CSCI-1200 Data Structures — Spring 2022
Homework 6 — Star Battle

In this homework we will work with a variation on Star Battles, however we will modify one of the rules. As such, you should read the entire handout carefully. Star Battles in various forms have seen some popularity, so there is lots of material available online. You may not search for, study, or use any outside **code** related to Star Battles (or variants of Star Battles). You are welcome to find additional puzzles to experiment with and try to solve by hand. Some examples of puzzles are "Trees" (S=1, uses trees instead of stars as a symbol), "Two Not Touch" (S=2), and more generally "Star Battles" (S=1, S=2, S=3 examples).

The basic goal is to consider a two dimensional grid in which each square belongs to exactly one contiguous group of squares. We will refer to these groups as *zones*. Some puzzles use background colors to make this more clear, we will use single letter labels for our zones (see Input File) below. The goal in our version is that for a positive integer S, we should place stars (we will use @ instead) such that:

1. Each zone should have exactly S stars.
2. No two stars can be adjacent (touching), not even diagonally!
3. Each row can have up to S stars in it.
4. Each column can have up to S stars in it.

Star Battle Arguments

Your program will accept five command line arguments.

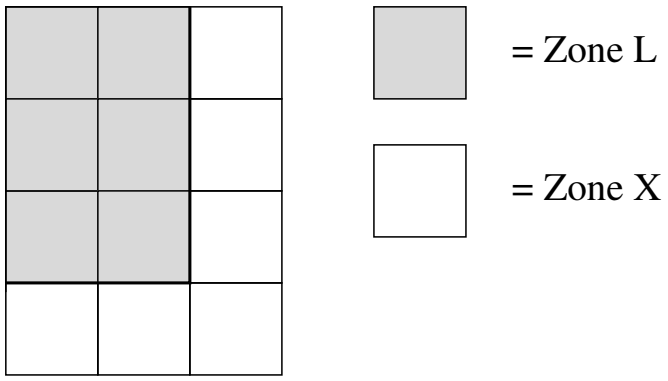
Execution looks like: `./a.out [input file] [output file] [stars per zone] [output mode] [solution mode]`

Input File

The input file describes the puzzle. The first two integers are the number of rows and then number of columns. Next that, we describe one zone at a time by first giving the label (letter) for zone and the number of squares in the zone. After each label and square count, the file contains a series of (x,y) coordinates for the zone. We define $(0, 0)$ as the top left of the puzzle. You can assume zones always have at least 1 square, and that every input has at least 1 zone.

The input file `custom1.txt` and a visualization of the puzzle's initial state is:

```
4 3
L 6
0 0
1 0
0 1
1 1
0 2
1 2
X 6
2 0
2 1
2 2
0 3
1 3
2 3
```



Output Mode

The output mode will either be *count* in which case you will only print the number of solutions you found (just the first line of output from the example in the section below), or *print* in which case you should print the count and print all solutions.

Solution Mode

The solution mode will either be *one_solution* meaning you should only find up to one solution, or it will be *all_solutions* meaning you should find all solutions that satisfy the inputs.

Output Formatting

Partial output for the `custom1.txt` puzzle with $S = 1$ is:

```
Number of solutions: 17

Solution 1:
LLX
LL@
@LX
XXX

Solution 2:
LLX
L@X
LLX
XX@

[see out_custom1_1_all.txt for full output]
```

To ensure full credit on the homework server, please format your solution exactly as shown above. Solutions may appear in any order, but the first line must start with `Number of solution(s):` then a space and the number of solutions. Each solution should start with a line that starts with `Solution` followed by one row of the grid per line, with $(0, 0)$ being located in the top left. For any square that has a star, you should print @. For all other squares, you should print the letter of the zone the square belongs to.

Additional Requirements: Recursion, Order Notation, & Extra Puzzles

You must use recursion in a non-trivial way in your solution to this homework. As always, we recommend you work on this program in logical steps. Partial credit will be awarded for each component of the assignment. Your program should do some error checking when reading in the input to make sure you understand the file format. *IMPORTANT NOTE: This problem is computationally expensive, even for medium-sized puzzles with too much freedom! Be sure to create your own simple test cases as you debug your program.*

To help with runtime, your program should be written to do the three following things (also discussed in the HW6 Discussion lecture videos):

1. Start with the smallest zones first, and work up to the bigger zones in a puzzle
2. Stop considering a partial solution if we've seen it before (keep a history of states)
3. Stop considering a partial solution if it is no longer solveable

To get started, we recommend starting by solving `sporcle1.txt` first, since for $S = 1$, only one solution exists. After that, you may either want to move to `twonot1.txt` to handle $S > 1$, or you may want to move to `custom1.txt` (Still $S = 1$ but has many solutions).

Once you have finished your implementation, analyze the performance of your algorithm using order notation. What important variables control the complexity of a particular problem? In your `README.txt` file write a concise paragraph (< 200 words) justifying your answer. Also include a simple table summarizing the running time and number of solutions found by your program on each of the provided examples.

You should include 1-3 new puzzles that either helped you test corner cases or experiment with the running time of your program. Make sure to describe these puzzles in your README.

You can use any technique we have covered in Lectures 1-14, Homework 1-5, and Lab 1-7. This means you cannot use STL `pair`, `map`, `set`, etc. on this homework assignment.

You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.

NOTE: If you earn 7 points on the homework submission server for tests 3 through 10 by 11:59pm on Wednesday, March 16, you will earn a one day extension on HW6.