Wormhole

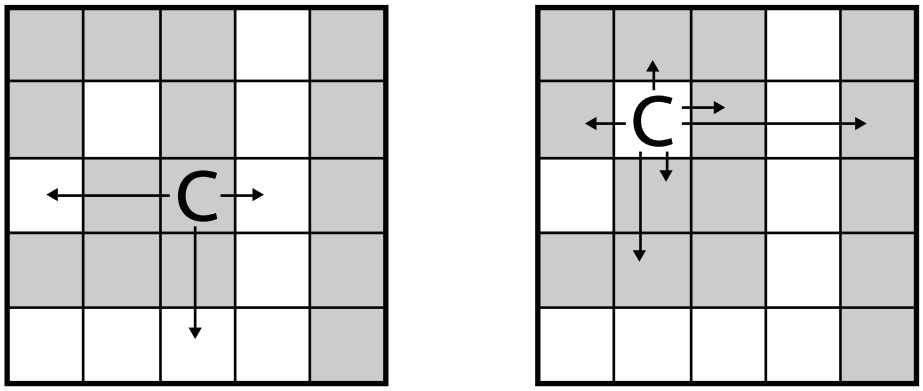
Time Limit: 1 second

You are the lead programmer of *Wormhole*, a widely anticipated computer game. The premise of Wormhole is rather thrilling: test subject Chaz is trapped in a rectangular grid of orange and blue chambers each containing one cupcake. Chaz chooses a chamber to start in, and then needs to collect all the cupcakes by visiting every chamber at least once. If this is possible the level is considered *solvable*.

The chambers are walled off from each other by thick glass, such that Chaz can only see chambers which are directly north, south, east or west of his current position. Conveniently Chaz has a 'wormhole gun', which he can use to shoot through the glass into any chamber he can see and so be instantly teleported there. However, a wormhole can only be created between two chambers of different colours. This means that if Chaz is standing in a blue chamber, he can only move to an orange chamber, and vice versa.

That is, Chaz can teleport to another chamber if:

- It is of a *different colour* to his current chamber.
- It is *directly north, south, east or west* (i.e. in the same row or column) of his current chamber.



Arrows indicate possible teleports Chaz can make. From those chambers he may continue to teleport to other chambers.

You have been stuck in a meeting for hours now where the designers are interested in how *modifications* to the chamber colours affect a level. Each modification involves changing the colour of one chamber. After each change they make, the designers want to know how many more modifications are needed to make the level solvable – that is, the smallest number of chambers that need to have their colour changed so that Chaz can reach every chamber from a starting chamber of his choosing.

As working everything out by hand gets more and more tedious, you escape from the meeting and whip out your trusty laptop to write a computer program that will answer the designers' questions after each modification.

Input

Your program should read from the file .

- The first line of input contains three space-separated integers, R, C, and Q. There will be R rows and C columns in the grid, and Q modifications made by the designers.

- The next R lines each contain C characters, representing the initial design of the game. An '`O`' represents an orange chamber, while a '`B`' represents a blue chamber.
- The following Q lines each contain two integers r_i and c_i. This represents a modification to the grid that changes the colour of the chamber in row r_i and column c_i. Rows are labelled from 1 to R (from top to bottom) and columns are labelled from 1 to C (from left to right).

Output

Your program should write to the file . The output should contain $Q + 1$ lines. The first line should contain the smallest number of modifications required to make the level solvable. The next Q lines should correspond to the Q modifications in input. After each modification has been made (including all previous modifications), your program should write one line containing the smallest number of modifications required to make the level solvable (or `0` if it is already solvable).

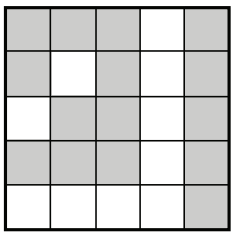
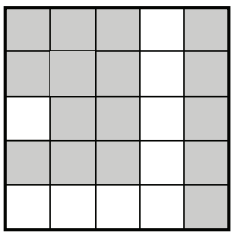
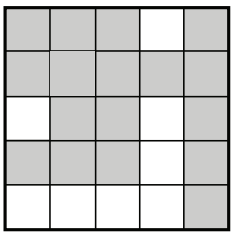
Sample Input 1

```
5 5 2
BBBOB
BOBOB
OBBOB
BBBOB
OOOOB
2 2
2 4
```

Sample Output 1

```
0
0
1
```

Explanation

The left image is the inital grid (with blue chambers represented by gray and orange chambers represented by white). It is already solvable – there exists a sequence of valid moves starting from some chamber that will visit every chamber at least once. After the first modification, the grid (the middle image) is still solvable. After the second modification, as shown on the right, the grid is no longer solvable but by reversing the modification it can be made solvable again.

Sample Input 2

```
5 6 3
OOBBBO
OOOOOO
OOBBBO
OOOOBO
BOBOBO
3 4
4 5
1 6
```

Sample Output 2

```
1
1
2
1
```

Constraints

To evaluate your solution, the judges will run your program against several different input files. All of these files will adhere to the following bounds:

- $1 \leq R, C \leq 300$
- $1 \leq Q \leq 10\,000$

For 50% of test cases, $1 \leq R, C \leq 50$ and $1 \leq Q \leq 5000$.

Furthermore, your program will be given 50% of available marks for a test case if it **correctly distinguishes between solvable and unsolvable grids** (that is, you will be marked on whether each line of output is 0 or non-zero).