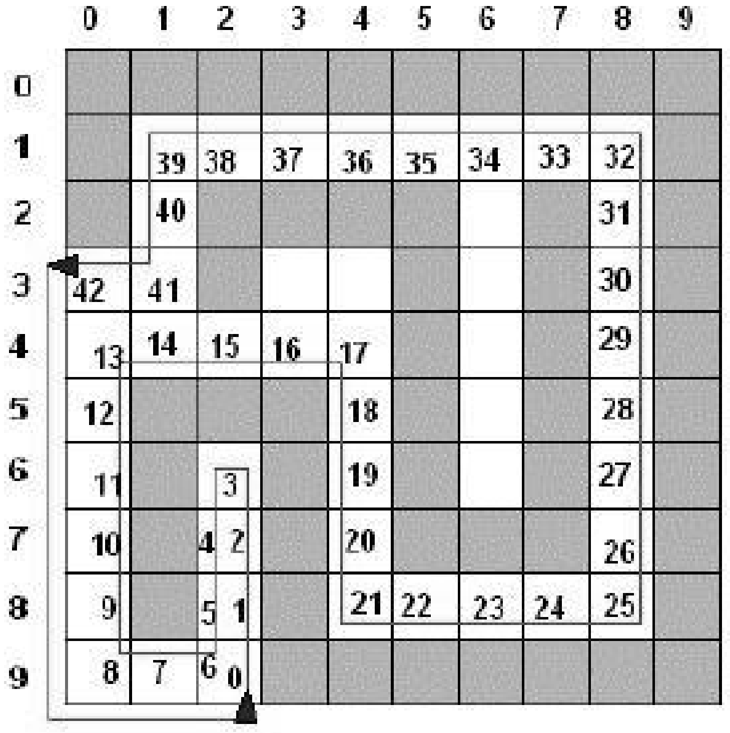
10796 Right Hand Rule

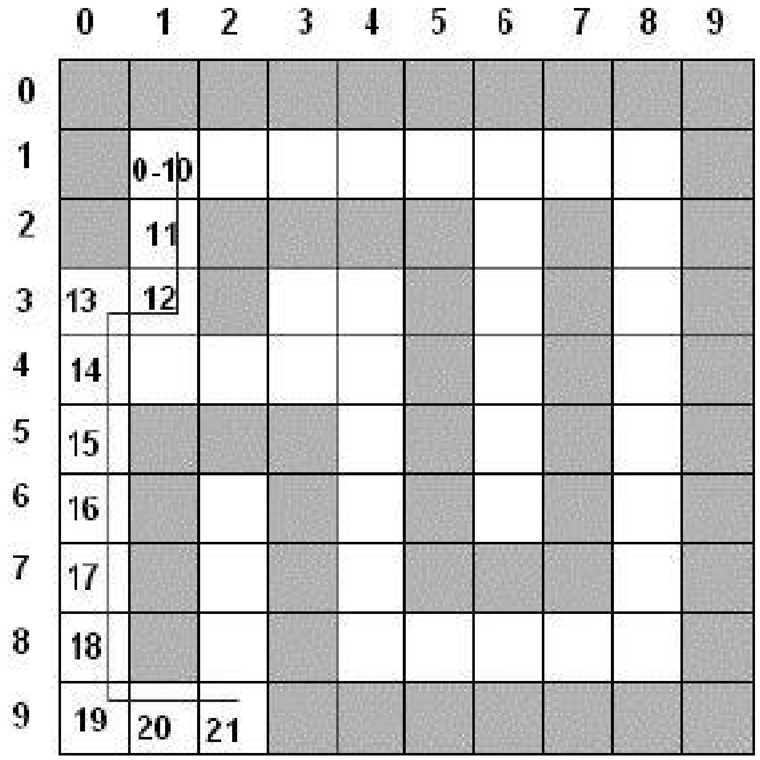
Mazes are one of the most popular forms of puzzles. The numerous branches and dead ends of intricately constructed mazes can put the patience and problem-solving skill of the solver to a challenging test. But even for a budding programmer, writing a computer program that explores a maze is a trivial task. However, the addition of a rather "not so intelligent" agent to the maze can really make it delicate.

