

Problem Set 4

Specification

In this problem set, the problems are a bit longer. You may write each problem in its own file and submit a zipped folder of all the files. I suggest you name the files something meaningful, and your zipped file should have your name.

A note about printing to the console. When printing the puzzle, keep in mind that vertical whitespace is two times as large as horizontal whitespace in the terminal.

2D Lists as Matrices

Problem 1

Write a method that takes two 2D arrays and returns the sum of the two arrays. You may assume that the 2D arrays behave as matrices.

```
// a = {{1, 2}, {1, 3}}, b = {{4, 5}, {2, 1}}
addMatrices(a, b) // {{5, 7}, {3, 4}}
```

Problem 2

Write a method that takes two 2D arrays and returns the multiplication of the two arrays. You may assume that the 2D arrays behave as matrices. You do not need to check the inputs to ensure they can be multiplied. `mulMatrices(a, b)` returns AB , not BA .

```
// a = {{1, 2}, {1, 3}}, b = {{4, 5}, {2, 1}}
mulMatrices(a, b) // {{8, 7}, {10, 8}}
```

Problem 3

The *transpose* of a matrix is found by swapping all elements a_{ij} to a_{ji} . The transpose is usually denoted with a superscript T . Consider the following

matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

The transpose is $A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

Write a function that takes a 2D array A and returns A^T .

```
// a = {{1, 2}, {3, 4}}
transposeMatrix(a) // {{1, 3}, {2, 4}}
```

Problem 4

A *diagonal* matrix is one whose off-diagonal elements are all zero. In other words, $a_{ij} = 0$ for $i \neq j$. Write a function that takes a 2D array representing a matrix and returns `true` if the matrix is diagonal and `false` otherwise.

```
// a = {{1, 2}, {3,4}}
isDiagonal(a) // false
// b = {{1, 0, 0}, {0, 3, 0}, {0, 0, 10}}
isDiagonal(b) // true
```

2D Lists as Grids

In the next few problems we will look at random walkers on a grid. A random walker is an agent that moves on a grid either up, down, left, or right with equal probability. They are useful models for systems ranging from ink in a dish, biological molecule binding, even chemical polymerization, cereal floating around in a bowl, and much more. For the following questions you will assume that a random walker begins at the origin $(0, 0)$ of a box that goes from $(-n, -n)$ to (n, n) . In other words, the box has side lengths $2n$.

Problem 5

One particular thing of interest is how many steps it takes for a random walker to leave the box. Write a method that takes a single parameter n and lets the walker go until it leaves the box. The method returns a 2D String array where an explored site is a # and an unexplored site is a space.

Problem 6

Using a Monte Carlo simulation, write a method that returns the number of steps expected for a random walker to leave a box given a size n .

Problem 7

Now assume that the random walker is confined to the box, e.g. there are walls on the box. Write a method that returns the number of steps expected for a random walker to explore every site in the box. Banging against the walls counts as a step. Again, n is a parameter.

Problem 8

A self-avoiding random walk is similar to a random walk, but the same site may not be visited twice. Write a method that takes a single parameter n and lets the walker go until it leaves the box, or gets stuck in a dead end. The method returns a 2D String array where an explored site is a # and an unexplored site is a space.

Problem 9

Using a Monte Carlo simulation, write a method that returns the probability that a self-avoiding random walker escapes the box, given a size n .

2D Lists as Game Boards

Problem 10

Create a fully functional game of tic-tac-toe for two players. The board should be printed after every turn. The display should say the current player's turn and show open/occupied spaces with X's and O's. The game should automatically determine a winner. Find the rules here:

<https://en.wikipedia.org/wiki/Tic-tac-toe>.

