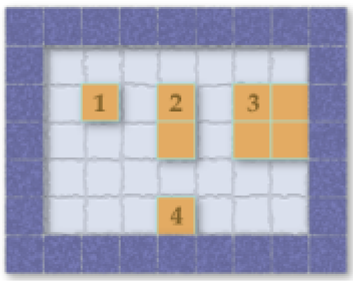
Code Jam 2009 - Round 3

EZ-Sokoban

Problem

Sokoban is a famous Japanese puzzle game. Sokoban is Japanese for "warehouse keeper". In this game, your goal is to push boxes to their designated locations in the warehouse. To push a box, the area behind the box and in front of the box must be empty. This is because you stand behind the box when pushing and you can push only one box at a time. You cannot push a box out of the board and you cannot stand outside the board when pushing a box.

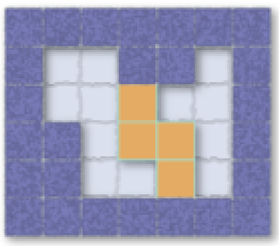
For example, in this picture:



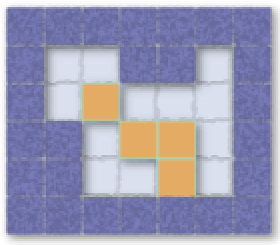
Box 1 can be pushed in any of the four directions because the four spaces adjacent to it are empty. Box 2 can only be pushed east or west; it cannot be pushed north or south because the space to its south is not empty. Box 3 cannot be pushed in any direction. Box 4 can only be pushed east or west because there is a wall south of it.

Sokoban was proved to be a P-Space complete problem, but we deal with an easier variation here. In our variation of Sokoban, boxes have strong magnets inside and they have to stick together *almost* all the time. Under "stable" conditions, all boxes should be connected, edge to edge. This means that from any box we can get to any other box by going through boxes that share an edge. If you push a box and boxes are no longer connected, you are in "dangerous mode". In dangerous mode, the next push must make the boxes connected again.

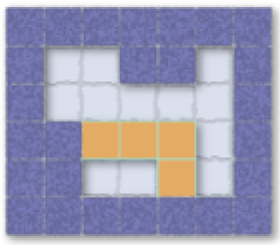
For example, in this picture:



The situation is stable, since all 4 boxes are connected, edge to edge. Let's assume that you decided to push the northmost box west:



Now, we are in dangerous mode since the northmost box is not connected to any other boxes. The next push must return us to a stable position. For example, we can push that northmost box south:



Making the boxes stable again.

A Sokoban puzzle consists of a board, initial configuration of the boxes and the final configuration (where we want the boxes to be at the end). Given an EZ-Sokoban puzzle, find a solution that makes the minimum number of box moves, or decide that it can't be solved. The final and initial configurations will not be in "dangerous" mode.

To simplify things, we will assume that you, the warehouse keeper, can jump at any time to any empty spot on the board.

Input

The first line in the input file contains the number of cases, **T**.

Each case consists of several lines. The first line contains **R** and **C**, the number of rows and columns of the board, separated by one space. This is followed by **R** lines. Each line contains **C** characters describing the board:

- '.' is an empty spot
- '#' is a wall
- 'x' is a goal (where a box should be at the end)
- 'o' is a box
- 'w' is a both a box and a goal

The number of boxes will be equal to the number of goals.

Output

For each test case, output

```
Case #X: K
```

where **X** is the test case number, starting from 1, and **K** is the minimum number of box moves that are needed to solve the puzzle or **-1** if it cannot be solved.

Limits

Memory limit: 1 GB.

1 ≤ **T** ≤ 50
1 ≤ **R**,**C** ≤ 12

Small dataset

Time limit: 30 seconds.
1 ≤ the number of boxes ≤ 2

Large dataset

Time limit: 45 seconds.
1 ≤ the number of boxes ≤ 5

Sample

Sample Input

```
4
5 4
....
#..#
#xx#
#oo#
#..#
7 7
.######
.x....#
.x....#
..#oo.#
..#...#
.######
.######
4 10
##########
#.x...o..#
#.x...o..#
##########
3 4
.#x.
.ow.
....
```

Sample Output

```
Case #1: 2
Case #2: 8
Case #3: 8
Case #4: 2
```