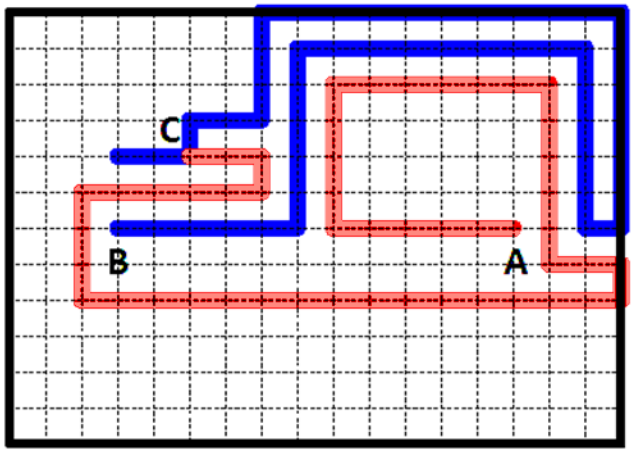
Problem 3 – Trails 3D

Trails is a simple reaction game where players race on a grid, leaving colored trails behind them; any player who touches the trails is eliminated, and the last player alive is the winner.

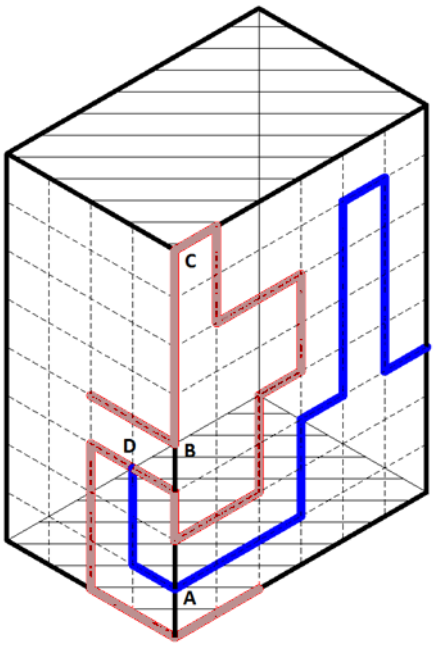


In the example, the two players start at points A and B respectively, moving on the grid and turning left and right, trying to surround each other. In point C, the red player crashes into the blue player's trail and loses the game.

Trails 3D is similar, with the players racing on the surface of a 3D parallelepiped instead of inside a rectangle.

There are two players, **starting from the centers of two opposing walls** of the cube, and **turned in the same direction** (towards one another). They move on a grid on the surface, and **in every game cycle, they move one position in their current direction**; before every move, they can turn left or right (turning is instant and doesn't count as taking a cycle of game time).

In the example, the two players start at the centers of the front and back faces; they race on several walls of the cube, until the red player surrounds the blue player, and the blue player crashes in point D.



Additional rules for movement:

- two of the walls of the cube are **forbidden** – a player who tries to move on one of these walls crashes and loses the game; the forbidden walls are opposite one another (on the picture they are marked with diagonal lines);
- when a player reaches an edge of the cube, **he continues moving on the next wall**, in the same direction (see point A on the example);
- a player can **move on the edge of the cube** (see segment B-C); when he reaches a corner of the cube, he must turn left or right (see point C).

The game ends when one or both of the players crash. If both players crash on the same game cycle, the game is a draw; otherwise, the one who didn't crash wins. Your program will read a sequence of moves from the console, and **determine the winner** and the **distance between the start and endpoint** of the red player, **along the grid** (in this case, 8 – 4 down, and 4 along the bottom edge).

Input

The input data should be read from the console. On the first line, you will read three integers - X, Y and Z - representing the dimensions of the cube. X and Y represent the dimensions of the walls on which the players start, X and Z are the dimensions of the forbidden walls, and Y and Z are the dimensions of the other two walls. The players start in the center of the two opposite X*Y walls, and move in the direction of edge Y (towards each other; see example input 2 below).

On the second and third line you will read two strings of characters representing the motion of the red and blue players respectively. The motion is represented as a string of 'M', 'L' and 'R' characters, where M means "move without turning, L means "turn left", and R means "turn right". 'M' character may have a number prefix. This number prefix shows how many "move without turning" actions must be performed.

The input data will be correct and there is no need to check it explicitly.

Output

The output data should be printed on the console.

On the first output line you should print “RED”, “BLUE” or “DRAW”, depending on who won the game.

On the second line, print the distance between the start and end points of the red player, measured along the playing grid, as an integer (if the red player crashes into a forbidden wall, his final position is the point where he crashed).

Constraints

- The numbers **X**, **Y** and **Z** are positive even integers in the range [2…50].
- The motion strings will be between 2 and 120 characters long (only characters M, L and R).
- The length of the motion strings will be long enough to finish the game.
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Example input (example on picture)	**Example output**
8 4 6 2MLM1MRM2MR2MLMLMR3MRM LMMR2M4MRMLMRMR1M2MRM	BLUE 9

Example input (players move without turning and crash into each other)	**Example output**
4 2 4 3M1M 2M1M1M	DRAW 4