890 Maze (II)

Johnny likes solving puzzles. He especially likes drawing and solving mazes. However, solving a maze he has drawn himself is too easy for him.

Since his computer is his best friend, he figures that he needs a program drawing the mazes for him. So he starts thinking about an algorithm performing this difficult task for him and he comes up with "Johnny's Simple Algorithm."

Johnny's Simple Algorithm

You start with a $M \times N$ grid, where M is the number of rows and N is the number of columns of the grid. Initially, no two cells of the grid are connected to each other, so every cell is surrounded by walls on all four sides. The walls consist of an underscore ('_') for a horizontal wall, and a vertical bar ('|') for a vertical one. For example, if $M = 3$ and $N = 4$, the grid looks like this:

```
 _ _ _ _
|_|_|_|_|
|_|_|_|_|
|_|_|_|_|
```

Every cell of the grid has unique coordinates (p, q). The lower left corner in the example is $(1, 1)$, the upper right corner is $(3, 4)$.

After choosing the dimensions of the maze, you choose a starting cell. From now on you keep track of a list of *pending* cells, which initially contains only one cell (the starting cell), and you repeat the following steps:

1. If the list is empty, you stop. The maze is ready.
2. Else, you consider the most recently added cell in the list (call this cell AC). If this cell (at the end of the list) has no *unvisited* neighbor cells then you remove this cell from the list. Every cell has at most 4 neighbor cells: on the right, left, above and below. A cell is unvisited if it has never been added to the list.
3. If AC has at least one unvisited neighbor cell, you choose one of the unvisited neighbor cells (call this cell NC), remove the wall between AC and NC and add NC to the end of the list.

Johnny makes a nice little program using this algorithm and it works fine, but Johnny is not completely satisfied with the results. He is a demanding little boy and in his opinion the so-called *branching factor* of the maze is too low, i.e. the generated mazes contain very long paths and too few crossings. Therefore, the mazes are still too easy to solve for him.

A little trick can be applied to Johnny's Simple Algorithm, giving much better results. Johnny does not know it, but you will, since it will be explained below!

The idea behind the trick is to sometimes change the order of the cells in the list. This avoids long paths and results in more branches. Changing the order of the cells in the list is done by 'flipping' part of the list. A *flip* can be specified by giving the position of a cell in the list (where the first cell has position 1) and consists of reversing the sub-list starting at the specified cell and ending with the last cell in the list.

For example, if the list consists of the following cells:

$$(1, 1)(1, 2)(2, 2)(3, 2)(3, 3)$$

then a flip with starting cell number 2 results in:

$$(1,1)(3,3)(3,2)(2,2)(1,2)$$

Now, we will reveal "Johnny's Advanced Algorithm."

Johnny's Advanced Algorithm

The algorithm is pretty much the same as "Johnny's Simple Algorithm," only sometimes part of the list is flipped. The steps you repeat after choosing the dimensions of the maze, choosing the starting cell and adding this cell to the list are:

1. If the list of cells is empty, you stop. The maze is ready.
2. Else you consider the last cell in the list. If this cell has no unvisited neighbor cells, then you remove this cell from the list.
3. Otherwise, you read a command. If this command is:

   '`F` n' you flip the list, starting at position n.

   '`U`' you go up: you remove the wall between the last cell in the list and the cell above it. The cell above the last cell in the list is added to the list.

   '`D`' you go down.

   '`L`' you go left.

   '`R`' you go right.

Since you are taking part in a programming contest, we ask you to write a program generating nice mazes for Johnny, using "Johnny's Advanced Algorithm," to make him happy again. The maximum size of a maze is 39×39.

Input

The first line of the input contains the number of test cases. The input for every test case is divided into three parts:

- The first line contains two integer values M and N, specifying the dimensions of the maze: the number of rows M followed by the number of columns N.
- The second line contains the coordinates of the starting point (again, row followed by column).
- The next lines each contain a command. A command is one of the upper case characters '`F`', '`U`', '`D`', '`L`' and '`R`', appearing at the start of a line. An '`F`' character is followed by a space and an integer (the starting position of the flip.)

The input for each test case contains exactly the number of commands needed for that maze.

Output

The resulting mazes should be printed using spaces ('` `'), underscores ('`_`'), vertical bars ('`|`') and end-of-line characters. No unnecessary whitespace should be printed. The mazes should be followed by one blank line.

Sample Input

```
2
3 3
1 1
U
U
R
D
D
R
U
U
3 4
2 1
R
U
L
F 2
R
U
R
D
D
F 4
D
L
L
```

Sample Output

```
 _ _ _
|   | |
| | | |
|_|_ _|

 _ _ _ _
|_  |   |
|_ _   | |
|_ _ _|_|
```