The class has at least 5 methods:

- boolean loadPuzzle(BufferedReader stream) – Read a puzzle in from the given stream of data. Further description on the puzzle input structure appears below. Return true if the puzzle is read. Return false if some other error happened.
- boolean readyToSolve( ) – determine whether or not we have all the information needed to be able to try and solve the puzzle. If you have other tests to see if the puzzle is at all solvable, test them too. Return true if we can call solve( ) and expect a meaningful attempt at solving the puzzle. Return false if there is something missing or amiss with the puzzle that is obvious (so not needing to solve the puzzle) and that would prevent a solution. For example, if one of the groupings does not have a constraint (operation or result) then we cannot solve the puzzle.
- boolean solve( ) -- Do whatever you need to do to find a solution to the puzzle. The solution is stored within the class, ready to be retrieved. Return true if you solved the puzzle and false if you could not solve the puzzle with the given set of words.
- String print( ) – print the current puzzle state to the returned string object.
- int choices( ) – return the number of guesses that your program had to make and later undo while solving the puzzle.

You get to choose how you will represent the puzzle in your program and how you will proceed to solve the puzzle.

Inputs

The loadPuzzle( ) method will accept a description of the puzzle as an input stream. The input stream will have 2 parts to it.

- Part 1: Consists of the puzzle square itself. For an n x n puzzle square, the input will have n lines of input, each being a string of n characters (excluding the end of the line). For one line, the n characters are the n cells in the line; each cell is a letter that represents the cell grouping to which the cell belongs.
- Part2: Consists of the constraints for each cell grouping in the puzzle. There will be one line for each cell grouping. That line will have 3 values: the letter representing the grouping, the operation outcome for the grouping, and the operator for the grouping (one of +, -, *, /, or =). The values are separated by one another by at least one space.

Example: The puzzle in Figure 1 would be represented as the following input. The constraints are given alphabetically by the grouping name here, but that's not a guarantee in all input.

```
aabb
cdee
cfeg
ffhg
a 3 –
b 1 –
c 2 /
d 2 =
e 5 +
f 9 *
g 2 /
h 4 =
```

Outputs

The print( ) method produces a String that can later be printed. The output provides each row of the current puzzle state; listed from top-to-bottom and rows separated by a carriage return (\n) character. No space is printed between the columns of the rows. If the cell has a value from 1-n assigned to it then that's the value printed for the cell. Otherwise, print the grouping letter for the cell.

The following is the returned string for the puzzle in Figure 2, noting that \n should be seen as just one character in the text below:

```
1423\n4231\n2314\n3142\n
```

Assumptions

You may assume that

- The character for each cell grouping is case sensitive. So, a cell entered as z and another as Z are two different groupings.