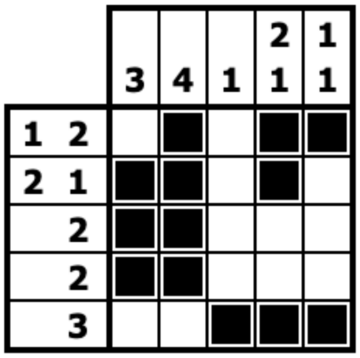
CSE 202 Project: Nonogram Solver Implementation

Brian Tsao, Darren Yeung, Zhichao Liu, Jihu Mun, Shihui Wang

1 Project Summary

The game we will be attempting to solve computationally will be Nonogram. Nonogram is a picture logic puzzle game with squares that must be either filled or left blank according to the numbers at the side and top of the grid.

Example of a solved Nonogram puzzle:



The numbers next to each column and row are clues to the puzzle. For example, for the first row, there is "1 2". This means that from left to right, the first row must have 1 consecutive filled squares, one or more blank squares, and then 2 consecutive filled squares. The puzzle is solved when all of these clues are satisfied. We will refer to a single cell as a **square**, and a consecutive sequence of filled squares as a **block**.

2 Problem Specification

Input:

- The dimensions of the puzzle (n, m), where n is the number of rows and m is the number of columns
- The clues for the puzzle $(R_1, \ldots, R_n)$ and $(C_1, \ldots, C_m)$
  - For each row clue $R_i = (r_{i1}, \ldots, r_{ik})$, $\left(\sum_{j=1}^{k} r_{ij}\right) + k - 1 \leq m$.
  - For each column clue $C_i = (c_{i1}, \ldots, c_{ik})$, $\left(\sum_{j=1}^{k} c_{ij}\right) + k - 1 \leq n$.

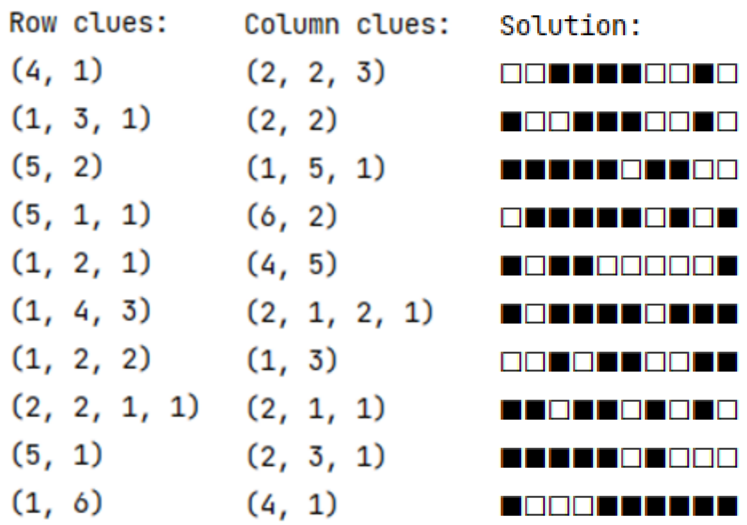
Output:

- A binary matrix ($n \times m$) that represents a solution for the puzzle. 1 represents a filled square and 0 represents a blank square. The matrix must satisfy the following:
  - For each row i and the row clue $(r_{i1}, \ldots, r_{ik})$, from left to right, there must be r_{i1} consecutive filled squares, one or more blank squares, r_{i2} consecutive filled squares, and all the way until r_{ik} consecutive filled squares.
  - For each column i and the column clue $(c_{i1}, \ldots, c_{ik})$, from top to bottom, there must be c_{i1} consecutive filled squares, one or more blank squares, c_{i2} consecutive filled squares, and all the way until c_{ik} consecutive filled squares.
- If there is no valid solution for the puzzle, the algorithm will output "unsolvable".

3 Implementation

We implemented a backtracking algorithm shown in our project algorithm design document. The implementation was done in Python 3.9.2 and run on an Intel i7-9750H CPU.

The following example is a 10-by-10 Nonogram. The inputs were the row and the column clues, and the output was the solution. It can be seen that the solution satisfies all of the clues.