Problem 11

A sliding puzzle is a mechanical puzzle in which a player tries to rearrange tiles on a grid to recreate a picture, or reorder numbers. Create an eight puzzle that is automatically scrambled (and is solvable). Let the player use input to play the game and produce output when the puzzle is correctly solved. Find an example of the puzzle here: https://en.wikipedia.org/wiki/Sliding_puzzle

Problem 12

2048 is a sliding puzzle number game. The objective of the game is to slide numbered tiles on a grid to combine them to create a tile with the number 2048. However, a player can continue to play the game after reaching the goal. In our implementation, a player wins if they create a 2048 tile. The game is played on a 4×4 grid with numbered tiles that slide when a player moves them using the arrow keys. Every turn, a new tile randomly appears in an empty spot on the board with a value of either a 2 or a 4. Tiles slide as far as possible in the chosen direction until they are either stopped by another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. That resulting tile cannot merge with another tile in the same move. If a move causes three consecutive tiles of the same value to slide together, only the two tiles farthest along the direction of motion will combine. The score is calculated by increasing when two tiles combine, by the value of the new tile. The game ends when the player has no legal moves. Implement a playable terminal game of 2048. Find the rules here:

[https://en.wikipedia.org/wiki/2048_\(video_game\)](https://en.wikipedia.org/wiki/2048_(video_game))

Problem 13

Sudoku is a number placement puzzle played on a 9x9 grid. A puzzle is solved if each row, column, and 3x3 sub-block contain the numbers 1 through 9 only once. Write a program that takes a Sudoku puzzle from standard input and lets the user specify a row/column and number to play the game. Initialize empty spaces to 0's. Play a congratulatory message when the player wins. Find the rules here: <https://en.wikipedia.org/wiki/Sudoku>. The sudoku puzzle 1.puz is shown below:

```
0 0 6 0 0 0 5 0 8
1 0 2 3 8 0 0 0 4
0 0 0 2 0 0 1 9 0
0 0 0 0 6 3 0 4 5
0 6 3 4 0 5 8 7 0
5 4 0 9 2 0 0 0 0
0 8 7 0 0 4 0 0 0
2 0 0 0 9 8 4 0 7
4 0 9 0 0 0 3 0 0
```

Problem 14

The goal of this question is to represent a crossword puzzle app. A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the crossword labeling rule.

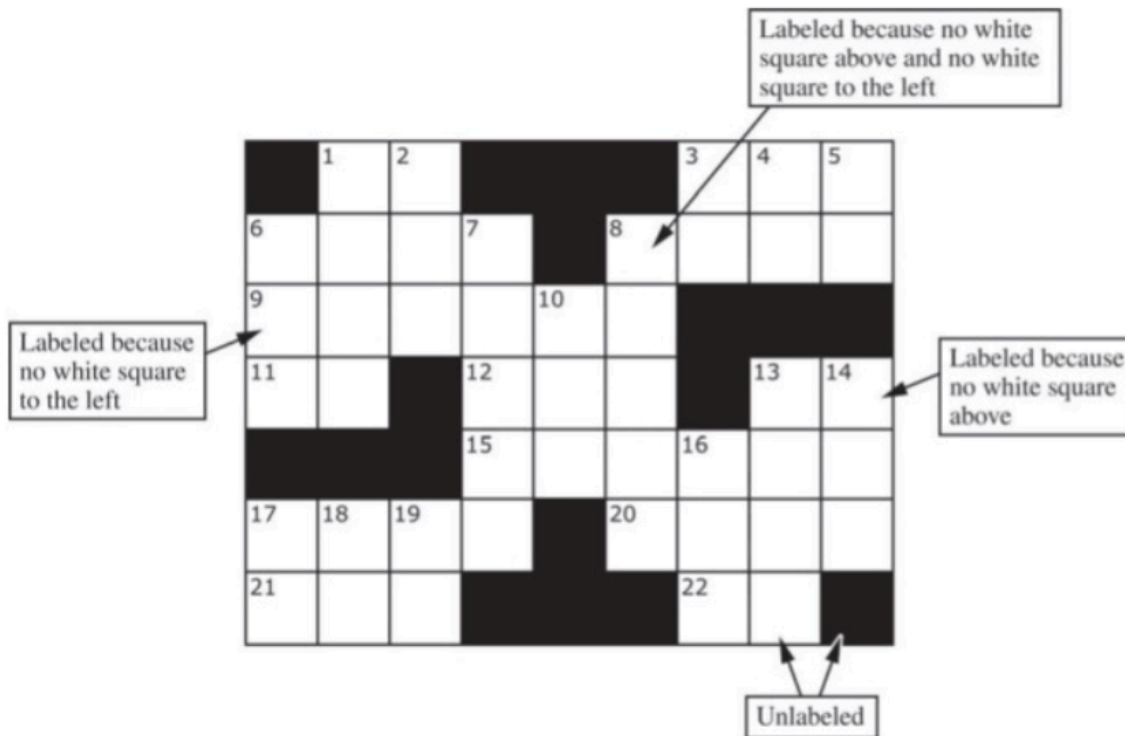
The crossword labeling rule identifies squares to be labeled with a positive number as follows.

