Expanding Nebula
================

You've escaped Commander Lambda's exploding space station along with numerous escape pods full of bunnies. But - oh no! - one of the escape pods has flown into a nearby nebula, causing you to lose track of it. You start monitoring the nebula, but unfortunately, just a moment too late to find where the pod went. However, you do find that the gas of the steadily expanding nebula follows a simple pattern, meaning that you should be able to determine the previous state of the gas and narrow down where you might find the pod.

From the scans of the nebula, you have found that it is very flat and distributed in distinct patches, so you can model it as a 2D grid. You find that the current existence of gas in a cell of the grid is determined exactly by its 4 nearby cells, specifically, (1) that cell, (2) the cell below it, (3) the cell to the right of it, and (4) the cell below and to the right of it. If, in the current state, exactly 1 of those 4 cells in the 2x2 block has gas, then it will also have gas in the next state. Otherwise, the cell will be empty in the next state.

For example, let's say the previous state of the grid (p) was:

.O..

..O.

...O

O...

To see how this grid will change to become the current grid (c) over the next time step, consider the 2x2 blocks of cells around each cell. Of the 2x2 block of [p[0][0], p[0][1], p[1][0], p[1][1]], only p[0][1] has gas in it, which means this 2x2 block would become cell c[0][0] with gas in the next time step:

.O -> O

..

Likewise, in the next 2x2 block to the right consisting of [p[0][1], p[0][2], p[1][1], p[1][2]], two of the containing cells have gas, so in the next state of the grid, c[0][1] will NOT have gas:

O. -> .

.O

Following this pattern to its conclusion, from the previous state p, the current state of the grid c will be:

O.O

.O.

O.O

Note that the resulting output will have 1 fewer row and column, since the bottom and rightmost cells do not have a cell below and to the right of them, respectively.

Write a function solution(g) where g is an array of array of bools saying whether there is gas in each cell (the current scan of the nebula), and return an int with the number of possible previous states that could have resulted in that grid after 1 time step. For instance, if the function were given the current state c above, it would deduce that the possible previous states were p (given above) as well as its horizontal and vertical reflections, and would return 4. The width of the grid will be between 3 and 50 inclusive, and the height of the grid will be between 3 and 9 inclusive. The answer will always be less than one billion (10^9).

Languages

=========

To provide a Java solution, edit Solution.java

To provide a Python solution, edit solution.py

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Java cases --

Input:

Solution.solution({{true, true, false, true, false, true, false, true, true, false}, {true, true, false, false, false, false, true, true, true, false}, {true, true, false, false, false, false, false, false, false, true}, {false, true, false, false, false, false, true, true, false, false}})

Output:

```
    11567
```

Input:

Solution.solution({{true, false, true}, {false, true, false}, {true, false, true}})

Output:

```
    4
```

Input:

Solution.solution({{true, false, true, false, false, true, true, true}, {true, false, true, false, false, false, true, false}, {true, true, true, false, false, false, true, false}, {true, false, true, false, false, false, true, false}, {true, false, true, false, false, true, true, true}}

Output:

```
    254
```

-- Python cases --

Input:

solution.solution([[True, True, False, True, False, True, False, True, True, False], [True, True, False, False, False, False, True, True, True, False], [True, True, False, False, False, False, False, False, False, True], [False, True, False, False, False, False, True, True, False, False]])

Output:

```
    11567
```

Input:

solution.solution([[True, False, True], [False, True, False], [True, False, True]])

Output:

```
    4
```

Input:

solution.solution([[True, False, True, False, False, True, True, True], [True, False, True, False, False, False, True, False], [True, True, True, False, False, False, True, False], [True, False, True, False, False, False, True, False], [True, False, True, False, False, True, True, True]])

Output:

```
    254
```

Use verify [file] to test your solution and see how it does. When you are finished editing your code, use submit [file] to submit your answer. If your solution passes the test cases, it will be removed from your home folder.