SAC '22 Code Challenge 1 P3 - That Pool

Time limit: 1.0s **Memory limit:** 256M

You decided to start putting heads in your pool of blood.

You are viewing a cross-section of the pool, which is a 2D, $N \times M$ grid.

Each square in your pool can have 3 different types: an `X` that represents a floating head, a `.` that represents empty space, or a `W` that represents blood.

You can perform 2 actions:

`1`: Drop all `X` heads one cell lower: overwrite the cell below each `X` with an `X` and leave a `.` in the original position of the `X`; then, the blood moves to fill in the empty spaces.

`2`: Output the current grid.

Whenever you perform a `1` operation, the blood automatically flows, adhering to the following constraints:

- Blood can only move to empty cells (and cannot move off the grid).
- Whenever blood moves, the cell it originally occupies becomes an empty cell, and the new cell is filled with blood.
- The blood first moves to the leftmost cell that it can reach, then, if possible, drops down if the cell below it is empty and exists, and it repeats this process until it cannot move.
    - The stages of this 2-step process are applied one after another, while each stage is applied to each blood cell simultaneously.

Additionally, when an `X` head reaches the bottom of the grid and a `1` action is performed, the `X` head disappears from the grid.

Constraints

For all subtasks:

$1 \leq N, M, Q \leq 100$

Subtask 1 [10%]

There are no `W` characters in the grid.

Subtask 2 [90%]

No additional constraints.

Input Specification

The first line will contain N and M, the number of rows and columns, respectively.

The next N lines will contain M characters, representing the type of square for the pool.

The next line will contain Q, the number of actions to perform.

The next Q lines will contain one of the above queries.

Note: The first grid will always be stable (i.e., the blood will already be in the most optimal place).

Output Specification

For every `2` query, output the state of the board.

Sample Input 1

```
2 9
WWXX.XWWW
XXX...XXX
4
1
2
1
2
```

Sample Output 1

```
WW.......
WWXXWX...
WW.......
WWW......
```

Explanation for Sample Output 1

After all `X` cells lower in the first query, the current board looks like this:

```
WW....WWW
..XX.X...
```

Now, every blood cell moves to the leftmost cell it can reach:

```
WWWWW....
..XX.X...
```

Then, every blood cell moves down if possible:

```
..WW.....
WWXXWX...
```

Finally, all blood cells move to the leftmost cell again, reaching our state for the first `2` query:

```
WW.......
WWXXWX...
```

Sample Input 2

```
6 3
...
WW.
WWW
XWX
WWX
WXX
2
1
2
```

Sample Output 2

```
...
W..
WW.
WW.
XWX
WWX
```

Explanation for Sample Output 2

After all `X` cells lower in the first query, the current board looks like this:

```
...
WW.
WWW
.W.
XWX
W.X
```

Then, every blood cell moves to the leftmost cell it can:

```
...
WW.
WWW
W..
XWX
W.X
```

Now, every blood cell moves down if it can (and then each blood cell moves to the leftmost cell, but all blood cells are already in their respective leftmost cells):

```
...
WW.
W..
WWW
X.X
WWX
```

Again, every blood cell moves down if it can:

```
...
W..
WW.
W.W
XWX
WWX
```

Finally, every blood cell moves to the leftmost cell it can and stabilizes, reaching the result for the `2` query:

```
...
W..
WW.
WW.
XWX
WWX
```

Sample Input 3

```
5 5
.....
WWWX.
WXWW.
WWWWW
WXWWW
6
1
2
1
2
1
2
```

Sample Output 3

```
.....
WW...
WWWX.
WXWW.
WWWWW
.....
WW...
WW...
WWWX.
WXWWW
.....
WW...
WW...
WW...
WWWXW
```

Sample Input 4

```
2 3
W..
XXW
2
1
2
```

Sample Output 4

```
W..
W..
```

Sample Input 5

```
6 6
X.....
X...XW
..XXXX
X.XXXX
X.XXXX
X.XXXX
4
1
2
1
2
```

Sample Output 5

```
......
x.....
x...x.
w.xxxx
x.xxxx
x.xxxx
......
......
x.....
x...x.
..xxxx
x.xxxx
```