DonutsOnTheGridEasy

Problem Statement

Petya likes donuts. He tries to find them everywhere. For example if he is given a grid where each cell contains a '0' (zero) or '.' he will construct donuts from the cells.

To make the donuts:

1. Petya first selects a rectangle of cells of width, w, and height, h, where both are at least 3.
2. Then he removes a rectangular hole of width w-2 and height h-2 centered in the larger rectangle.
3. For the remaining cells (a closed rectangular ring) to be a valid donut, all of the cells must contain '0' (because donuts are shaped like zeros). Cells in the hole can contain anything since they are not part of the donut.

An example of large donut containing a smaller donut.

```
..........
.00000000.
.0......0.
.0.0000.0.
.0.0..0.0.
.0.0..0.0.
.0.0000.0.
.0......0.
.00000000.
..........
```

Donuts may contain other donuts and donuts may touch, but the cells of one donut may not overlap the cells of another. Petra is most interested in donuts that contain other donuts. He first places one valid donut on the grid (if possible). He then places a valid donut in the hole of the first donut (if possible). He continues to place a donut into the hole of the most recently placed donut until this is no longer possible.

Your task is to find the **maximum** number of donuts that Petya can place on the grid as described such that each donut (except for the first) is contained within the previous donut.

Input

First line contains an integer n (n<200), indicating number of testcases.
In every testcase, it starts with a integer r (r<=50), indicating the number of rows of that grid.
Followed by r strings that only contain ‘0’ or ‘.’. The length of the string is <= 50.
The length ofstrings in same testcase will be the same.

Output

For every test case, print a line of the form “Case X: Y” where X is the serial number of output (starting from 1). Y is the maximum number of donuts that Petya can place on the grid

Sample Input and output

Input:

```
5
7
0000000
0.....0
0.00000
0.0..00
0.00000
0.....0
0000000
3
000
0.0
000
3
...
...
...
4
00.000
00.000
0.00.0
000.00
7
0000000....
0000000....
0000000.000
0000000.0.0
0000000.000
0000000....
0000000....
```

Output:

```
Case #1: 2
Case #2: 1
Case #3: 0
Case #4: 0
Case #5: 3
```