A square is labeled with a positive number if and only if

- the square is white and
- the square does not have a white square immediately above it, or it does not have a white square immediately to its left, or both.

The squares identified by these criteria are labeled with consecutive numbers in row-major order, starting at 1.

The following diagram shows a crossword puzzle grid and the labeling of the squares according to the crossword labeling rule.



The `Crossword` class has the following API:

- `private Square[][] puzzle` : stores a crossword puzzle
- `public Crossword(boolean[][] blackSquares)` : Constructs a crossword puzzle grid
- `private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)` : returns true if row/col of puzzle should be labeled
- `public String toString()` : prints crossword puzzle

The `Square` class has the following API:

- `private boolean isBlack` : is a square black?
- `private int num` : stores the number of a square
- `public Square(boolean isBlack, int num)` : initialize a square
- `public String toString()` : either a number, hash, or whitespace

Write a program that create a correctly numbered crossword puzzle grid, given a 2d `boolean[][]` array of black squares.

Problem 15

KenKen are a style of arithmetic logic puzzle invented by Tetsuya Miyamoto who intended for the puzzles to be an instruction-free method of training the brain. The puzzles are constructed on an $n \times n$ grid. The rules of the puzzle are simple:

- Each row must contain exactly one instance of $1, \dots, n$
- Each column must contain exactly one instance of $1, \dots, n$
- The digits in each of the heavily outlined groupings must combine to give the result indicated within the grouping when the operation indicated is applied to the operands.
- Subtraction and Division may only have two operands, but multiplication and addition can have more.

An example puzzle and solution are shown below.

11+	2÷		20×	6×	
	3-			3÷	
240×		6×			
		6×	7+	30×	
6×					9+
8+			2÷		
11+	2÷		20×	6×	
5	6	3	4	1	2
6	1	4	5	2	3
240×		6×			
4	5	2	3	6	1
		6×	7+	30×	
3	4	1	2	5	6
6×					9+
2	3	6	1	4	5
8+			2÷		
1	2	5	6	3	4

Part 1 of this question requires you to take the puzzle as a text file where each line contains the coordinates of grouped cells and the operation to be applied to the values in the cells, followed by the expected result. The values on the line should be separated by a space. The first cell provided is the one that is labeled. You may assume that no input is larger than 4 characters (3 digits plus 1 operator). The first line is a single integer determining the size of the grid. The second line is a single integer determining the number of groups in the grid. Example input for the example puzzle is shown below. For the first part of the question, write a method `isSolved()` that takes a KenKen puzzle as a 2D int array, and returns `true` if the puzzle is correctly solved, and `false` otherwise.

```
kenken1.puz
6
15
0 0 0 1 + 11
0 1 0 2 / 2
0 3 1 3 * 20
0 4 0 5 1 5 2 5 * 6
1 1 1 2 - 3
1 4 2 4 / 3
2 0 2 1 3 0 3 1 * 240
2 2 2 3 * 6
3 2 4 2 * 6
3 3 4 3 4 4 + 7
3 4 3 5 * 30
4 0 4 1 * 6
4 5 5 5 + 9
5 0 5 1 5 2 + 8
5 3 5 4 / 2
```

A bonus part 2 of this question is to create a playable terminal implementation of a KenKen puzzle. Your code should take input for a puzzle as shown above, display the puzzle in its current state to the user, check for correctness, and inform the user when the puzzle is correctly solved. When printing the puzzle, keep in mind that vertical whitespace is two times as large as horizontal whitespace in the terminal. It helps to consider each block as larger than 1×1 so that there is enough space to enter numbers and it does not interfere with the labeling. Player input is of the form `row col value` to enter a value into a given cell. Find the rules for KenKen here: <https://en.wikipedia.org/wiki/KenKen>