Snake

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

This is an interactive problem. Remember to flush the output buffer after every print. To flush your output, you can use:

- `fflush(stdout)` or `cout.flush()` in C/C++;
- `System.out.flush()` in Java and Kotlin;
- `sys.stdout.flush()` in Python.

You are playing the classic snake game on a grid of size $n \times m$. The rows are numbered 1 to n from top to bottom, and the columns are numbered 1 to m from left to right. We denote the cell at row i and column j as (i, j).

The snake can be represented as a sequence of coordinate pairs that determine where its body is located: $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$. Here, k denotes the length of the snake. The head of the snake is at (x_1, y_1), the tail is at (x_k, y_k), and neighboring parts of the body are located in cells that share a side.

Initially, the snake has length 1 and is located at a given cell (r_s, c_s).

There are 4 types of commands to move the snake:

- `U`: Command the snake to move one step up. The head will then move to $(x_1 - 1, y_1)$.
- `D`: Command the snake to move one step down. The head will then move to $(x_1 + 1, y_1)$.
- `L`: Command the snake to move one step left. The head will then move to $(x_1, y_1 - 1)$.
- `R`: Command the snake to move one step right. The head will then move to $(x_1, y_1 + 1)$.

When the head moves, each part of the body also moves accordingly. Specifically, the i-th part of the body ($2 \le i \le k$) moves to the position where the $(i-1)$-st part was before the command. Meanwhile, the cell previously occupied by the tail becomes empty.

The snake cannot move outside the grid. Besides, the snake cannot collide with itself — you must guarantee that no two parts of the body share the same cell after any command. Consider the following corner case: the head is at (x_1, y_1), and the tail is at (x_k, y_k). If the head is moving to (x'_1, y'_1), then it is allowed that $(x'_1, y'_1) = (x_k, y_k)$: if we think about a real-world scenario, the head moves into the cell just as the tail moves out. In a similar fashion, it is allowed to swap the head and the tail by using a single command when $k = 2$.

There are $(nm - 1)$ apples that will appear one at a time. Each time, an apple appears at some cell (r, c) that is not currently occupied by the snake. You need to output a sequence of commands that moves the snake's head to the apple. The head must arrive at (r, c) exactly on the last command, and it must not enter (r, c) before that.

For each sequence of commands you output, every command except the last one is a normal move: the tail vacates its cell as described above. The last command in the sequence is an eating move: the snake eats the apple when the head arrives at (r, c). On this move, the tail does NOT vacate its cell, so the length of the snake increases by 1.

Your task is to successfully eat all $(nm - 1)$ apples.

Input

There are multiple test cases. The first line of the input contains an integer T ($1 \le T \le 100$), indicating the number of test cases. For each test case:

The first line contains four integers n, m, r_s, and c_s ($2 \le n, m \le 50$, $nm \le 100$, $1 \le r_s \le n$, $1 \le c_s \le m$), indicating the size of the grid and the starting position of the snake.

Interaction Protocol

You need to eat all $(nm - 1)$ apples one by one. For each apple:

- Read a line containing two integers r and c ($1 \le r \le n$, $1 \le c \le m$), indicating the position of the apple. It is guaranteed that the position of the apple is not currently occupied by the snake.
- Output a string consisting of the characters `U`, `D`, `L`, `R` — the sequence of commands that moves the snake's head from its current position to (r, c). To make this problem harder, we constrain the length of the string to be at most nm.

After printing each line, make sure to flush the output. It can be shown that an answer always exists.

Example

standard input	standard output
2 2 3 1 2 2 1 1 3 2 1 1 1 1 2 2 2 1 1 2 2 1 1 1 2	 LD URR DLL U R RD LU R

Note

The following image illustrates the first sample test case after eating 3 apples.