Malware (`malware`)

A malware creator needs to be able to control where a certain number of samples of the particularly harmful malware strain *av23* are located on a $n \times n$ grid-connect network of computing devices. To this aim, they need to be able to move each of them individually along the grid.

Before implementing the actual infreastructure, the malware creator needs to run some tests on a simulated grid network to check the effects of his evil plan.

One of the problems they have to simulate can be modeles as folows:

1. the grid is represented as $n \times n$ board that contains up to 10 malware samples indentified by characters between 0 and 9. Empty locations are marked with a non-digit character.
2. Each sample can be moved a certain number of times in any of the four directions: up (U), down (D), left (L) and right (R). No diagonal moves are allowed.
3. A sample can only be moved towards an empty location and cannot escape the board's boundaries.
4. When a malware sample is moved, its previous location becomes empty.

You have to stop the malware creator. To do so, you have to be able to reproduce yourself what they would do to run the simulation.

More specifically, you are given in input the size n of the board, the $n \times n$ matrix of chars, the number m of move operations and the list of m move operation. Each move operation is defined by: SAMPLE_ID (id of the sample to be moved), DIR (movement direction as U, D, L or R) and REP (number of repeated moves to be performed).

After each move operation you have to print the return code of the operation as follow:

- 0: malware sample successfully moved.
- 1: error: invalid number of repetitions (REP < 1).
- 2: error: SAMPLE_ID does not appear on the board.
- 3: error: attempt to move the sample over a non-empty location.
- 4: error: attempt to move the sample past the board's boundaries.

In case of multiple errors return the one with the lower code.

At the end of the m operations you have to print the final board.

Example:

Given a board 5×5 containing 3 samples 1, 7 and 9:

```
. . . . .
. 1 . . .
. . . 9 .
. . . . .
. . . 7 .
```

Execute the following operations:

7 U 3 (move the sample 7 up of 3 positions)

Return code 3 (moving up sample 7 eventually hits 9)

9 L 3 (move the sample 9 left of 3 positions)

Return code 0 as it is possibile to move 9 left of 3 positions, the final board is thus:

```
. . . . .
. 1 . . .
9 . . . .
. . . . .
. . . 7 .
```

Implementation

You should submit a single file, with either a `.c`, `.cpp`, `.java` or `.py` extension.

Your program must read input data from `stdin` and write the output data into `stdout`.

`stdin` consists of $1 + n + m$ lines:

- Line 1: the integers n and m, space separated, the size of the board and the number of moves operations.
- Next n lines: string of n characters, a single row of the board.
- Next m lines: SAMPLE_ID DIR REP, an integer, a character and an integer, space separated describing a move operation.

`stdout` consists of $m + n$ line:

- First m lines: return code of the m move operations, one for line.
- Next m lines: string of n characters, a single row of the final board.

Constraints

- $1 \leq n \leq 1.000$
- $1 \leq m \leq 1.000$
- DIR $\in \{U, D, L, R\}$

Scoring

Your program will be tested on a number of testcases grouped in subtasks. In order to obtain the score associated to a subtask, you need to correctly solve all testcases of which it is formed.

- **Subtask 1** [**20 points**]: $n, m \leq 10$.
- **Subtask 2** [**30 points**]: $n, m \leq 100$.
- **Subtask 3** [**50 points**]: $n, m \leq 1.000$.

Examples

stdin	stdout

```
5 2
.....
.1...
...9.
.....
...7.
7 U 3
9 L 3
```
	```
	3
	0
	.....
	.1...
	9....
	.....
	...7.
	```