There are many strategies that one can employ while exploring a maze. Marking off the cells, and carrying a ball of thread are two such strategies. There is yet another strategy which we call the "Right Hand Rule". This rule suggests that the solver keeps the right hand in constant contact with one of the walls as he proceeds. This method guarantees that if there is an exit, the solver will be able to reach it, if it is reachable from the cell he started. Otherwise, it would take him through every cell that is reachable from his starting cell. This may not be the best idea when one would want to go from a cell to another cell in the maze. But if we would like to put a programmed agent to keep exploring the maze, this rule can be quite useful.



In this problem, we have one such programmed agent, called "**Agent K**", that is continuously exploring a rectangular maze. **Agent K** can move in four directions, **UP**, **DOWN**, **LEFT** and **RIGHT**, but cannot move through the walls. **K** has its right hand in constant contact with the wall. It goes through the cells of the maze taking one unit time to make a move. It does not take any time to make a turn in the same cell, though. It may happen that **K** finds an exit from the maze and falls out. But that does not end **K**'s exploration. The moment it falls out, it is re-spawned in the cell from where its journey began.

The accompanying picture illustrates this scenario. In a 10 × 10 rectangular maze, Agent **K** is placed in cell (9,3) facing **UP**. The impermeable walls are gray, and the passable cells are white. **K** follows the right hand rule and moves along the path in the red line. At time 42, **K** is at cell (3,0) – so at time 43, **K** is supposed to be out of the maze. But instead of falling out, **K** reappears at cell (9,2) at time 43. For some mazes it may happen that **K** never finds an exit. In those cases, **K** will have a closed path, which in which it would loop forever.



Now your task is to find the shortest path from a given starting cell to a designated destination cell. You must avoid **Agent K** at all cost. You may not be in the same cell as **K** at the same time as him, and you may not walk to a cell if **K** is walking from that cell to your cell (then you would be caught in the corridor if you and **K** approach each other from opposite direction). You can move in the same four directions as **K**, and you both make your moves simultaneously. Additionally, you may choose to not move, and wait one unit of time for agent **K** to do its next move.

Let us illustrate this scenario in the same sample maze. If you are to go from cell (1,1) to cell (9,2) in shortest possible time, one approach could be waiting at cell (1,1) till time 10 and then start moving along the path shown in blue. There are of course, other ways to go from (1,1) to (9,2) but the minimum possible time required remains to be 21. You only need to compute the minimum time, not the path.

Input

The input consists of several test cases. The first line of the input file contains a single integer, T $(1 \leq T \leq 25)$ indicating the number of test cases. Then T test cases follow. The first line of each of the test case would give you the number of rows, R $(1 \leq R \leq 20)$ and the number of columns, C $(1 \leq C \leq 20)$ of the maze. In the next line you would find three pairs of numbers and a letter. The pairs are given in row, column format representing your starting cell, destination cell and agent **K**'s starting cell respectively. The letter at the end of this line can be '`U`', '`D`', '`L`', or '`R`' giving you the direction **K** is facing. So '`1 1 9 2 9 2 U`' would essentially mean that you are at cell (1,1), you need to go to cell (9,2), and agent **K** is at cell (9,2) and is facing **UP**. It is guaranteed that there will be a wall to the right hand side of agent **K**. In the next R lines, the maze description is given. A '`#`' in a cell represents an impermeable wall, whereas a '`.`' would mean that the cell is passable. Cell (0,0) is the upper left corner of the maze.

Output

The output for each test case starts with the test case number in the format '`Maze` x`: `' where $1 \le x \le T$. Then there would be exactly one integer denoting the minimum unit time required for you to reach the destined cell; if it is impossible to reach the destined cell, output '`-1`' instead.

Sample Input

```
3
10 10
1 1 9 2 9 2 U
##########
#........#
#.####.#.#
..#..#.#.#
.....#.#.#
.###.#.#.#
.#.#.#.#.#
.#.#.###.#
.#.#.....#
...#######
5 5
0 0 4 3 4 3 U
.....
##.#.
....#
.##.#
....#
5 5
0 0 2 3 3 3 U
.....
#####
.#..#
.#..#
.####
```

Sample Output

```
Maze 1: 21
Maze 2: 15
Maze 3: -1
```