There is a robot which is located in a grid of unit squares. Some of the squares are free and some of the container obstacle. The robot always occupies one of the free squares and faces one of the four directions (n, s, e, w). The robot understands two commands: “move forward” and “turn left.” Also it understands simple conditional and looping statements. The main educational potential of the language lies in the possibility of defining new procedures for more complex tasks. The language can be described by the following grammar:

```
<program>   :=  "" | <simple-command> <program>

<simple-command>   :=   "m" | "l" | <procedure-call> |

                              "i" <condition> "(" <program> ")(" <program> ")" |

                              "u" <condition> "(" <program> ")"

<condition> :=  "b" | "n" | "s" | "e" | "w"

<procedure-call> :=  <uppercase-letter>

<procedure-definition>  :=  <uppercase-letter> "=" <program>
```

There are five types of simple-commands:

- m (“move forward”) advances the robot in the position by one grid square in its current direction. If the next cell is blocked then the command has no effect.
- l (“turn left”) turn left 90 degrees. n (north)→w (west) → s (south) → e (east)
- X where X is any uppercase letter, executes procedure named X.
- i (“if” statement) followed by a condition, and two programs in parentheses. If the condition is satisfied, then the first bracket is executed, otherwise the second one.
- u (“until”) followed by a condition, and a program in parentheses. If the condition is satisfied, nothing is done. Otherwise, the program is executed and then the command is repeated.
- Conditions can be
  - ‘b’, which is satisfied if and only if there is a barrier in the next square in robot’s current heading,
  - one of the four directional letters ‘n’, ‘s’, ‘e’, or ‘w’, which is satisfied if and only if robot’s current heading is north, south, east, or west, respectively.

For example, a simple program ib(l) can be understood to mean: “if blocked turn left” while us(l) means “turn to the south.” A procedure definition M=ub(m)us(l) defines a new procedure ‘M’ which means move until the wall and turn south in the end.

The format of the data is the following:

All input and output should be done from standard input/output.

In:

The first line of input has four integers r, c, p, and q, where r and c ($1 \le r, c \le 40$) size of the 2D grid where the robot operates, p ($0 \le p \le 26$) is the number of defined procedures, and q ($1 \le q \le 10$) is the number of programs to be executed.

Then there are “r” lines describing the operational area (2D grid) from north (first row) to south (last row), with “c” characters from west (first character) to east (last character). Each character is a free space ‘.’ or blocked cell ‘#’ . The robot cannot leave the area, which means when robot heading north in the first row it sees a barrier.
Each of the next “p” lines have procedure definitions. Each line defining one procedure assigning an upper case letter as a procedure name and specifying a program forming the procedure body. All the procedures are distinct.
The last 2*”q” lines describe programs to run on a robot. The first line of each pair contains initial position of the robot in the 2D grid. The position is specified by 2 integers “x”,”y” (1 ≤ x ≤ r; 1 ≤ y ≤ c) positions by row and column and direction “d” ∈ {n, s, e, w} represents robots direction north, south, east, west. The robot is always located in a free space. The second line of the program pair describes the program to run on a robot.
There is no white spaces in the programs or procedures.
The procedures bodies and programs have at least 1 and at maximum 100 characters and syntactically correct.
Out:
For each program to be run on a robot, output the position of the robot after execution. If the program never finishes then output “inf”.

For example:
In:
```
4 8 4 6
.......#
..#....#
.###...#
.....###
G=ub(B)
B=ub(m)lib(l)(m)
H=ib()(mmHllmll)
I=III
1 1 w
G
1 1 e
G
2 2 n
G
4 1 s
ib(lib()(mmm))(mmmm)
1 1 e
H
2 2 s
I
```
Out:

```
1 1 w
inf
1 1 w
4 4 e
1 4 e
inf
```