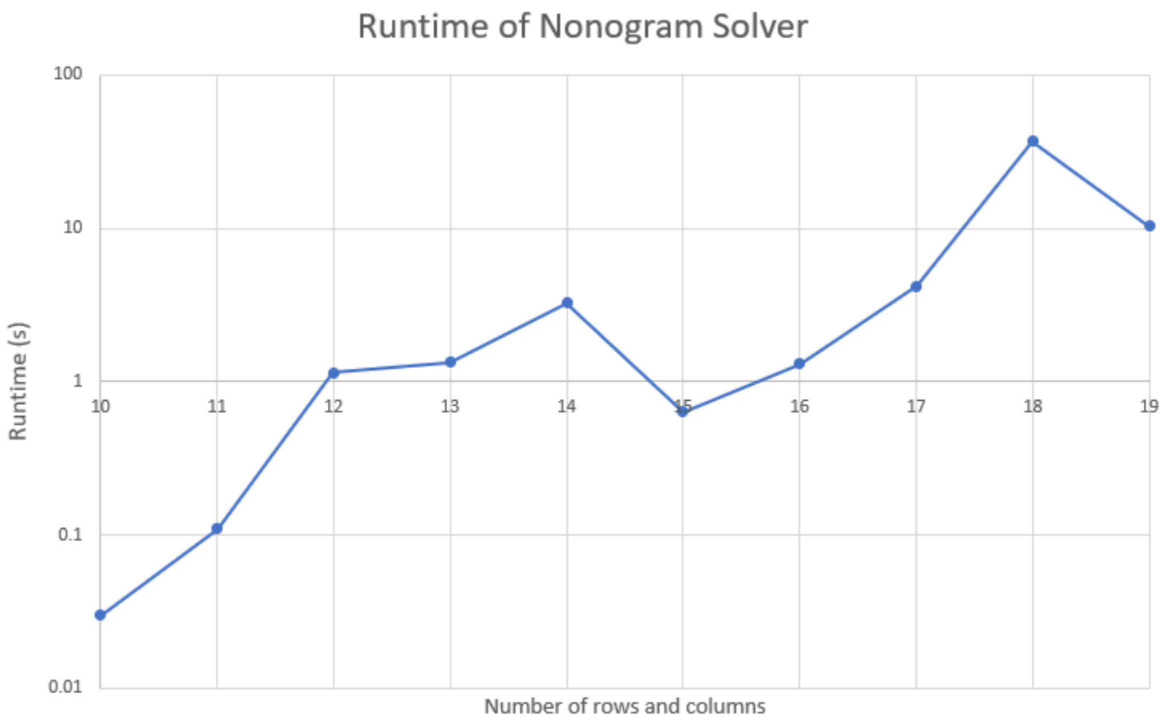
```
Row clues:      Column clues:   Solution:
(4, 1)          (2, 2, 3)       □□■■■■□□■□
(1, 3, 1)       (2, 2)          ■□□■■■□□■□
(5, 2)          (1, 5, 1)       ■■■■■□■■□□
(5, 1, 1)       (6, 2)          □■■■■■□■□■
(1, 2, 1)       (4, 5)          ■□■■□□□□□■
(1, 4, 3)       (2, 1, 2, 1)    ■□■■■■□■■■
(1, 2, 2)       (1, 3)          □□■□■■□□■■
(2, 2, 1, 1)    (2, 1, 1)       ■■□■■□■□■□
(5, 1)          (2, 3, 1)       ■■■■■□■□□□
(1, 6)          (4, 1)          ■□□□■■■■■■
```

If the puzzle is unsolvable, such as the number of total squares specified by the row clues and the column clues being different, the program does not output any solution.

4 Results

We tested the program on random Nonograms where 50 percent of the squares were filled. To make the runtimes more stable, we made the puzzle unsolvable by decrementing the bottommost row clue by 1. This way, the program goes through the entire board before deciding that there is no solution. However, even after doing that, the runtimes were wildly inconsistent. This is because some puzzles have a set of clues that make it harder to eliminate cases than in other puzzles.

The following is the result of running the program on Nonograms of size 10-by-10 to 19-by-19.



In this graph, the y-axis is in a logarithmic scale, and the graph is roughly linear. Therefore, it can be seen that the runtime is an exponential function of the number of rows and columns.